Correctness-Aware High-Level Functional Matching Approaches For Semantic Web Services

A thesis submitted for the degree of
Doctor of Philosophy

Islam M. Elgedawy
M.Sc. (Alexandria University, Egypt)
B.Sc. (Alexandria University, Egypt)

School of Computer Science and Information Technology,
RMIT University,
Melbourne, Victoria, Australia.

29^{th} August, 2006

Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; and, any editorial work, paid or unpaid, carried out by a third party is acknowledged.

Islam M. Elgedawy
School of Computer Science and Information Technology
RMIT University

29^{th} August, 2006

For the ones who dare to be different and choose what is difficult.

Acknowledgments

While offering my deepest gratitude to Prof. Dr. Zahir Tari and Dr. James Thom. Prof. Tari is my first supervisor who provided all the needed support to finalize this thesis. Dr. Thom is my co-supervisor whose meticulous reviews always guided me to the right direction. Also I would like to thank Dr. Michael Winikoff and Dr. Gregory Craske for their participation in the production of the thesis. Dr. Winikoff was my co-supervisor in the preliminary stages of the thesis, while Dr. Craske reviewed the first draft of the thesis. Finally, I would like to thank Mrs. Dalia Moussa for her help in analyzing the freight forwarding case study.

Credits

Portions of the material in this thesis have previously appeared in the following publications:

- I. Elgedawy, Z. Tari, and J. Thom. A high level functional matching for semantic web services[1]. In *Proceedings of the third International Conference on Service Oriented Computing (ICSOC)*, pages 115−129, Amsterdam, Netherlands, 2005.
- I. Elgedawy, Z. Tari, and M. Winikoff. Scenario matching using functional substitutability in web services. In *Proceedings of the fifth International Conference on Web Information Systems Engineering (WISE)*, pages 59−65, Brisbane, Australia, 2004.
- I. Elgedawy, Z. Tari, and M. Winikoff. Exact functional context matching for web services. In *Proceedings of the second International Conference on Service Oriented Computing (ICSOC)*, pages 143−152, NY, USA, 2004.
- I. Elgedawy. A conceptual framework for web services semantic discovery. In *Proceedings of On The Move (OTM) to meaningful internet systems*, pages 1004−1016, Catania, Italy, November 2003.

The thesis was written using the `WinEdt 5.3` editor on Windows XP, and typeset using the LaTeX 2_ε document preparation system.
All trademarks are the property of their respective owners.

[1]This paper won the best paper award for a Ph.D level in ICSOC05.

Contents

Abstract **1**

1 Correctness-Awareness Requirements **4**

1.1 Web Services Matching 7

1.1.1 Limitations of Existing Service Matching Approaches 10

1.2 Research Questions 11

1.3 Thesis Contributions 13

1.4 Thesis Organization 16

2 Towards Semantic Web Services Matching **18**

2.1 Matchmaking versus Brokering 18

2.2 Ontologies 20

2.3 Goal-Oriented Requirements Engineering 22

2.4 Service Representation 23

2.4.1 UDDI 24

2.4.2 E-speak 24

2.4.3 LARKS 25

2.4.4 EbXML 25

2.4.5 DAML-S/OWL-S 26

2.4.6 WSMO 26

2.5 Service Direct Matching 27

2.6 Service Composition versus Service Aggregation 33

2.7 Service Composite Matching 34

3 SWSMF:A Semantic Web Services Matching Framework 38

3.1 Preliminaries 38
3.2 Limitations of Existing Matching Frameworks 43
3.3 SWSMF Approach 46
3.4 A Meta-Ontology for Modelling Application Domains 52
3.4.1 The Schematic Layer 53
3.4.2 The Semantic Layer 56
Concepts Substitutability Graph 56
3.5 The G^+ Goal Model 61
3.6 From a G^+ Model to a GAP-Forest 68
3.6.1 GAP Correctness 71
3.7 Conclusion 75

4 The Functional Substitutability Matching Scheme 77

4.1 Preliminaries 77
4.2 FSMS Components 79
4.2.1 Functionality as a Comparison Aspect 80
4.2.2 Substitutability as a Matching Rule 85
4.2.3 Goal Achievement as a Correctness Criterion 100
4.3 Behavior Models Matching using FSMS 103
4.3.1 State Matching 104
4.3.2 State Expansion 107
4.3.3 State Sequence Clustering 117
4.4 Conclusion 124

5 Correctness-Aware Service Matching Approaches 126

5.1 Service Direct Matching Approach 126
5.2 Service Aggregate Matching Approach 137
5.2.1 High-Level Functional Context Aggregation 141

5.2.2 GAP Aggregate Matching 147
5.2.3 GAP Forest Aggregation 153
5.3 Service Matching Techniques Evaluation 154
5.3.1 Service Direct Matching 157
Imprecise Service Matching Approaches 158
5.3.2 Behavior Models Direct Matching 162
5.3.3 Service Aggregate Matching 165
5.4 Conclusion 166

6 Conclusion **167**
6.1 Future Work 169
6.1.1 Flexibility and Performance Enhancement 169
6.1.2 Service Selection 169
6.1.3 Service Discovery 169
6.1.4 Service Composition 170

A SWSMF Realization **171**

Bibliography **177**

List of Figures

1.1 Intelligent Web Services Vision (Adapted from [Fensel and Bussler, 2002]) . . 5
1.2 Components of The Service Matching Process 7
1.3 Web Service Matching Architecture 16

2.1 Matchmaking versus Brokering (Adapted from [Decker et al., 1996]) 19

3.1 Web Services Specifications Classification 42
3.2 Approaches for Application Domain Representation 47
3.3 G^+ Conceptual Model 48
3.4 SWSMF Conceptual Service Model 51
3.5 SWSMF Conceptual User Model 51
3.6 Meta-Ontology Layers 52
3.7 OMM for the edge from `Credit.Period` into `Payment.Type` 58
3.8 A CSG Representation 59
3.9 Relationships between the G^+ Model and the Meta-Ontology 62
3.10 Example of a User G^+ Model 64
3.11 Example of a Service G^+ Model 65
3.12 Example of a GAP 68
3.13 Example of a GAP Forest 70
3.14 GAP Transformation Function 71

4.1 The Matching Scheme Architecture 78
4.2 Relation between the Different Types of Constraints 82

4.3 Direct versus Indirect Constraint Satisfiability 87
4.4 Constraint Indirect Satisfiability via Scopes Direct Reachability 89
4.5 Indirect Satisfiability Decision Tree 93
4.6 One-to-One State Sequence Matching 107
4.7 Expansion Direction versus Merge Direction 108
4.8 Consecutive States Merge 109
4.9 Consecutive Operations Merge 110
4.10 f_{x+2} Elements 111
4.11 State Merge Example 112
4.12 f_m^i Elements 113
4.13 Source Down Expansion 114
4.14 Clusters Restructuring Operation 115
4.15 Example of Target Reverse Expansion 116
4.16 Target Down Expansion 117
4.17 SMP Invocation Effect 118
4.18 SMP Trace 122
4.19 SMP Result when Source and Target are Switched 123

5.1 The Service Direct Matching Approach 127
5.2 Anticipated User Behavior 128
5.3 Service Direct Matching Example 132
5.4 The Anticipated and the Required Behaviors after Applying SMP 134
5.5 Strict versus Loose GAP Forest Matching 136
5.6 An Aggregate Service 137
5.7 A Plug-in GAP and the corresponding Computation Chunk 138
5.8 Chunk List 139
5.9 Array of Stacks 140
5.10 Accepted versus Rejected Aggregate Patterns 140
5.11 High-level Functional Context Aggregation 141
5.12 Describing-Constraints Aggregation 142

5.13 GAP Aggregation Approach 147
5.14 A Request GAP and its corresponding Plug-in GAPs 150
5.15 Aggregation Process Trace 151
5.16 GAP Forest Aggregation 153
5.17 Simulation Experiments Plot 157
5.18 Simulated Comparison between Service Direct Matching Approaches 161
5.19 Comparison between Operations Sequence Matching Approaches 164
5.20 Direct versus Aggregate Semantic Matching 166

List of Tables

1 List of Acronyms xiii
2 List of Symbols xiv

1.1 Semantic Web Services versus Conventional Web Services 8

2.1 Direct Service Matching Classification 28
2.2 Composite Service Matching Classification 35

3.1 Comparison between Existing Matching Frameworks 44
3.2 Substitutability Graph Segment for *CargoTransportation* Operation 60
3.3 User Perspective versus Service Perspective 63
3.4 Part of the Ontology Operations' Definitions Selected by the User 67
3.5 Part of the Ontology Operations' Definitions Selected by the Service Provider 67
3.6 HLFC Matching 68
3.7 Comparison between SWSMF and Existing Matching Frameworks 76

4.1 Persisting Constraints Distribution 83
4.2 User Required Functionality 85
4.3 Goal Achievement Operator Properties 102

5.1 User's Behavior Model 133
5.2 Service's Behavior Model 133
5.3 Anticipated Behavior Model 134

List of Algorithms

1 Transformation Formation via Scopes Direct Reachability 92
2 Transformation Formation via Scopes Indirect Reachability 95
3 Constraint-Set Substitutability 99
4 SMP Matching Case Handling 119
5 Sequence Mediator Procedure (SMP) 121
6 Describing-Constraints Divider 143
7 Describing-Constraints Aggregator 144
8 GAP Aggregator 149
9 GAP-Forest Aggregation 154

Abbreviation	Meaning
API	Application Program Interface
BOV	Business Operational View
CFS	Context Flat Scope
CMM	Constraints Mapping Matrix
CRO	Clusters Restructuring Operation
CSS	Context Structured Scope
DAML-S	DARPA Agent Markup Language Services Ontology
FRM	Functionality Representation Matrix
FSMS	Functional Substitutability Matching Scheme
FSV	Functional Service View
GAP	Goal Achievement Pattern
HLFC	High-Level Functional Context
IDF	Inverse Document Frequency
ILT	Inland Transportation
INCOTERM	International Commercial Terms
LARKS	Language for Advertisement and Request for Knowledge Sharing
LLFC	Low-Level Functional Context
NFC	Non-Functional Context
NTC	Non-Technical Context
POL	Port Of Loading
POD	Port of Discharge
OWL	Web Ontology Language
QoS	Quality of Services
RDF	Resource Description Framework
SAV	Service Active View
SCPL	Service Composition Planning Language
SMP	Sequence Mediator Procedure
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-oriented computing
SPV	Service Passive View
SVD	Singular Value Decomposition
SWSMF	Semantic Web Services Matching Framework
TF-IDF	Frequency-Inverse Document Frequency
UAV	User Active View
UDDI	Universal Description, Discovery, and Integration
WSDL	Web Services Description Language
WSMF	Web Service Modelling Framework
WSMO	Web Services Modelling Ontology
URL	Universal Resource Locator

Table 1: List of Acronyms

Notation	Meaning
$Cnst_j$	Constraint j
$Ctxt$	An instance of HLFC
$Ctxt^{Pre}$	The set of $Ctxt$ pre-constraints
$Ctxt^{Post}$	The set of $Ctxt$ post-constraints
$Ctxt^{Desc}$	The set of $Ctxt$ describing-constraints
Op_x	Operation x
Op_x^{Pre}	The pre-constraints of Op_x
Op_x^{Post}	The post-constraints of Op_x
G^+	Extended Goal Model
$\vDash$	Direct Satisfiability Operator
$\gg_G$	Indirect Satisfiability Operator with respect to goal G
$\in$	Disjunctive Bounding Constraint Operator
$\subseteq$	Conjunctive Bounding Constraint Operator
λ	Comparative Constraint Operator
$\trianglerighteq_G$	Substitutability with respect to the achievement of a goal G
$\urcorner$	A transformation
$\vdash$	Goal Achievement Operator
$\diamond$	The Semantic Difference Operator
$\downarrow_G$	Source state expandable with respect to G
$\uparrow_G$	Source state reversely expandable with respect to G
$\Downarrow_G$	Target state expandable with respect to G
$\Uparrow_G$	Target state reversely expandable with respect to G
$\preceq_G$	Plug-in operator with respect to goal G
f_x	A set of Persisting Constraints
f_x^i	The Idle Constraints of f_x
f_x^e	The Effective Constraints of f_x
S_x	State x
Srv_x	Service x
β	A Behavior Model
ρ_s	Sequence Similarity Ratio
Δ	The set of all possible constraints
$\Xi(Cnst)$	Scope of a constraint Cnt
$\Xi(f_x)$	The scope of a set of constraints f_x
$\Xi(S_x)$	The scope of a state S_x

Table 2: List of Symbols

Abstract

Today we are experiencing the emergence of a new computing paradigm that changes the way future applications and systems will be designed, architected, delivered and consumed. Service-oriented computing (SOC) is this new emerging computing paradigm that enables technologies for developing and connecting future applications in different areas, such as e-commerce, e-science, telecommunications, and bioinformatics.

Evolving from object-oriented and component computing, SOC utilizes published services as the fundamental elements for developing and dynamically connecting different collaborating applications and systems, which are distributed within and across organizational boundaries. One of the major challenges in SOC is to locate these published services without a-priori knowledge of their existence. This is known as the service discovery problem. Part of the service discovery problem is the service matching problem, in which the descriptions of the published services are examined against users' requests in order to find the right services that fulfill these requests.

Existing service matching approaches trade precision for recall, creating the need for humans to manually choose the correct services from the matching results (that fulfill users' goals). This need for human intervention is a major obstacle for automating both the service discovery and the service composition processes. To overcome this problem, we argue that the matchmaker must be able to automatically determine the correctness of the matching results without the need for human intervention. Such correctness is a user-based correctness that needs to be determined with respect to the defined users' goals. To achieve correct matching, we have identified the following obstacles:

- The high-level functional specifications (namely goals, roles, contexts and expected external behavior) for both users and services need to be semantically captured in a machine-understandable format, in order to enable the matchmaker to understand them and later use them to find the correct answers.

- The high-level functional semantics of application domains need to be explicitly captured in a machine-understandable format such that the matchmaker can use them to mediate between users' requests and services' descriptions.

- Services need be primarily aggregated according to their high-level functional specifications, taking into consideration that the descriptions of the component services could be described using different concepts.

- Formal matching schemes need to be adopted during the matching process to indicate how the captured semantics will be used to determine the correct matching results.

In this thesis we propose solutions to these obstacles. First, we introduce a Semantic Web Services Matching Framework (SWSMF), that indicates which types of semantics need to be captured and how these semantics will be represented for both services and users in a machine-understandable format. Within this framework, we have developed an extended goal model (G^+ model) to capture the *goal-based* high-level functional specifications (namely, goals, contexts and external behaviors). To overcome the semantic interoperability problem, we propose a meta-ontology approach for modelling application domains that captures application domains' concepts and operations. It also captures the functional substitution semantics of application domains' concepts using the proposed concepts substitutability graph, that enables the matchmaker to determine the concepts' functional equivalence based on the involved users' goals and contexts; avoiding the use of any rigid concept taxonomies. This enables the mediation process during the matching of services' high-level functional specifications to be based on users' goals and contexts, which minimizes the appearance of false negatives and eliminates the false positives. Second, we propose a new matching scheme for matching the high-level functional specifications of services and users that is called the

Functional Substitutability Matching Scheme (FSMS). FSMS indicates how goals, roles, contexts and behaviors of web services will be semantically matched. Finally, adopting FSMS, we propose two new correctness-aware service matching techniques: a direct matching technique and an aggregate service matching technique; taking into consideration that services' descriptions and users' requests could be described using different concepts. The results of the conducted simulation experiments indicate that the proposed matching techniques succeed to eliminate the appearance of false positives, and minimize the appearance of false negatives compared to the existing major service matching approaches. By adopting the proposed matching framework, matching scheme, and service matching techniques (direct and aggregate), we believe that we have established the first step to achieve a fully automated service matching process.

Chapter 1

Correctness-Awareness Requirements

The World Wide Web (WWW) has become a key infrastructure for information exchange due to its easy and fast accessibility, its scalability, and its inexpensive global connectivity. This has attracted a huge number of commercial activities toward using the WWW as their infrastructure for communication and interaction. Thus, a new computing paradigm known as *Service Oriented Computing (SOC)* [Papazoglou and Georgakopoulos, 2003] has appeared to fulfill these interests. SOC enables computerized objects, systems, processes and applications to be seen as services that can interact with each other in order to accomplish any required goals.

Unfortunately, the WWW is designed "only for humans" when it comes to publishing and accessing information. Computers cannot understand the provided information and in return provide limited support for processing this information. Therefore, hard-coded processes are the only way to enable the interactions between existing systems; which is a very expensive and time consuming process. Due to the cost and time barriers of these technologies, there is a lack of flexible, dynamic and scalable usage of existing systems, which hinders new solutions and systems to be easily developed using existing systems. To overcome such a problem, two complementary approaches appeared to transform the WWW from being for humans only into a WWW for humans and computer systems: *Semantic Web* [Berners-Lee et al.,

2001] and *Web Services* [Alonso et al., 2004]. The semantic web is about adding machine-understandable semantics to data such that computers can understand the information and therefore process it on behalf of human users. Web services provide facilities for integrating various solutions and systems. However, in order to achieve a dynamic and intelligent WWW, both semantic web and web services need to be integrated, producing what is known as *Intelligent Web Services* [Fensel and Bussler, 2002]. Intelligent web services can automatically interact and discover each other without any human intervention. Figure 1.1 depicts such vision graphically.

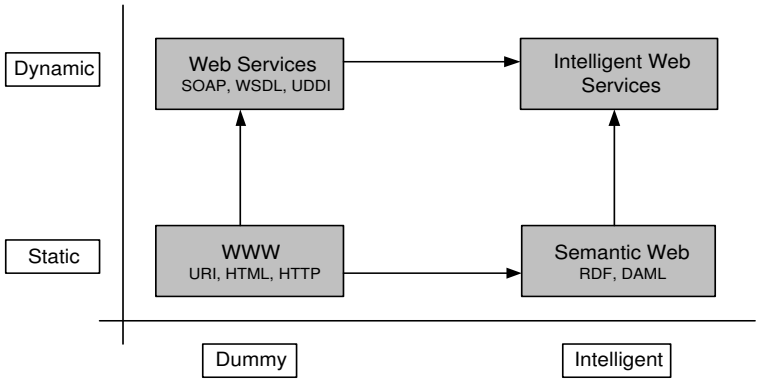

Figure 1.1: Intelligent Web Services Vision (Adapted from [Fensel and Bussler, 2002])

One of the key challenges for achieving the intelligent web services vision is to locate published services without a-priori knowledge of their existence. This is known as the *service discovery problem*. Part of the service discovery problem is the *service matching problem*, in which the descriptions of the published services are examined against users' requests[1] to find the right services that properly fulfill these requests. An *answer* to a user's request could be a service or a group of services that can fulfill this request. A *correct answer* is either a service or a group of services that, when invoked, can properly achieve the required user's goal. A *matchmaker* is a program designed to accept users' requests and return answers to these requests by applying a given service matching technique over a given service registry. When a matchmaker examines one service description against a user's request, this is known as *Service Direct Matching*. While, when it examines a group of services' descriptions against

[1]In this thesis, we use the term user to represent both humans and systems.

a user's request, this is known as *Service Aggregate Matching*.

Web services are widely adopted by many application domains such as e-business, e-science, telecommunications, and bioinformatics. *Logistics* is one of the application domains that can be easily formalized and universally unified. *RosettaNet* [Sundaram and Shim, 2001; Dogac et al., 2002] is an example of such efforts proposed to standardize the supply chain process. We believe that the logistics application domain can be fully automated via web services, and therefore we will be using a logistic case study to illustrate the concepts and techniques proposed in this thesis. Example 1 indicates a typical real-life scenario in the logistics application domain that shows the limitations of the existing technologies to find services on the WWW, which motivates our work.

Example 1 (Motivating Example) *A company would like to automate its business process. This company wants to find a freight forwarder service that can handle its cargo transportation. The company needs a specialist in transporting cars, that can deal with freight on board transactions, allows credit payments, finalizes custom clearance, and can handle shipments from Melbourne-Australia to Alexandria-Egypt. The company wants to pay not more than 10% commission and wants the freight forwarder to have more than 6 million dollars turn over per year and less than 48 hours response time. Also the company wants to negotiate any proposals before executing the shipment order. Currently, the developer of the company has to exhaustively search through the results returned by a given search engine (that adopts keyword-based matching techniques) to find a service that satisfies these requirements, which can be a very time consuming process. Then the developer has to carry out the necessary procedures to integrate the chosen service(s) with the existing systems, taking into consideration that services' descriptions do not necessarily use the same vocabulary adopted in the query, may not contain the required information or may represent the required information differently.*

Based on the specifications of Example 1, the developer will need to decide about the correctness of the matching results by manually checking services' descriptions one by one. This obviously represents a major obstacle for automating the matching process.

A solution for this problem is to enable the matchmaker to determine the correctness of the results without the need for any further human intervention, and without executing any of the matching services. Thus, the aim of this thesis is to propose new concepts, models and techniques enabling the matchmaker to automatically determine the correctness of the matching results without the need for human intervention as well as without the need to execute any matching services. This will provide a new level of open and flexible automation between web services, as they can be discovered and composed on-the-fly with low programming costs, minimizing the cost and the time for service integration and interaction.

1.1 Web Services Matching

To enable the matchmaker to automatically determine the correctness of the matching results, we argue that the semantics of web services, users, and application domains must be captured in a machine-understandable format such that they can be understood by the matchmaker. Additionally, the matchmaker needs to know how to use such semantics to find the correct services. In other words, the service matching process needs to be designed such that the matchmaker understands both the components of the matching process and how they interact with each other to find the correct services. Hence, we argue that the service matching process should be based on four basic components: service semantics, user semantics, application domain semantics and matching schemes, as indicated in Figure 1.2.

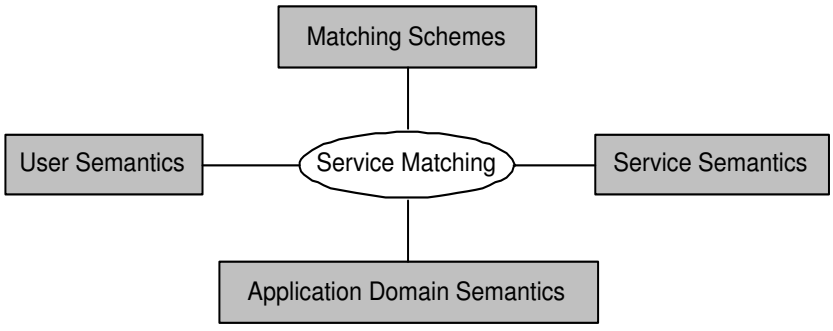

Figure 1.2: Components of The Service Matching Process

Service Semantics: This models the semantics obtained from the descriptions of web services. Such descriptions contain the captured services' specifications. Having such specifications represented in a machine-understandable format enables the matchmaker to reason about these services. Services with machine-understandable specifications are known as *Semantic Web Services*[2] [Bussler, 2003; Rajasekaran et al., 2004]. A comparison between conventional web services and semantic web services is given in Table 1.1 (adapted from Abecker et al. [2004]).

Comparison Aspect	**Conventional Web Services**	**Semantic Web Services**
Service Requester	Human	System/Human
Service Description	Text-based	Ontology-based
Service Matching	Syntactically-Based	Semantically-based
Service Discovery	Manually	Automatically
Service Composition	Manually	Automatically
Complexity	Simple	Complex

Table 1.1: Semantic Web Services versus Conventional Web Services

As a service could have different types of specifications that could be involved in the matching process (for example, functional and nonfunctional specifications), identifying the types of specifications that need to be captured and how they will be represented are crucial issues for the service matching process.

User Semantics: As the focus is on the user-based correctness, fulfillment of a user's desire is the only reference that a matchmaker needs to take into consideration to determine the correctness of the matching results. Consequently, services that cannot fulfill a user's desire must be rejected. A user's desire must be captured in the user's request via listing all the required types of specifications. The types of specifications required by the user must be compatible with the types of specifications captured in services' descriptions. Also they need to be captured in a machine-understandable format such that the matchmaker can understand them.

[2]Intelligent web services are semantic web services that can automatically discover and collaborate with each other.

It is important to differentiate between two different types of services when capturing a user's desire: *Transaction Services* and *Information Services*. A transaction service affects and changes the real world status by performing actual tasks. `Cargo transportation` and `flight booking` services are examples of such services. Whereas an information service is used to report the real world status (such as a service that lists currency exchange rates) and/or to list and browse transaction services (such as a service that lists flight reservation services). Transaction services and information services are completely different as they represent different desires, semantics and specifications. Hence, users must clearly state in their requests the types of specifications that are relevant to the required services.

Application Domain Semantics: Users' requests are not expected to be exactly the same as services' descriptions, therefore a mediation process between users' requests and services' descriptions is required. One of the factors that affects the mediation process is the captured semantics of the application domain involved [Bussler, 2003]. Hence, the elements of the application domains and their corresponding relations need to be identified, captured, and represented. Capturing such information (in a machine-understandable format) is crucial for the matching process as this enables the matchmaker to automatically mediate between users' requests and services' descriptions.

Matching Scheme: A matching scheme indicates how services will be matched with respect to a given type of specifications. It also indicates how the mediation between services' descriptions and a user's request will be performed adopting the semantics of the involved application domain. A matching scheme describes how the captured semantics of services, users, and application domains will be used to determine the correctness of the matching results with respect to a given type of specifications. Therefore, the matchmaker needs to know which matching schemes will be adopted in the matching process.

1.1.1 Limitations of Existing Service Matching Approaches

Unfortunately, existing service matching techniques (such as those described by Chakraborty et al. [2001], Bernstein and Klein [2002], Paolucci et al. [2002b], Sajjanhar et al. [2004], Dong et al. [2004], and Sycara et al. [2004]) treat services as documents when it comes to service matching and retrieval, as they trade precision for recall. In other words, they try to maximize the number of returned services; hoping they contain the correct services, and leave the responsibility of choosing the correct services to the user. We argue such approaches cannot be adopted in the automated service matching process, as they require human intervention to determine the correctness of the matching results.

Existing service matching techniques examine the descriptions of web services by eliciting various concepts appearing in them such as inputs, outputs and involved entities. Such elicited concepts are later matched using either a syntactic approach [Sycara et al., 2002; Sajjanhar et al., 2004; Dong et al., 2004], or in a more precise semantic approach [Castillo et al., 2001; Paolucci et al., 2002b; Ganesan et al., 2003; Roddick et al., 2003; Patil et al., 2004; Roman et al., 2005].

Matching the elicited concepts using a syntactic approach (such as TF-IDF keyword matching technique [Salton, 1991]) is an imprecise and rigid approach known to have poor *precision* and *recall* [Bernstein and Klein, 2002; Paolucci et al., 2002b], as it assumes users' requests and services' descriptions will be based on the same vocabulary, neglecting the need for any mediation techniques as well as ignoring the semantics of services, users, and the involved application domains. Matching the elicited concepts using a semantic matching approach takes into consideration the need for mediation between the elicited concepts. A common characteristic of the semantic matching approaches is that elicited concepts are matched using concept subsumption rules that could be provided by either generic domain taxonomies [Resnick, 1995; Weinstein and Birmingham, 1999; Ganesan et al., 2003; Roddick et al., 2003; Patil et al., 2004], or use of a form of logic [Kashayp and Sheth, 1996; Castillo et al., 2001; Paolucci et al., 2002b; Keller et al., 2004; Roman et al., 2005].

We argue that use of generic domain taxonomies to match the elicited concepts is not a precise approach as it completely ignores the achievement of users' goals as well as users'

contexts (there is no perfect taxonomy that can suit all contexts and goals). Furthermore, only using the schematic characteristics of the adopted taxonomy (for example, the number of edges between two concepts) is not sufficient to determine the equivalence between the involved concepts [Kashayp and Sheth, 1996]. We also argue that the use of logic to determine concepts' subsumption is not a practical approach as the current reasoners are not computationally powerful enough to handle complex domain subsumption relations. For example, current Description Logic reasoners such as FaCT [Horrocks, 1998] and RACER [Haarslev and Möller, 2001] do not adequately leverage the powerful expressiveness of the Description Logic as they are not scalable, not dynamic, and do not support multiple entity interconnection [Castillo et al., 2001].

Existing aggregate service matching approaches (such as those described by McIlraith et al. [2001], Bultan et al. [2003], Pistore et al. [2004], and Berardi [2005]) assume that all services' descriptions and users' requests adopt the same vocabulary, and they require the behavior models' traces to have the same number of states in order to be matched. Such aggregate approaches are very rigid as they do not adopt any mediation techniques between the services' descriptions and users' requests, which indeed is not a practical assumption that leads to the appearance of false negatives and false positives. Furthermore, they adopt internal behavior of web services during the matching process. We argue the external behavior should be adopted instead, as use of internal behavior violates services' encapsulation.

1.2 Research Questions

The main goal of this thesis is to provide the necessary concepts, models and techniques required to deal with service matching automation. Our approach for service matching automation is to enable the matchmaker to automatically determine the correctness of the matching results with respect to the defined users' goals such that the need for human intervention to determine the correct results can be discarded. To achieve this objective, the matchmaker must be able to do the following tasks:

- Decide which services can achieve the required users' goals without the need for human intervention or the need for executing the matching services.

- Mediate between services' descriptions and users' requests using the semantics of the involved application domains to determine the correct services, as services' descriptions and users' requests could be described using different concepts.
- Aggregate different services, as a user's request could not be fulfilled by one service.

We strongly argue that services must be primarily matched according to their high-level functional specifications (namely, goals, roles, contexts, and external behaviors), as this allows the mediation process to be performed in a high-level manner where the semantics of application domains can be used. Later, matching using other types of specifications (such as prices, quality of service parameters and access interfaces) could be performed to refine the matching results returned from the primary stage. As this thesis focuses on the matching of the high-level functional specifications, the following core research questions are addressed:

1. How to represent the high-level functional specifications of semantic web services in a format that is both human-understandable and machine-understandable such that they can be easily described by both humans and machines in a systematic manner, taking into consideration that users' requests and services' descriptions could be described using different concepts?
2. Which types of application domains semantics need to be captured in order to mediate between the high-level functional specifications of both semantic web services and users' requests, and how these semantics will be represented in a machine-understandable format such that they can be understood and used by the matchmaker to find the correct services, taking into consideration that users' requests and services' descriptions could be described using different concepts?
3. How to mediate between the high-level functional specifications of services and users' requests using the captured application domains' semantics in order to automatically determine the correct services that guarantee the achievement of users' goals; without the need for human intervention as well as the need for executing the matching services?

4. How will web services be aggregated according to their high-level functional specifications taking into consideration that component services could be described using different concepts, that also could differ from the concepts adopted by users?

1.3 Thesis Contributions

This thesis proposes a new approach for increasing the accuracy of the service matching process. The approach uses the high-level functional semantics of services, users, and application domains to determine the correctness of the matching results. It automatically verifies the services according to the achievement of users' goals. That services that only achieve the users' goals are considered correct and returned to the users. We summarize our research contributions to the service matching process as follows:

1. A new goal model for capturing the goal-based high-level functional specifications of semantic web services (namely goals, contexts, and external behaviors), known as the G^+ model [Elgedawy, 2003]. This model extends the existing goal-scenario-coupling approaches [Rolland et al., 1998; Rolland and Achour, 1998], used for goal modelling, to link between contexts, goals, and scenarios, in order to capture the goal-based high-level functional specifications of semantic web services. In the G^+ model, a goal's context describes the requirements and effects of the goal achievement via three sets of constraints: pre-constrains, post-constrains, and describing constraints, respectively. The G^+ model also links between the different scenarios for achieving the goal using the proposed *scenarios-network* that is used to describe the major external behavior patterns of the service. The G^+ model is an ontology-based model that is both machine-understandable and human-understandable. Unlike existing goal-scenario coupling approaches, by adopting the G^+ model, the matchmaker will be able to automatically determine whether a given service can achieve a user's goal or not. That is done by checking if such service can transform the required pre-constraints into the required post-constraints using one of the required scenarios; satisfying the required describing-constraints.

2. A meta-ontology for application domains [Elgedawy, 2003; Elgedawy et al., 2004b]. The proposed meta-ontology approach compromises between the multi-ontology approach and the consensus approach used for handling the semantic interoperability problem. It maintains the modelling flexibility of the multi-ontology approach while simplifying the ontology mapping process by having a common structure for application domains' ontologies. The *schematic layer* of the meta-ontology captures the concepts and operations of application domains. The *semantic layer* captures the concepts' functional substitutability semantics using the proposed *concepts substitutability graph* to determine the semantic equivalence between domain concepts. Unlike existing approaches for determining concepts' semantic equivalence [Castillo et al., 2001; Paolucci et al., 2002b; Ganesan et al., 2003; Roddick et al., 2003; Patil et al., 2004; Roman et al., 2005], the proposed approach determines the concepts' equivalence according to the semantics of users' goals. That two concepts are considered equivalent with respect to a given goal, and the same two concepts are considered nonequivalent with respect to another goal. This will eliminate the false negatives and the false positives resulting from adopting the existing approaches for determining concepts' semantic equivalence that use either a fixed taxonomy or fixed subsumption rules for determining concepts' semantic equivalence; ignoring the semantics of users' goals.

3. A new constraint matching approach, known as the constraint substitutability approach [Elgedawy et al., 2004b; 2005]. The powerfulness of this approach comes from its ability to semantically match constraints with different scopes[3]. This will eliminate the false negatives resulting from adopting the existing constraint matching approaches (such as the ones discussed by Jeavons and Cooper [1995]; Pearson and Jeavons [1997]) that require the constraints to have the same scope in order to be matched. The constraint substitutability approach uses the concepts substitutability graph to find a transformation that maps the source constraint into an intermediate constraint such that the intermediate constraint has the same scope as the target constraint. Then matches the intermediate constraint to the target constraint using the existing con-

[3]A scope is an attribute and its corresponding concept appeared in the constraint.

straint matching approaches. The constraint substitutability approach is one of the mediation techniques used in both context matching and behavior matching of services.

4. A new behavior matching approach for semantic web services [Elgedawy et al., 2004a; 2005]. The uniqueness of this approach comes from its ability to semantically match linear state sequences that consist of different number of states. This will eliminate the false negatives resulting from adopting existing behavior matching approaches [Magee et al., 1997; Pierce and Sangiorgi, 2000; Findler et al., 2001; Berardi, 2005] that require the state sequences of the behavior models to have the same number of states in order to be matched. The proposed approach defines a behavior state based on the persisting constraints at the corresponding transition points, then uses a sequence mediator procedure to re-cluster the state sequences with different number of states into new state sequences with the same number of states. The sequence mediator procedure uses different state expansion operations to find the matching clusters. Finally, it matches the expanded behavior states based on their persisting constraints adopting the constraint substitutability approach.

5. A new high-level functional aggregate matching approach for semantic web services. This approach is a dynamic sequential aggregate approach that is based on services' G^+ models. The flexibility of this approach comes from its adoption for the proposed constraint substitutability and behavior matching approaches. The strength of this approach comes from its ability to aggregate services described using different concepts. This eliminates the false negatives resulting from adopting the existing aggregate approaches as they require all component services and users' requests to be described by the same concepts.

Figure 1.3 depicts the architecture needed to support the proposed matching approaches. The figure indicates that:

- The first layer of the automated service matching process is the meta-ontology that indicates how application domains will be modelled.

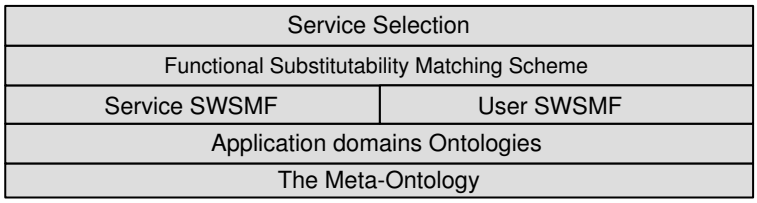

Figure 1.3: Web Service Matching Architecture

- The second layer is the ontologies of application domains that provide the vocabulary used in building services' descriptions and users' requests. These ontologies must adopt the proposed meta-ontology.

- The third layer is SWSMF that indicates how web services' descriptions and users' requests will be modelled using the ontologies of application domains (the G^+ model is part of this framework).

- The fourth layer is the Functional Substitutability Matching Scheme (FSMS) that indicates how services will be matched (directly and aggregately) according to their high-level functional specifications, also it indicates how the proposed constraint substitutability and behavior matching mediation approaches will be realized to mediate between services' descriptions and users' requests.

- The fifth layer is the service selection layer, in which services (that is returned after applying the proposed service matching approaches) will be filtered according to other types of specifications such as services' price and quality using other matching schemes.

As the main focus of this thesis is the high-level functional semantics, other types of specifications are captured and matched using existing approaches, leaving the investigation about these issues for future work.

1.4 Thesis Organization

The rest of this thesis is structured as follows.

Chapter 2 (Towards Semantic Web Services Matching): This chapter provides the technical background of the concepts used in this thesis and discusses the major research efforts in the area of service matching.

Chapter 3 (A Semantic Web Services Matching Framework): This chapter discusses the required characteristics to be adopted by a service matching framework. It shows how the proposed framework (SWSMF) models both the semantic web services' descriptions and the users' requests, then introduces the proposed G^+ model and the proposed meta-ontology. Finally, it provides a criteria for the correctness of the G^+ models.

Chapter 4 (The Functional Substitutability Matching Scheme): This chapter proposes a matching scheme for matching the high-level functional specifications of semantic web services, known as the Functional Substitutability Matching Scheme (FSMS). It presents the constraint substitutability approach, providing the corresponding data structures, theoretical proofs, and the realization algorithms. Finally, it proposes the behavior matching approach that adopts an m-to-n state matching.

Chapter 5 (Dynamic Service Matching Approaches Adopting FSMS): Adopting the mediation techniques proposed by FSMS, this chapter proposes two new service matching techniques: a direct matching technique and an aggregate matching technique, then evaluates the devised service matching techniques using simulation experiments.

Chapter 6 (Conclusion): This chapter concludes the thesis with a discussion of the achieved developments, then highlights some areas for future work.

Appendix A (SWSMF Realization): This appendix discusses the required steps to realize SWSMF from the user's perspective, the service provider's perspective, the ontology engineer's perspective, and the matchmaker's perspective.

Chapter 2

Towards Semantic Web Services Matching

This chapter surveys the basic principles and concepts for matching semantic web services according to our perspective. As our service matching process adopts a matchmaking approach in order to respond to users' requests, Section 2.1 investigates the difference between the matchmaking and brokering approaches for fulfilling users' requests. Section 2.2 introduces ontologies, as they are adopted by our approach during the modelling and the matching processes. Section 2.3 provides a quick overview of the characteristics of goal-oriented requirements engineering, as we require explicit models for capturing the goals of web services and users. Section 2.4 summarizes the major approaches for service representation and highlights their limitations. Section 2.5 summarizes the major approaches used for service direct matching. Section 2.6 investigates the difference between service composition and service aggregation. Finally, Section 2.7 gives an overview of the major approaches for service composite matching.

2.1 Matchmaking versus Brokering

Service oriented architecture is the commonly accepted minimal framework for service oriented computing [Papazoglou, 2003], in which a service requester asks for services, a service

provider develops and advertises services, and discovery agencies match users' requests with the advertised services in order to fulfill these requests. Generally, users' requests could be fulfilled via either matchmaking or brokering processes [Decker et al., 1996], as indicated Figure 2.1.

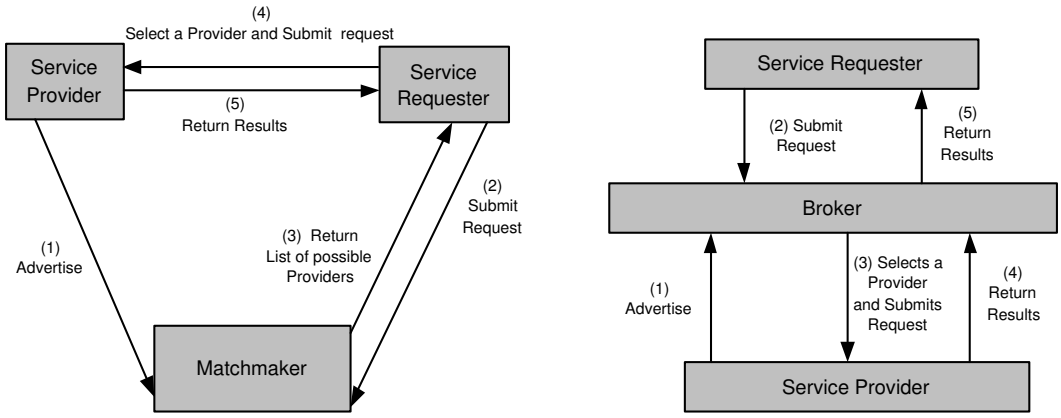

Figure 2.1: Matchmaking versus Brokering (Adapted from [Decker et al., 1996])

In the matchmaking process, a service requester submits a query to the matchmaker, then the matchmaker searches its service registry for services that could fulfill the requester's query by applying a given matching technique, then returns the results to the requester to choose the suitable service provider. It is the responsibility of the service requester to contact the service provider in order to execute the required service and obtain results.

In the brokering process, a service requester submits a query to the broker, and then the broker searches a service registry for services that could fulfill the requester's query by applying a given matching technique. The broker then chooses one of the service providers using a given selection criteria that generally depends on execution environment characteristics (such as load balancing, bandwidth and latency), and then communicates with the service provider to obtain the results, and then returns the results to the service requester.

The main difference between matchmaking and brokering is that the service provider is hidden from the service requester during the brokering process. Also, a global optimization

for the execution environment could be obtained via brokering as the broker keeps track of the execution environment status, and it could apply execution monitoring and recovery techniques to guarantee the best utilization of the resources as well as enhancing the overall performance of the system.

2.2 Ontologies

Berners-Lee et al. [2001] proposed the semantic web in order to enable machine-understandable access for information, in which information is defined using machine-understandable semantics (known as the meta-data). Ontologies[1] represent the part of the semantic web architecture that is concerned with *domain conceptualization* such that a shared common understanding of a domain can be communicated across people, applications, and systems [Guarino, 1997; Lassila and Swick, 1999; Patel-Schneider and Fensel, 2002]. A domain conceptualization is an abstract view of the domain to be represented. An ontology should include descriptions of domain entities and their existing relations, as well as specify any attributes of domain entities and their corresponding values. An ontology can range from a simple taxonomy, to a thesaurus (words and synonyms), to a conceptual model (more complex relations are defined), to a logical theory (formal axioms, rules, theorems, and theories are defined). Ontologies should overcome any incompatibility problems that might appear during domain conceptualization process such as [Kashayp and Sheth, 1996]:

- **Entity Definition Incompatibility**: This problem arises when the entity descriptors used for the same entity are not compatible.
- **Attribute Domain Definition Incompatibility**: This problem arises when attributes have different domain definitions.
- **Data Value Incompatibility**: This problem arises when there exists inconsistent values between semantically related data.

[1]The term ontology is borrowed from philosophy, where an ontology is a systematic account of existence [Kalfoglou and Schorlemmer, 2003].

- **Abstraction Level Incompatibility**: This problem arises when the same entity is represented at different levels of abstraction.
- **Model Discrepancy**: This problem arises when the same object is represented differently in different models, for example in one model it is represented as an entity and in another model it is represented as an attribute.

An ontology might include axioms that constrain the interpretation of an entity, but this is dependent on the intended use of the ontology. When creating an ontology, the following characteristics should be taken into consideration [Gruber, 1995]:

- **Clarity**: The definitions given by an ontology should communicate effectively without depending on a social or computational context.
- **Coherence**: An ontology should be coherent that inferences obtained from the definitions must be consistent with the definitions.
- **Extendibility**: When an ontology is evolving, the new definitions should not violate any existing definitions. In other words, an ontology needs to be monotonically built.
- **Minimality**: An ontology should not have redundant definitions; the defined terms should be sufficient to support the intended knowledge needing to be shared. Also an ontology should make a minimal number of claims as possible.
- **Commitment**: The terms defined by an ontology should cover all the aspects required to be shared by an ontology.

There are two approaches that can be followed when ontologies are adopted in the matching process: the *consensus approach* [Benjamins et al., 2004], and the *multi-ontology approach* [Kalfoglou and Schorlemmer, 2003]. The consensus approach implies that every application domain is described only by one ontology, while the multi-ontology approach implies that multiple different ontologies could be used to describe the same application domain. Indeed, reaching consensus for every application domain conceptualization will overcome the semantic interoperability problem, but this is far from feasible. Although the multi-ontology

approach provides flexibility for application domains conceptualization, it requires ontology-mapping process to be adopted.

Ontology mapping provides a common layer from which several ontologies could be accessed and used to exchange information in a semantically sound manner. However, ontology mapping is a complex process as different types of entities and different semantics are used by different ontologies. The main idea of ontology mapping is to identify entities in different ontologies that are semantically related in order to resolve any appearing differences. Resolution of such differences are mainly handled by conversion functions, which until now are manually developed [Kalfoglou and Schorlemmer, 2003]. Current approaches for automatic ontology mapping such as the approaches addressed by Kalfoglou and Schorlemmer [2003] mainly determine the similarity of elements based on predefined classification or taxonomies. However, purely schematic considerations do not suffice to determine the similarity between entities as they are generic and ignore the involved contexts [Kashayp and Sheth, 1996].

We differentiate between a domain ontology and a web service ontology. A *domain ontology* is concerned with conceptualizations of application domains, while a *web service ontology* is concerned with a description of web services that defines what should be captured in a service model and how it should be represented.

2.3 Goal-Oriented Requirements Engineering

As a web service has similar characteristics to a computer-based system, we argue that goal-oriented requirements engineering approaches could be adopted to model services' goals.

Requirements engineering is the part of software engineering that is concerned with the development of requirements for computer-based systems [Kotonya and Sommerville, 1998]. However, developing requirements for computer-based systems is associated with several problems such as neglecting systems contexts, and poor understanding of application domains' semantics [Green, 2004]. Therefore, goal-oriented approaches such as [Dardenne et al., 1993; Loucopoulos and Kavakli, 1995] are proposed to overcome such problems. These approaches try to describe the system's behavior and structure; driving the requirements based on the analysis of the derived behavior and structure. However, there are some problems that

are related to the goal-oriented approach since the concept of a *goal* is intentional by nature. That it sometimes becomes hard to identify, characterize and decompose a goal, which leads to a limited requirement representation [Green, 2004].

An alternative approach to requirements engineering is capturing requirements via scenarios [Potts et al., 1994]. Scenarios provide powerful means to understand complex systems by capturing examples, illustrations and use cases. Since scenarios describe real situations, they capture real requirements. However, scenarios only provide partial and restricted requirements descriptions that are based on the captured examples, illustrations and use cases. In spite of partialness and incompleteness of scenarios, they are sufficient to express a majority of services' external behavior [Weidenhaupt et al., 1998; Uchitel, 2003].

As goals are intentional by nature and lack an operational point of view, and scenarios can provide an operational description for the requirements, various research efforts in the requirement-engineering field (such as [Rolland et al., 1998; Rolland and Achour, 1998; Dardenne et al., 1993; Green, 2004]) propose coupling goals with scenarios to capture the requirements. However, we will extend such approach in Chapter 3 to link between goals, scenarios and contexts.

2.4 Service Representation

Services are not just middleware components that can be easily described via RPC interfaces such as work in [Bernstein, 1996; Lee and Helal, 2003]. However, services could represent more complex systems such as business processes [Casati et al., 2000; Schuster et al., 2000]. This creates a need for more advanced models to describe web services such that they can be easily discovered and composed automatically [Staab et al., 2003]. Currently, there are different frameworks for representing services, from different perspectives, such as UDDI, OWL-S, and WSMO. Every framework has its own understanding of what is a service, what a service does, how a service should be consumed, and how a service should be managed. Consequently, there is no common understanding about what a service means and what constitutes a service [Dumas et al., 2001; Berardi, 2005]. In the following subsections, we indicate the most well known frameworks used for describing services.

2.4.1 UDDI

UDDI (Universal Description, Discovery, and Integration) [UDDI-Technical-Committee, 2004] ultimate goal is to have a universal centralized registry for web services. In UDDI, service description is conceptually divided into three categories known as *white pages*, *yellow pages*, and *green pages*. The while pages contain general contact information about service providers. The yellow pages contain classification of web services based on predefined standard taxonomies. The green pages contain technical information about services. That is captured via specification pointers that point to any additional information about services such as their WSDL files and how they can be invoked. UDDI provides different collections of APIs (Application Program Interfaces) for publishing and enquiring, but it does not specify any services' semantics to be captured, however there are some efforts to add semantics to UDDI descriptions (such as Paolucci et al. [2002a] and Pokraev et al. [2003]), while other approaches adopt application domain semantics during the service matching process (such as Lu [2005] and Luo et al. [2005]).

2.4.2 E-speak

E-speak [HP, 1999] uses the notion of *resource* to represent a ubiquitous service over the network such as files, printers, java objects, or legacy applications. A resource is described using an attribute-value approach, where attributes' names are controlled via a pre-defined vocabulary. This pre-defined vocabulary is created through a vocabulary builder service provided by the E-speak framework. Every service provider could define a new vocabulary or use existing ones. Service providers register their resources with the Core (a common registry), where resource contacts (that is, interfaces) and access control information are stored as well. Users' requests are submitted to the Core in form of search recipe. A search recipe indicates which resources a user needs and what should be done in case of multiple resources are found. However, E-speak did not survive due to many implementation limitations.

2.4.3 LARKS

LARKS (Language for Advertisement and Request for Knowledge Sharing) [Sycara et al., 2002] is a language for describing agents' capabilities. As an agent could be seen as a service [Huhns, 2002], languages used for describing agent's capabilities could be adopted to describe services.

LARKS uses a structured description for an agent's capability that includes its inputs, outputs, types, contexts, constraints, and functionality. An agent's context is described via a set of keywords, and an agent's functionality is described via a text description. Vocabulary used in agent description is defined via an private ontology (called the `Concept`) that specifies the vocabulary for this particular agent. LARKS does not provide means for describing agent behaviors. However, it does emphasize on the importance of explicitly capturing the context of an agent, and for sure using only keywords for describing an agent's context is not enough for capturing contexts' semantics. Furthermore, currently use of public ontologies is proliferating, it is more likely that vocabulary used for an agent description will be provided by public ontologies rather than incorporated as an ontological description within the agent description itself.

2.4.4 EbXML

EbXML [Hofreiter et al., 2002] provides a framework for business electronic data interchange. EbXML infrastructure components include: collaborative protocol profile, registry and repository. The collaborative protocol profile defines XML data structures that describe what each trading partner supports such as communication processes, security requirements, and contact information. Each service in the ebXML registry has two views: a Business Operational View (BOV) and a Functional Service View (FSV). The BOV addresses the business related information such as business agreements, arrangements, obligations, and requirements. The FSV addresses the technical related information such as service interfaces and data transfer infrastructure interfaces. There is no formal specification for describing services in ebXML, as a service is mainly described via different sets of business documents such as UML diagrams included in the BOV and another set of technical documents included in the FOV such

as WSDL files. Consequently, only keyword-based approaches could be used for the service matching process.

2.4.5 DAML-S/OWL-S

DAML-S (DARPA Agent Markup Language Services Ontology) is an ontology for describing web services' specifications. DAML-S is based on the DAML + OIL language [Fensel et al., 2001] but recently its new specification is based on OWL (Web Ontology Language). This new version of DAML-S is known as OWL-S [OWL-Services-Coalition, 2003]. OWL-S describes a web service via three components: a service profile, a service model, and a service grounding.

- The service profile includes service provider information, service functionality information, and a set of features. Service functionality is described by a text description for its purpose, service inputs, service outputs, and service pre-conditions. Service features are described in an attribute-value format.
- The service model captures the internal behavior of a web service via a process model. An ontology for building such a process model is provided. The process is described via its inputs, outputs, preconditions, effects, and sub-processes.
- The service grounding provides service access details such as message formats, serialization, transport, and addressing issues. WSDL over SOAP is the approach adopted for realizing the grounding model.

There are many pitfalls appeared during OWL-S implementation [Balzer et al., 2004; Mika et al., 2004], as OWL-S is loosely designed and no guidelines are given for its realization. Furthermore, describing a service purpose via a text description is not a suitable approach when automation for service matching is required, as it leads to use of keyword-based approaches.

2.4.6 WSMO

WSMO (Web Services Modelling Ontology) [Roman et al., 2005] is an alternative technology to OWL-S. WSMO is based on WSMF (Web Service Modelling Framework) [Fensel and

Bussler, 2002] and consists of: ontologies, goals, mediators, and services. An ontology consists of concept definitions, relation definitions, axioms, non-functional properties, and mediators. A goal description consists of non-functional properties (chosen from the adopted ontology non-functional properties), used mediators, post-conditions, and effects. A mediator is used to link heterogeneous components involved in the modelling of a web service. It should define the mappings and the transformations between the linked elements. A service is described by its pre-conditions, assumptions, post-conditions, effects, and the used mediators. Services are linked to goals via mediators. WSMO supports use of multiple grounding models with the same web service (unlike OWL-S, which allows only one grounding model). F-logic is used in WSMO to express the logical expressions defined by goals, mediators, ontologies and services. However, WSMO lacks explicit representation for web services behaviors (both internal and external). Also there is no information about how the mediators should be build. However, WSMO emphasizes the importance of explicitly capturing the services' goals in a machine-understandable format.

2.5 Service Direct Matching

We classify existing direct service matching approaches to identify the main similarities and differences between existing direct service matching techniques in which a user request is examined against one service description at a time. A service matching technique uses different filters in order to obtain the final matching results. We classify the matching approaches according to the adopted primary filter as indicated in Table 2.1.

Existing service direct matching approaches are either structure-based or non structure-based. The structure-based approaches differentiate between the different types of service specifications during the matching process, while the non structure-based approaches do not. Only keyword-based matching techniques are used when a non structure-based approach is adopted. However, these keywords could be matched syntactically such as the techniques adopted by Wang et al. [2003]; Sajjanhar et al. [2004]; Colgrave et al. [2004], or semantically such as the technique adopted by Paolucci et al. [2002a]; Ganesan et al. [2003]; Pokraev et al. [2003].

<table>
<tr><td rowspan="6">Service Direct Matching</td><td rowspan="2">Non Structure-Based</td><td>Syntactic</td><td></td><td></td><td>Wang et al. [2003]
Sajjanhar et al. [2004]
Colgrave et al. [2004]</td></tr>
<tr><td>Semantic</td><td></td><td></td><td>Benatallah et al. [2005]
Paolucci et al. [2002a]
Ganesan et al. [2003]
Pokraev et al. [2003]</td></tr>
<tr><td rowspan="5">Structure-Based</td><td>Non Functional-based</td><td></td><td></td><td>Zeng et al. [2004]
Ran [2003]
Maximilien and Singh [2004]</td></tr>
<tr><td rowspan="4">Functional-based</td><td rowspan="2">High-Level</td><td>Semantic</td><td>Keller et al. [2004]</td></tr>
<tr><td>Syntactic</td><td>Sycara et al. [1999; 2002]
Mecella et al. [2001]</td></tr>
<tr><td rowspan="2">Low-Level</td><td>Semantic</td><td>Sivashanmugam et al. [2003]
Aggarwal et al. [2004]
Patil et al. [2004]</td></tr>
<tr><td>Syntactic</td><td>Paolucci et al. [2002b]</td></tr>
</table>

Table 2.1: Direct Service Matching Classification

For example, Sajjanhar et al. [2004] determine the similarity between two services by constructing two vectors of words from their descriptions, then weights these words based on the Inverse Document Frequency (IDF). After that, it computes the similarity between the two weighted vectors using Singular Value Decomposition (SVD) techniques adopted from linear algebra. While, the approach of Ganesan et al. [2003] determines similarity between the two vectors based on the distance between their words, that is, computed according to the words' depth in an adopted domain taxonomy. [Benatallah et al., 2005] provide a formalization for the service discovery problem using description logic. They mapped the service matching problem into a concept covering problem, providing the rules for concept subsumption. They match services to request according to the common covered concepts.

However, such work cannot be adopted to automate the service matching process, as it does not take into consideration the achievement of users' goals. However, adopting a non structure-based approach during the matching process leads to poor precision and recall values [Bernstein and Klein, 2002; Paolucci et al., 2002b], as the semantics of services, users and application domains are ignored. Thus, human intervention becomes mandatory in order to select the correct results from the returned matching results.

The structure-based matching approach could be functional-based or nonfunctional-based. The functional-based approach is mainly concerned with functional specifications matching, while the nonfunctional-based approach is mainly concerned with nonfunctional specification matching. Currently, service matching techniques that are based on nonfunctional specification matching are mainly focused on matching services according to their quality of service (QoS) properties, such as Huang and Venkatasubramanian [2002]; Ran [2003]; Maximilien and Singh [2004]; Zeng et al. [2004]. For example, Maximilien and Singh [2004] provide an ontology for describing QoS parameters for both services and users. Users provide their constraints in terms of this ontology (user QoS policy). A service matches a user request when each of its QoS parameters satisfies the corresponding user's constraints. However, nonfunctional-based matching approaches are usually applied as secondary filters to refine the matching results returned from applying a given functional-based matching approach. Zeng et al. [2004] provide two selection approaches, a local (task-level) selection of services, and the other based on global allocation of tasks to services. The local optimization approach performs optimal service selection for each individual task in a composite service without considering QoS constraints spanning multiple tasks and without necessarily leading to optimal overall QoS. The global planning approach on the other hand considers QoS constraints and preferences assigned to a composite service as a whole rather than to individual tasks, and uses integer programming to compute optimal plans for composite service executions.

The functional-based matching approach could be based on high-level functional specifications such as goals, roles, contexts and behaviors, or it could be based on low-level functional specifications such as inputs and outputs. Both high-level functional matching and low-level functional matching can be performed by adopting either a syntactic or a semantic basis.

The semantic basis approach adopts mediation techniques to overcome semantic differences between a user's request and descriptions of services, while the syntactic does not adopt any mediation techniques. Matching in LARKS [Sycara et al., 1999; 2002] and matching proposed by Mecella et al. [2001] are examples of syntactic high-level functional-based matching approaches, while matching in WSMO [Keller et al., 2004] is an example of semantic high-level functional-based matching approaches.

In LARKS, agents are primarily matched based on their contexts. As the context of an agent is defined as a set of keywords, agents with different contexts are ignored. Later, four types of matching phases are applied: a profile matching, a similarity matching, a signature matching and a semantic matching. In profile matching, the text descriptions of the profiles are filtered using term Frequency-Inverse Document Frequency (TF-IDF). In similarity matching, the similarity of inputs and outputs of agents are computed as the average distance between elements (number of edges) within a predefined domain taxonomy, such that elements with distances greater than a given threshold are considered different. In signature matching, agents are filtered according to their inputs and outputs. Finally, in the semantic matching, the constraints over the inputs and outputs of the agents are matched via constraint implication. An agent that passes all these filters is considered the exact match of the user query. Matching in LARKS emphasizes the importance of taking an agent's context into consideration during the matching process. However, use of keywords to model the context is a very primitive approach, as the semantics of agents' contexts cannot be modelled. Also, capturing an agents' functionality as a text description in its profile leads to the use of a keyword-based matching technique (that is TF-IDF) for matching agent functionality. Hence, such approach cannot be adopted to automate the matching process.

Mecella et al. [2001] match web services based on their external behavior compatibility. A services' behavior is represented by a state chart that specifies the order in which events can be invoked. A state machine is extracted from the corresponding state chart. A service Srv_i is considered compatible with another service Srv_j when the external behavior of Srv_i can be restricted to the external behavior of Srv_j. This is done by ensuring that the state machine of Srv_j includes the same transition sequences of the state machine of Srv_i beginning from

its starting state to its final state, adopting a one-to-one state matching is adopted, where events and states are syntactically matched. For sure, this is a strict approach that leads to the appearance of false negatives as no mediation techniques are adopted.

In WSMO [Keller et al., 2004] web services are matched based on their goals and capabilities. Web services' capabilities and user goals are represented as sets of objects (such as pre-conditions, assumptions, post-conditions, and effects). WSMO uses different types of mediators to match these objects. A service is considered an exact match for a user request when every object in the service description matches an object in a user's request and vice-versa. Thus, the similarity between a service description and a user request is determined according to the number of common objects they have. Matching in WSMO emphasizes the importance of using goals and capabilities as the matching basis for web services. However, WSMO does not indicate how the proposed mediators will be designed to overcome the heterogeneities between different objects. Also it does not take services' behaviours into consideration during the matching process, which could lead to the appearance of dead and live locks during web services compositions.

Paolucci et al. [2002b] is an example of a syntactic low-level functional-based matching approach, while matching in METEOR-S [Aggarwal et al., 2004; Sivashanmugam et al., 2003; Patil et al., 2004] is an example of a semantic low-level functional-based matching approach. Paolucci et al. [2002b] match web services according to their inputs and outputs defined in their DAML-S profiles. Inputs and outputs are matched via sub-typing using the *subclassOf* property defined in the DAML-S profile, such that an element is considered a match to elements with subsuming types. A service is considered an exact match for a user request when all the inputs and outputs of a service matches corresponding inputs and outputs specified by the user. However, this is a strict approach that leads to the appearance of false negatives and positives, as it is very likely to find two similar services (that accomplish the same goal) with different signatures.

In METEOR-S, web services are primarily matched based on their annotated WSDL descriptions. These WSDL annotations are defined according to a given domain ontology. METEOR-S applies three phases for service matching. In the first phase, it matches web

services based on the operations defined in different WSDL files. In the second phase, the result set from the first phase is ranked on the basis of semantic similarity between the concepts appeared in the annotations of the selected operations and the concepts of the query respectively. The optional third phase involves ranking based on the implication between the preconditions and effects of the selected operations and the preconditions and effects of the query. The approach taken in METEOR-S is a bottom-up matching approach, that starts by matching according to the low-level functional aspects then filters the obtained results according to the high-level functional aspects. Such an approach cannot mediate between services' descriptions and user's request on a high-level manner, using the semantics of the involved application domain. Therefore, it cannot be adopted to automate the service matching process, as it does not take into consideration services' behaviors as well as achievement of users' goals.

Papazoglou et al. [2002] and Aiello et al. [2002] proposed a service request language called XSRL to formally express service requests against UDDI-resident web services. XSRL integrates between AI planning and constraint satisfaction techniques with web service technology. It has a parser, a model based planner, and a plan executor. A request written in XSRL is input to an automated planner/scheduler. After receiving the requests from a user, the planner generates a schedule for interacting with the service providers without further interaction from the user. Unfortunately, XSRL does not differentiate between the different types of services' specifications (functional, nonfunctional, and nontechnical), work in [Lazovik et al., 2004] and [Lazovik et al., 2006] extended XSRL to include business rules with relevant processes involved in a user request. However, XSRL still lacks support for service choreography, also adopting syntactic AI planners to parse and execute raises many questions about its practicality, especially it cannot handle the semantic interoperability problem.

When a centralized service discovery approach is adopted (services are located in one service registry), the direct service matching approaches discussed in this section could be used as centralized service discovery approaches. However, when multiple service registries are used, interaction protocols are needed for exchanging information among these registries. Therefore, the approaches to handle such problems mainly adopt current solutions from the

P2P technology for service execution, publishing and querying such as Wang et al. [2003] and Schmidt and Parashar [2004]. Wang et al. [2003] propose a framework that is composed of providers, brokers, and requestors to support web services in a P2P environment. Schmidt and Parashar [2004] adopt similar P2P architecture components that supports keyword queries adopting the Chord data lookup protocol [Stoica et al., 2001].

2.6 Service Composition versus Service Aggregation

A composite service is a virtual service (does not physically exist) that consists of other web services collaborating with each other in order to accomplish a given task. In this thesis, we divide the service composition process into two phases:

1. High-level Aggregation Phase: In this phase, a composite service is examined against the high-level functional aspects required by users such as goals, roles, contexts, and expected external behaviors. At this stage, a composite service is known as an *aggregate service*.

2. Low-level Execution Phase: In this phase, the execution details of the collaborating components of an aggregate service are examined in order to make sure that correct execution will take place such as ensuring transaction ACID properties, coordination, and execution monitoring [Lazovik et al., 2003; Alonso et al., 2004] .

Thus, the term *aggregate service* implies that a service exactly fulfils the user goals but execution details are not addressed, while the term *composite service* implies that a service exactly fulfils the user goals and the execution issues are resolved. In other words, a composite service is an executable aggregate service.

Service composition could be *static* or *dynamic*. Static service composition takes place during design time. The components to be used are chosen, linked, and finally complied and deployed manually. This type of composition is mostly found in industry and supported by many tools such as BPML, WSFL, WSCL, WSCI, BPEL4WS [Staab et al., 2003]. A major disadvantage of these approaches is lack of semantics representation as it mainly depends on the WSDL model for describing the messages and the data types, and also they lack the

notion of mediation. In dynamic service composition, components are identified, linked, and executed in run-time, which is main goal need to be achieved by current research efforts.

Service composition could be *conformant* or *conditional* [Berardi, 2005]. A conformant composite service matches a user request against all the possible ways component services actually behave during the coordination. A conditional composite service does not realize the user request for all behaviors of component services, however it perform some sensing activities at run-time to find suitable components. For instance, if a structure of the form "if-then-else" is adopted in a conditional composition, at run-time, the value of the "if-condition" is sensed in order to determine the path that should be followed.

2.7 Service Composite Matching

Service composition is considered one of the most active research topics of web services. In this section, we attempt to classify some of the major research efforts that deal with service composition from different perspectives. Similar to the proposed direct service matching, we classify existing composite service matching approaches based on their initial filters, as indicated in Table 2.2.

Composite service matching approaches are structure-based as well as functional-based matching approaches (nonfunctional specifications always used as secondary filters). Either low-level functional specifications or high-level functional specifications can be used as the primarily matching aspect. Also, either a semantic or a syntactic approach could be adopted during composite matching. The approaches presented by Thakkar et al. [2002]; Medjahed et al. [2003]; Thakkar et al. [2004] are examples of the semantic low-level functional composite service matching, while the approach presented by Orriëns et al. [2003; 2004] is an example of the syntactic low-level functional composite matching. However, the approaches presented by McIlraith et al. [2001]; Bultan et al. [2003]; Pistore et al. [2004]; Berardi [2005] are examples of syntactic high-level functional matching.

Medjahed et al. [2003] present a framework for composing services that are described in terms of purpose, category, quality, operations, and binding protocols. The purpose describes the offered business functionality via a text description. The category describes the applica-

<table>
<tr><td rowspan="4">Service Composite Matching</td><td rowspan="4">Structure-Based</td><td rowspan="4">Functional-based</td><td rowspan="2">High-Level</td><td>Semantic</td><td></td></tr>
<tr><td>Syntactic</td><td>McIlraith et al. [2001]
Bultan et al. [2003]
Pistore et al. [2004]
Berardi [2005]</td></tr>
<tr><td rowspan="2">Low-Level</td><td>Semantic</td><td>Thakkar et al. [2002]
Medjahed et al. [2003]
Thakkar et al. [2004]</td></tr>
<tr><td>Syntactic</td><td>Orriëns et al. [2003; 2004]</td></tr>
</table>

Table 2.2: Composite Service Matching Classification

tion domain of the web service. Synonyms and specializations of the terms used to describe the purpose and category are supported using the adopted ontology. The proposed composability model compares first the syntactic low-level features (that is the supported interaction and binding protocols), then the semantic high-level features of services (that is purpose and category). Composition soundness is checked to determine if the combined services actually provides an added value service taken into consideration the services' qualitative properties.

Orriëns et al. [2003; 2004] represent services as interfaces of web components. A specification for web components reuse and specialization is given. A composite web component is manually constituted from other web components using what is called "composition logics" that specifies the composition type and message dependency. Composition type specifies the nature of composition that could be sequential, conditional, and/or parallel (alternative service executions). Message dependency specifies if there is message dependency between component services and the resulting composite service.

McIlraith et al. [2001] proposes a tool for service composition, where services are described using OWL-S ontology, and composition is performed based on situation calculus. However the resulting composite service is not an OWL-S service. A user's request is formulated as a generic ConGolog procedure (ConGolog is an agent programming language),

where every procedure is associated with a situation tree that denotes a partial specification of the required behavior. Services are represented as actions (tree nodes) in the situation tree. The situation tree is pruned according to user's constraints and preferences, after executing nodes' services until a linear sequence of services is reached.

Thakkar et al. [2002; 2004] represent services as data sources that are described in terms of inputs, outputs and constraints. A mediator based approach is adopted to relate between different sources using a given domain ontology. The mediator transforms the user's request (that is given in terms of inputs, outputs, and constraints) into a set of source models. Then uses a forward chaining algorithm to build an integration plan consisting of existing sources. Later, the mediator optimizes the integration plan using a data flow analysis algorithm, and filters the obtained data according to the required constraints.

Bultan et al. [2003] propose a framework for modelling the global behavior of service composition based on services' conversations. A service's conversation is the sequence of exchanged messages from the user's perspective, and is represented by a mealy machine. the user's request is represented by all the possible desired conversation patterns (that is the desired message sequences), and the answer to the request is the specification of services' mealy machines that matches the required conversations. Services are composed using projection-join between the mealy machines of component services, where messages, inputs and outputs are matched syntactically.

Pistore et al. [2004] address service composition using planning under uncertainty and model checking approaches, where services are represented by their behaviors using non deterministic finite state machines, and users' requests are expressed using the goal language EaGLe. The returned answer by the planner is a plan that indicates how to coordinate services in order to fulfil a user's goal. The planner also generates monitors in order to make sure the component services are behaving the way they are defined. A planning domain is defined in terms of states, actions, and observations. Syntactical model checking approaches are used to limit space search explosion due to the presence of non determinism.

Berardi [2005] presents a framework for service composition, where services are modelled as deterministic finite state machines, and a user's request is represented by a tree of actions.

Execution trees are generated from services' finite state machines to determine all the possible execution behaviors of services. A tree labelling approach is adopted to associate the the actions of the user's tree to the available services such that concurrent execution of services could be determined.

Most of the work in service composition is based on the internal behavior of services; whether it is presented in a form of execution trees, situation trees, finite state machines or even nondeterministic machines. We argue adopting internal behavior during the matching process is not a practical approach, as this violates services encapsulation (more details are given in Chapter 3). Furthermore, existing service composition approaches assume that all services' descriptions and users' requests adopts the same vocabulary, and they require the behavior models to have the same number of states in order to be matched. Such aggregate approaches are very rigid as they do not adopt any mediation techniques between the services' descriptions and users' requests, which indeed is not a practical assumption that easily leads to the appearance of false negatives and false positives. To overcome these limitations, in Chapter 5 we propose a correctness-aware aggregate service matching technique that is based on services' G^+ models as well as the functional substitution semantics of application domains. We classify our approach as a semantic high-level functional aggregate approach.

Chapter 3

SWSMF:A Semantic Web Services Matching Framework

A major obstacle for achieving a fully automated service matching process is the need for human intervention to determine the correctness of matching results. In order to overcome this problem, the matchmaker must be able to determine the correctness of the matching results with respect to the defined users' goals. Hence, the semantics of web services, users, and application domains need to be captured in a machine-understandable format, so that they can be understood and used by the matchmaker. Therefore, this chapter proposes a Semantic Web Services Matching Framework (SWSMF), that indicates the types of semantics to be captured from web services, users and application domains in order to determine the correctness of the matching results, also it indicates how these types of semantics will be represented in a machine-understandable format.

3.1 Preliminaries

A matching framework provides the concepts and the basis that constitutes the matching process. Therefore, the automation of the matching process is basically dependant on what is supported by the adopted matching framework. In order to automate the service matching process, we have identified the following characteristics that need to be supported by the

adopted matching framework:

1- Separates web services' semantics from their presentations: Semantics of web services are obtained from their descriptions. Having these descriptions presented in a way that suits only human users will not enable the matchmaker to understand services' semantics from such presentation. Therefore, the matching framework must separate between services' presentation (created for human advertisement purposes) and services' descriptions (created for automated matching purposes). The description of web services must be in a machine-understandable format.

2- Handles semantic interoperability: Semantic interoperability emerges due to the use of different concepts and semantics for modelling services' descriptions and users' requests, which leads to the appearance of false negatives and false positives during the matching process. Therefore, the matching framework must support solutions for this problem. For example, if ontologies will be used to overcome the semantic interoperability problem, the matching framework must indicate how ontologies will be used to describe both web services' descriptions and users' requests.

3- Adopts goal-based matching: One of the main characteristics of web services is that they are *goal-based*, as services are mainly invoked to achieve particular goals (such as flight reservation and freight movement). Achievement of users' goals is the indication that the invoked services are correct. To enable the matchmaker to automatically determine the correctness of the matching results, it needs to determine whether the services returned as matching results can achieve the required users' goals or not. This requires the matching framework to explicitly capture the goals of web services and users and their corresponding achievement process in a machine-understandable format, such that an automated goals-based matching process could be performed.

4- Adopts role-based matching: One of the key characteristics of web services is that they are role-based, because a web service is designed to serve specific category of users (a specific market sector) who are searching for services to help them fulfilling their roles. For example, services that provide medical analysis will not be invoked

by hotel managers. Furthermore, roles are important in providing services' access control, users with different roles from the ones supported by services should not access such services. Therefore, the matching framework must explicitly capture the roles supported by services and the roles played by the users in a machine-understandable format.

5- Adopts web services' external behavior: A web service can be seen as an object or as a software component, hence it has two types of behaviors: external and internal. The external behavior represents the sequence of interactions and data flows between a web service and a user in order to achieve the user's required goal, while the internal behavior represents the state transitions a service goes through in order to achieve a given goal. We argue that the matching framework should adopt the external behavior as a part of its goal-model (to describe goal achievement) instead of the internal behavior for the following reasons:(1) To maintain services' encapsulation, as the internal behavior may depend on internal algorithms that a service provider does not want to reveal. (2) A given user is mostly interested in only one achievement pattern (a scenario for the expected external behavior) for achieving the required goal, it does not matter to the user how a service will achieve the goal internally but what matters to the user is the set of interactions required by a service in order to accomplish the required goal. (3) The internal behavior is always complex to describe, and it costs a lot to build an accurate internal behavior model of a service in terms of expertise and time. On the other hand, external behavior can be partially captured via informal scenarios, which are more easily anticipated and understood.

6- Adopts web services different contexts: Adopting the concept of *context* is an important factor in the matching process as it provides the matchmaker with more information to enhance the quality of the returned answers [Fidge, 2002; Hattori et al., 2003; Lee and Helal, 2003]. A context in web services is generally identified as information about the execution environment that can be used to enhance service performance [Dey et al., 1999; Pratistha and Zaslavsky, 2004]. However, we argue that this is a limited

perspective for defining web services' contexts, because each web services' aspect should have its own identifying context. For example, a web service could have a context for pricing, a context for security, a context for execution details, and a context for its goal achievement (requirements and effects). Hence, the matching framework should capture such contexts in a machine understandable format such that they can be adopted in the automated matching process.

7- Facilitates web service aggregation: As a user's request could be complex such that it cannot be fulfilled by one service, a group of services could be aggregated together to fulfil such request. Therefore, the matching framework must provide a basis for service aggregation in order to make sure the aggregated service has a real value.

8- Supports non-functional and non-technical requirements: Customer satisfaction has motivated business to put more emphasis on service quality issues, as these issues have been previously neglected as part of the matching process. Therefore, the matching framework must capture such requirements in a machine-understandable format, as they form the basis of the service selection process. Like functional requirements, non-functional and non-technical requirements require collection, specification, analysis, allocation, and testing.

9- Adopts users' semantics: As we indicated before, achievement of users' goals when the returned matching services are invoked is the indication for the results' correctness. Therefore, the matching framework must take into consideration user's semantics in order to provide a basis for determining results' correctness. Users' requests should include the same types of specifications used for services' descriptions. Users' requests should be captured in a flexible manner. For example, a user can specify different categories of constraints that reflect their importance. A user can define a category of mandatory and essential constraints that must be satisfied, and define another category of optional constraints that is preferred to be satisfied, so that the matchmaker can use such semantics during the matching process[1].

[1]This approach of request modelling is known as imprecise computation modelling [Elhaweet et al., 2001].

According to these characteristics, we identify the following types of specifications to be captured in services' descriptions and users' requests (a classification for these types is depicted in Figure 3.1):

High-level functional specifications: They are concerned with describing services' functionality from a high-level perspective. For example, they should include information about services' goals, roles, external behaviors and goals' achievement contexts.

Low-level functional specifications: They are concerned with describing services' functionality from a low-level perspective. For example, they should include information about services' access interfaces, required bandwidth and messaging protocols.

Non-functional specifications: They are concerned with describing services' non functional aspects. For example, they should include information about quality of service and security.

Non-technical specifications: They are concerned with describing services' marketing aspects. For example, they should include information about such as pricing, discounts, and service providers.

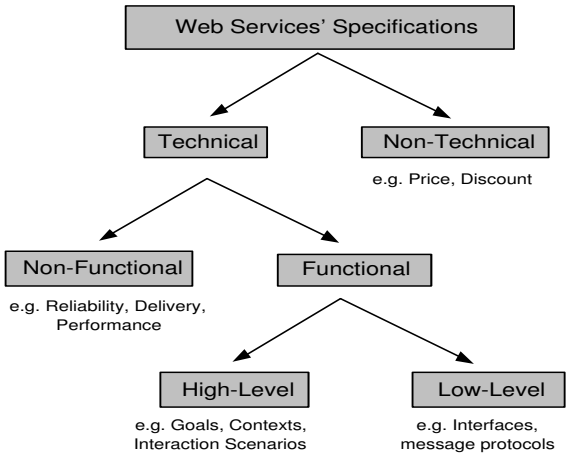

Figure 3.1: Web Services Specifications Classification

According to this classification, the matching process should be performed in a multi-stage manner such that every stage will be considered with matching only one type of specifications.

As we indicated before (see Section 1.2), the matching process should start by matching the high-level functional specifications, as this provides the matchmaker with the information required to determine the correctness of the matching results. Later, matching using the other types of specifications should be performed to refine the obtained matching results.

The remaining of this chapter is organized as follow. Section 3.2 indicates the limitations of existing frameworks in supporting all of these characteristics, motivating the need for the SWSMF framework. Section 3.3 provides an overview of the adopted approach for building the SWSMF framework, adopting the required characteristics. Also it provides the conceptual model for the SWSMF framework. Section 3.4 introduces the proposed meta-ontology approach for modelling application domains' ontologies, which acts as the basis for handling the semantic interoperability problem. Section 3.5 describes the proposed G^+ goal model, which acts as the basis for representing the goal-based high-level functional aspects (goals, goals' achievement contexts, and expected external behavior) of both web services and users. Section 3.6 indicates how different goal achievement patterns will be extracted from the proposed G^+ model. Finally, Section 3.7 concludes the chapter.

3.2 Limitations of Existing Matching Frameworks

This section illustrates the limitations of some of the important existing matching frameworks[2] in supporting the required characteristics; motivating the need for a new matching framework. We define a scale of four values to evaluate the support for a given characteristic. These values are:

- F: fully supported, which means the characteristic is explicitly addressed and solutions are proposed.
- P: partially supported, which means the characteristic is explicitly discussed, but no solutions are proposed.
- S: similarly supported, which means the characteristic is not explicitly discussed, but similar concepts are discussed.

[2]These matching frameworks were introduced in Chapter 2.

- N: not supported.

A matching framework that enables the matchmaker to automatically determine the correctness of the matching results need to have the value *F* against all the required characteristics. As indicated in Table 3.1, none of the existing frameworks fully supports all the required characteristics. The following discussion analyzes the values given for each framework listed in this table.

Characteristic	UDDI	EbXML	E-speak	LARKS	OWL-S	WSMO
1- **Separates Semantics From Presentation**	N	N	F	F	F	F
2- **Handles Semantic Interoperability**	N	N	F	F	P	F
3- **Adopts Goal-Based Matching**	N	N	N	N	N	F
4- **Adopts Role-Based Matching**	N	N	N	N	N	F
5- **Adopts External Behaviors**	N	N	N	N	N	N
6- **Adopts Different Contexts**	N	S	N	S	N	S
7- **Facilitates Service Aggregation**	N	N	N	N	N	F
8- **Supports non-functional/non-technical requirements**	S	S	N	N	P	F
9- **Adopts User Semantics**	N	N	N	N	N	F

Table 3.1: Comparison between Existing Matching Frameworks

As UDDI is used to describe web services using only text descriptions, it has no support for any of the mentioned characteristics. However, some parts of these text description could include information about non-functional specification, hence UDDI is considered similarly supporting it.

EbXML uses two views to describe business interactions: a Business Operational View (BOV), and a Functional Service View (FSV). BOV deals with business process semantics such as agreements, obligations, and data transactions (mostly captured via UML diagrams). FSV deals with low-level functional specifications such as interfaces and protocols. Both views are designed for human use only, hence no machine-understandable information is supported. Use of BOV and FSV is to separate the high-level from the low-level functional specifications. Also BOV could contain information about non-functional and non-technical specifications. Therefore, it is considered similarly supporting multiple contexts, non-functional and non-technical specifications.

E-speak handles the semantic interoperability problem using a controlled vocabulary approach. It does not take user semantics into consideration. However, it describes the semantics of services as a set of attributes using the controlled vocabulary. Therefore, it is

considered supporting semantics-presentation separation. However, E-speak does not support explicit goal models, explicit context models, or external behavior models.

LARKS does not take users' semantics into consideration, and handles the semantic interoperability problem by using a predefined vocabulary. LARKS defines a slot structure to describe an agent capability. Therefore, it is considered supporting semantics-presentation separation as parts of the slot structure are described in a machine-understandable format, such as inputs and outputs. Other parts are described as free text, such as agent's functionality. LARKS defines only one context for an agent as a set of predefined keywords. Hence, if more than one context is needed to be defined for the same agent, the keywords representing these contexts will be added to the original keyword set. Therefore, LARKS is considered similarly supporting multiple contexts. However, LARKS neither supports explicit goal models nor non-functional requirements. Also, capability aggregation of agents is not addressed in LARKS.

OWL-S does not take user semantics into consideration. It supports the use of domain ontologies by referring to the adopted ontology in the *service category* parameter located in the service profile. However, there is no explicit indication of how such an ontology will be used to define the adopted vocabulary. Therefore, OWL-S is considered partially supporting semantic interoperability handling. There are three different parts of OWL-S (profile, process model, and grounding model), OWL-S is considered supporting semantics-presentation separation. OWL-S has no means for explicitly capturing web services goals and their corresponding external behavior. Also, it does not provide any basis for service aggregation. The non-functional properties in the service profile (such as the service name, human-readable description and contact information) are not explicitly based on standard meta-data specifications, hence it is considered partially supporting non-functional requirements.

WSMO uses mediators to overcome the semantic interoperability problem. WSMO supports the use of explicit goal models, but it does not support any facility for behavior modelling (neither internal nor external). By the means of constraints; multiple contexts could be captured, but the concept of context is not explicitly mentioned in WSMO.

3.3 SWSMF Approach

This section demonstrates the basic ideas used to establish the SWSMF framework in order to fully support the required characteristics.

1- Separating Web Services' Semantics from their Presentation

The SWSMF approach to separate the semantics of web services from their presentation is to have two types of views for web services: a Service Active View (SAV) and a Service Passive View (SPV). The active view contains only machine-understandable information that will be used in the automated matching process (SAV structure will be given later in this section). The passive view contains information that is not in a machine-understandable format such as UML diagrams, business documents, and deployment models, hence it will be used by human users to select services from the answers returned by the automated matching process. An SPV is considered passive because it is not adopted in the automated matching process, while an SAV is considered active because it is the basis for the automated matching process. SWSMF allows a web service to have multiple different SAVs, for the following reasons:

1. An SAV represents a *contract* that a service provider could provide, hence different SAVs allows a service provider to advertise different contracts for the same service.

2. Service providers might need to publish the same service using different ontologies to cover more market sectors, so they need to have a separate SAV for each ontology.

3. A service could have different goals in different application domains. For example, a vehicle renting service could advertise its service in both tourism and logistics application domains, where every domain has its corresponding goals. In the tourism domain, a goal could be *car-renting* while in the logistics domain, a goal could be *truck-renting*. Thus, each domain will have a corresponding SAV.

However, a web service needs to have at least one SAV to enable the matchmaker to automatically determine the correctness of the matching results, otherwise this cannot be achieved, as keyword-based techniques will be the only option to match SPVs.

2- Handling the Semantic Interoperability Problem

SWSMF adopts the use of ontologies to overcome the semantic interoperability problem. However, as both the consensus approach, and the multi-ontology approach have their own pros and cons (previously discussed in Section 2.2), SWSMF compromises between the two approaches by proposing a meta-ontology approach, as illustrated in Figure 3.2.

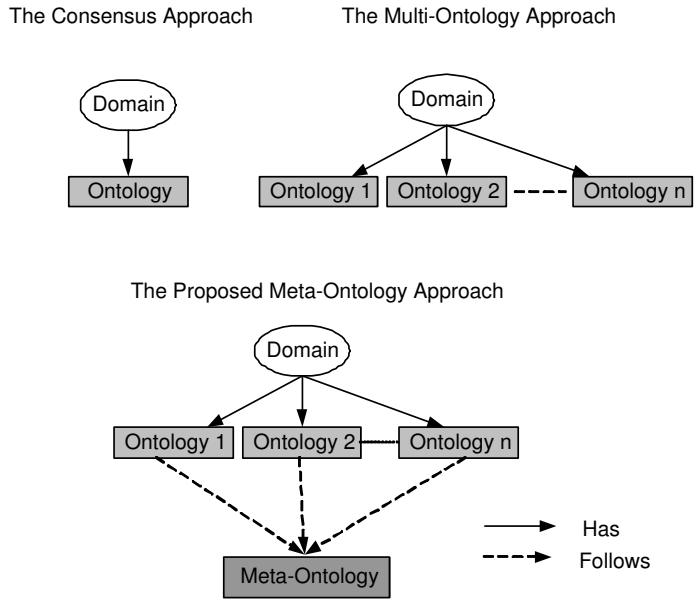

Figure 3.2: Approaches for Application Domain Representation

In the proposed meta-ontology approach, a given application domain is allowed to be represented by several ontologies (as in the multi-ontology approach) provided that these ontologies adopt the proposed meta-ontology (as in the consensus approach). The proposed meta-ontology (see Section 3.4) captures the concepts and operations of application domains. Also it captures the concepts functional substitution semantics using the proposed concepts substitutability graph. By adopting a meta-ontology, the flexibility of the multi-ontology approach is maintained as many ontologies can describe the same application domain, and the ontology mapping process becomes a straightforward process, as all ontologies will have the same structure and will be capturing the same types of semantics.

3- Adopting Goal-Based Matching

SWSMF explicitly captures web services' goals in their SAVs using the proposed G^+ goal model. We have overcome the limitations of coupling between goals and scenarios by linking between goals, scenarios and their corresponding goal achievement contexts. Hence, a G^+ model consists of a goal, its realization scenarios and the corresponding high-level functional contexts (goal achievement contexts), as indicated in Figure 3.3 (more details about G^+ are given in Section 3.5).

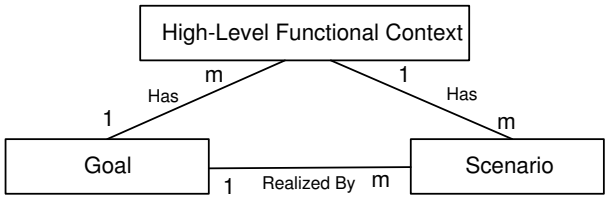

Figure 3.3: G^+ Conceptual Model

SWSMF allows a web service to have different goals. Of course, every defined goal has its own realization scenarios and achievement contexts.

4- Adopting Role-Based Matching

SWSMF explicitly captures services' supported roles in their SAVs, and captures users' roles in their UAVs. The proposed meta-ontology captures roles as concepts representing application domain actors (see Definition 6, Page 55). SWSMF does not require the roles of services and users to be defined using the same concepts, however they can use different concepts for describing the required roles, as SWSMF uses the semantics captured in the meta-ontology to determine functionally substitutable roles, more details are given in Section 3.4. During the high-level functional specification matching, SWSMF filters services according to their roles. Later, it filters the results according to their goals, contexts, and behaviors, as this minimizes the search space.

5- Adopting Web Services External Behaviors

SWSMF uses the concept of scenarios to capture the external behaviors of web services, in which a scenario indicates the expected interactions between users and a given web service in order to achieve a given service goal. However, there could exist different scenarios that can be used to achieve the same goal, and also these scenarios could be described in different abstraction levels. Therefore, SWSMF proposes the concept of a *scenarios-network* to define the different scenarios needed to achieve the required goals. A scenarios-network is part of the service G^+ model and is described in Section 3.5. Formal external behaviors models are automatically extracted from the defined scenarios-network during the matching process (more details are given in the next chapter).

6- Adopting Web Services Different contexts

Based on the classification given in Figure 3.1, SWSMF defines four different types of contexts for a web service to be captured in its SAV(s): a High-Level Functional Context (HLFC), a Low-Level Functional Context (LLFC), a Non-Functional Context (NFC), and a Non-Technical Context (NTC). An HLFC captures the requirements and effects of services' goals, as described in Section 3.5. An LLFC is restricted to capture the required bandwidth and the supported device types. An NFC is restricted to capture the following QoS parameters:

- Reliability: the probability that a web service has successfully responded to a request within the maximum expected time frame.
- Availability: the percentage of a web service's up-time.
- Reputation: the average ranking provided by service consumers to the web service.

An NTC is restricted to capture the service price. Indeed, LLFC, NFC, and NTC could capture more information than specified by SWSMF. However, we leave this issue for future work as this thesis is only concerned with the high-level functional specifications.

7- Facilitating Web Services Aggregation

SWSMF uses the G^+ models of web services as a basis for service aggregation. This is because one of the main characteristics of goals is that they could be aggregated forming more complex goals, which gives the opportunity to match the required users' goals.

8- Supporting non-functional and non-technical specifications

SWSMF differentiates between the non-functional and non-technical specifications. It captures the non-functional specifications via the NFC and the non-technical specifications via the NTC. Both the NFC and the NTC are parts of services' SAVs.

9- Adopting User Semantics

SWSMF captures users' semantics via the User Active View (UAV). A UAV has the same structure as SAV. SWSMF allows a user to have different active views with different roles and goals. The user creates the required UAVs using an adopted ontology that is compliant with the proposed meta-ontology.

SWSMF model acts as an ontology for modelling semantic web services and users' requests, specifying what should be captured and how. Figure 3.4 depicts SWSMF conceptual service model, while Figure 3.5 depicts SWSMF conceptual user model. SWSMF uses the same structure of an SAV to describe a UAV. The automated matching process will be between services' SAVs and users' UAVs. An SAV of a web service captures the following specifications in a machine-understandable format:

- Ontology: The URI of the adopted application domain ontology,
- G^+: The defined goal-based high-level functional specifications,
- Role: A supported actor of the involved application domain,
- Non-functional Context: The defined non-functional specifications,
- Non-technical Context: The defined non-technical specifications, and
- Low-level functional context: The defined low-level functional specifications.

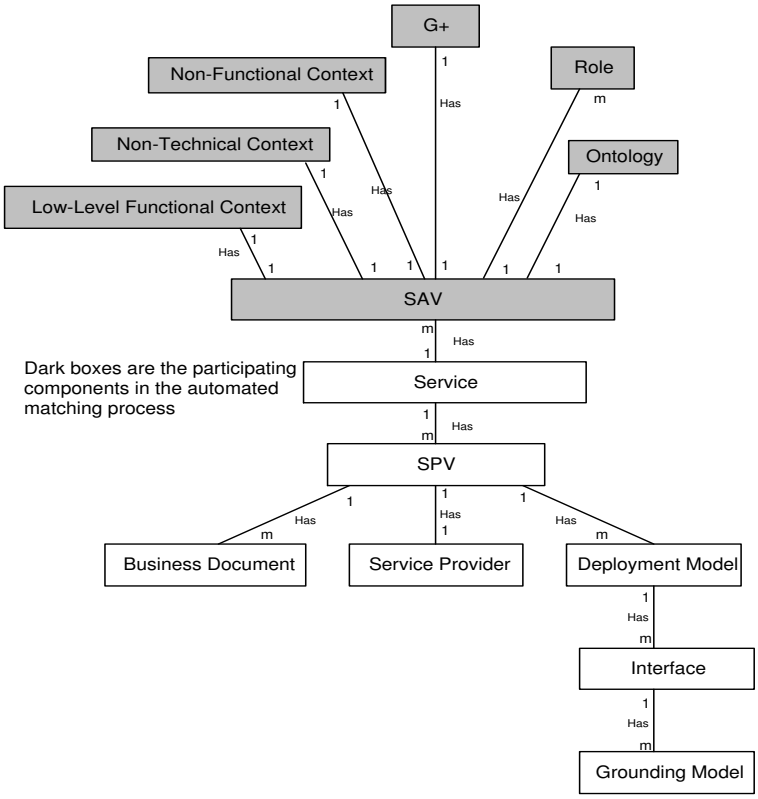

Figure 3.4: SWSMF Conceptual Service Model

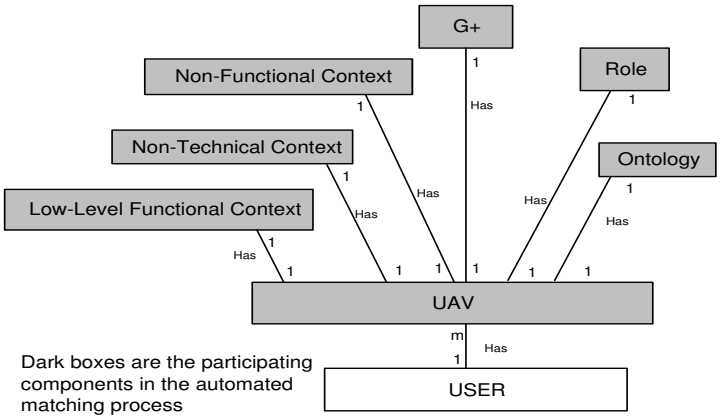

Figure 3.5: SWSMF Conceptual User Model

SWSMF does not require any specific description language to provide the flexibility for adopting our matching techniques. That any description language that can support the definition of the meta-ontology elements in the specified attribute-value format will be suitable to describe the SWSMF framework. There are four perspectives that should be taken into consideration for SWSMF realization, as each perspective plays a different role during the matching process. These perspectives are: an ontology engineer perspective, a service provider perspective, a user perspective, and a matchmaker perspective. The ontology engineer is the person responsible for building application domain ontologies. Service providers are the ones responsible for creating and publishing a given service. Users (humans and systems) need to format their request according to SWSMF specifications, while the matchmaker examines the submitted UAVs against the published SAVs, more details about SWSMF realization are given in Appendix A.

3.4 A Meta-Ontology for Modelling Application Domains

As this thesis is concerned only with the high-level functional specifications, the proposed meta-ontology specifies the elements and semantics to be captured in application domains' ontologies for modelling and matching the high-level functional specifications. The proposed meta-ontology consists of two layers: the schematic layer, and the semantic layer, as indicated in Figure 3.6. The schematic layer is responsible for defining the types of application domains' elements that need to be captured. The semantic layer captures the relations and their corresponding semantics between the captured elements' types.

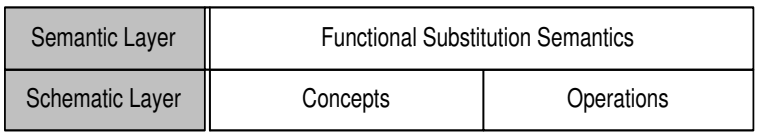

Figure 3.6: Meta-Ontology Layers

3.4.1 The Schematic Layer

Concepts and operations are the application domain elements that need to be captured in the schematic layer, as they are the basic elements needed to describe the high-level functional specifications of web services. Basically, concepts and operations are defined via a set of attributes. For each attribute of an application domain element: a name, a text description for the attribute, a type (such as string, numeric) and a range must be defined in the adopted ontology. For example, in the `Logistics` application domain, we can find a `Cargo` concept that has a `Weight` attribute that has a description such as "a numeric value that represents the number of tons of the cargo", a type such as `numeric`, and a range such as $\langle 0, 10000 \rangle$, where 0 is the lower limit and 10000 is the upper limit.

We capture the information about application domain elements using constraints. Without loss of generality, we adopt basic constraint models that are rich enough to explain the proposed *constraint substitutability* concept (more details are in the next chapter). However, more complex (semantically richer) constraint models could be adopted. In this thesis, a constraint could be a *Comparative Constraint* or a *Bounding Constraint*. A bounding constraint could be either a *Disjunctive Bounding Constraint* or a *Conjunctive Bounding Constraint*.

Definition 1 (Comparative Constraint) *A comparative constraint over an attribute $attr_k$ of an element E_i is defined as ($E_i.attr_k$ λ τ), where $\lambda \in \{=, \neq, <, >, \leq, \geq\}$, τ is a single value that has the same type as $attr_k$.*

For example, (`Cargo.Weight` $=$ 100), (`Cargo.Weight` $\geq$ 100), (`Cargo.Weight` $\neq$ 100) are examples for comparative constraints over the `Weight` attribute of the `Cargo` concept.

A disjunctive bounding constraint is used to limit the value of a given attribute to specific single value from a group of finite values.

Definition 2 (Disjunctive Bounding Constraint) *A disjunctive bounding constraint over an attribute $attr_k$ of an element E_i is defined as ($E_i.attr_k \in \eta$), where $\in$ is the disjunctive bounding constraint operator, and η is a finite set of single values that have the same type as $attr_k$.*

For example, (`Cargo.Type` $\in$ {`Motor-Vehicle,Dangerous-Cargo,Food-Product`}) is an example of a bounding constraint over the `Type` attribute of the `Cargo` concept. It indicates that `Cargo.Type` could be either a `Motor-Vehicle`, a `Dangerous-Cargo`, or a `Food-Product`.

A conjunctive bounding constraints is used to assign multiple values to a given attribute simultaneously.

Definition 3 (Conjunctive Bounding Constraint) *A conjunctive bounding constraint over an attribute $attr_k$ of an element E_i is defined as ($E_i.attr_k \subseteq \eta$), where $\subseteq$ is conjunctive bounding constraint operator, and η is a finite set of single values that have the same type as $attr_k$.*

For example, if a freight forwarder service could support multiple activities during its operation such as `Custom-Clearance` and `Packaging`. This should be modelled as (`Activity.Type` $\subseteq$ {`Custom-Clearance`, `Packaging`}).

Every constraint[3] has a corresponding scope that indicates which attribute and which element are restricted by the constraint.

Definition 4 (Constraint Scope) *Given a constraint $Cnst$ over an attribute $attr_k$ of an element E_i. The scope of $Cnst$ (denoted as $\Xi(Cnst)$) is $E_i.attr_k$.*

For example, the scope of the constraint `(Cargo.Weight` $\in$ {`100,200,300`}`)` and the scope of the constraint `(Cargo.Weight` $=$ 100`)` have the same value `Cargo.Weight`. Similarly, the scope of a set of constraints is formed from the scopes of the corresponding set elements.

Definition 5 (Constraints-Set Scope) *Given a set of constraints $f_i = \{ Cnst_1, Cnst_2,..., Cnst_n\}$, the scope of f_i (denoted as $\Xi(f_i)$) is $\{ \Xi(Cnst_1), \Xi(Cnst_2),..., \Xi(Cnst_n) \}$.*

For example, the scope of the set {`Cargo.Weight` $=$ 100, `Cargo.Type` $=$ `Cars`} is the set {`Cargo.Weight`, `Cargo.Type`}.

[3]In what follows, we will use the term *constraint* to refer to both comparative and bounding constraints unless explicitly specified.

Definition 6 (Concept) *A concept is an application domain element representing an entity or an actor that can be described via a set of features, which are defined in attribute-value format.*

For example, in the `Logistics` application domain, we can find entities such as `Freight`, `Cargo`, `Port`, `Origin`, `Destination`, `ShippingOrder`, and `Payment`. The `Cargo` concept could have a set of features such as {`Weight` = 100, `Volume` = 3000, `ExpiryDate` = Jan-2004, `Type` = Cars }, where `Weight`, `Volume`, `ExpiryDate`, and `Type` are attributes.

An application domain role is a concept representing an application domain actor. For example, we can find actors such as `Client`, `Freight Forwarder`, and `Transport Operator`. A `Client` actor could have a set of features such as {`Name` = `Islam Elgedawy`, `ReferenceNo` = `s3060314`, `Address` = `Melbourne-Australia`}, where `Name`, `ReferenceNo`, and `Address` are attributes.

Definition 7 (Operation) *An operation is an application domain element representing a transaction that can be described via a set of input concepts, a set of output concepts, a set of pre-constraints over the input concepts, a set of post-constraints over the output concepts, and also has a set of features that are defined in attribute-value format.*

For example, in the `Logistics` application domain, we can find operations such as `SendShippingOrder`, `AnalyzeShippingOrder`, and `SettlePayment`. A `SendShippingOrder` operation could have {`Freight`, `Origin`, `Destination`} as its set of inputs. Also it could have {`ShippingOrder`} as its set of outputs. Its pre-constraints could be described as the following set of constraints {`Freight.Details` $\neq$ `Null`, `Origin.Details` $\neq$ `Null`, `Destination.Details` $\neq$ `Null`}. Its post-constraints could be described as the following set of constraints { `ShippingOrder.Status` = `Created`}. The `SendShippingOrder` operation could have attributes in its features such as `Name` and `ID`. We use the notation Op_x^{Pre} to refer to the pre-constraints of the operation Op_x, and we use the notation Op_x^{Post} to refer to the post-constraints of the operation Op_x.

3.4.2 The Semantic Layer

The semantic layer captures the semantics of application domain to be used for mediation processes needed in the matching process. Such semantics are captured via different types of relations specified according to the adopted matching schemes. The proposed semantic layer (according to the proposed matching scheme described in the next chapter) captures the functional substitution semantics between application domain concepts.

When capturing the functional substitution semantics between concepts, the functional substitution semantic between concepts' attributes must be taken into consideration as well, as attributes with the same name could have different semantics. For example, in the `Logistics` application domain, the concept `Cargo` can be substituted by the concept `Freight` in a `SendShippingOrder` operation. However, their attributes do not necessarily have the same semantics, for example `Cargo.Type` and `Freight.Type` have the same semantic interpretation that indicates the nature of the items to be shipped, while `Cargo.Cost` and `Freight.Cost` may have different interpretation (that is for example, the cost of `Cargo` does not include taxes, while the cost of the `Freight` does). Therefore, we argue that the functional substitution semantics should be captured on the level of scopes (that is Concept.Attribute) indicating the mappings between the values of substitutable scopes. As the substitution semantics could vary according to the logic of the involved application domain operation, the substitution semantics should be defined with respect to every operation defined in the involved application domain. However, such substitution semantics must be captured in a context-based manner; avoiding use of rigid mapping rules (that is applied in all contexts); indicating the mapping between scopes' values. To fulfill these requirements, we propose the *concepts substitutability graph* to capture scopes' substitution semantics in a context-based manner.

Concepts Substitutability Graph

The concepts substitutability graph is constituted of segments, where each segment correspond to an application domain operation. A substitutability graph segment is a directed graph, in which a node is a scope, and a graph edge shows the substitutability direction, it

starts from the source node (the replacing/substituting scope) and the arrow head points to the target node (the scope to be replaced/substituted). Existence of an edge from a source scope into a target scopes means that the source scope is functionally equivalent to the target scope according to the semantics of the segment's operation. However, there are conditions for this substitution that must be satisfied first by the user's goal achievement context in order to apply such substitution. Hence, every edge in the graph has a set of substitution constraints, a corresponding conversion function, and an operator mapping matrix.

Definition 8 (Substitution Constraint) *A substitution constraint is a constraint that must be satisfied at the substitution time in order to be able to replace/substitute a target scope by another source scope.*

Definition 9 (Conversion Function) *A conversion function is a mapping function that maps between the values of one scope (source scope) into the values of another scope (target scope). It takes one input value and returns a set of finite output values. The type of the input value is the same as the type of the attribute appeared in the source scope, while the type of an output value is the same as the type of the attribute appeared in the target scope.*

Table 3.2 indicates different forms of conversion functions. As facts are captured via constraints, identifying the mappings between scopes' values is not enough, as we need to know the effect of such mappings on constraints' operators when we determine constraints substitutability (details in the next chapter). Therefore, we introduce, the use of *Operator Mapping Matrix* (OMM) to show the mappings between constraints' operators according to the logic of the corresponding conversion function (that is each conversion function has its own OMM). By doing so, we guarantee scopes' substitutability does not change the represented facts. A row of an operator mapping matrix represents the mappings from a given source operator (supported by the conversion function) into the other corresponding operators. As we have eight types of supported operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $\in$, $\subseteq$), an operator mapping matrix should have eight rows and eight columns. Elements of OMM are either `1` (when there is a mapping from the source operator to the operator in the corresponding column) or `0` (when there is no mapping). For example, the operator mapping

matrix depicted in Figure 3.7 is the one corresponding to the conversion function of the edge from `Credit.Period` into `Payment.Type` indicated in Table 3.2. The matrix indicates the supported operators to be mapped according to the defined conversion function are $(=, >)$ and they should be mapped into the $(=)$ operator when the scope `Credit.Period` substitutes the scope `Payment.Type`. For instance, the constraint (`Credit.Period` $= 15$) will be mapped into the constraint (`Payment.Type = Credit`) according to the defined conversion function and OMM, more details about use of conversion functions and their corresponding operator mapping matrices will be given in the next chapter.

	$=$	$\neq$	$<$	$>$	$\leq$	$\geq$	$\in$	$\subseteq$
$=$	1	0	0	0	0	0	0	0
$\neq$	0	0	0	0	0	0	0	0
$<$	0	0	0	0	0	0	0	0
$>$	1	0	0	0	0	0	0	0
$\leq$	0	0	0	0	0	0	0	0
$\geq$	0	0	0	0	0	0	0	0
$\in$	0	0	0	0	0	0	0	0
$\subseteq$	0	0	0	0	0	0	0	0

Figure 3.7: OMM for the edge from `Credit.Period` into `Payment.Type`

By adopting the concepts substitutability graph, the matchmaker can find the right substitution semantics according to the required goal, as every goal is represented by an operation (more details about the goal model are given in Section 3.5). We define a concepts substitutability graph as follows.

Definition 10 (Concepts Substitutability Graph) *A concepts substitutability graph CSG is a multi-segment graph such that $CSG = \bigcup_{i=1}^{n}\{\langle Op_i, V_C, E_C\rangle\}$, where $\bigcup$ is the set union operator, $\langle Op_i, V_C, E_C\rangle$ is the substitutability graph segment of operation Op_i (denoted as Seg_i), n is the number of operations in the domain, V_C is a set of graph nodes such that $V_C \subseteq \Delta_C \bigotimes \Delta_A$, where Δ_C is the set of all concepts, and Δ_A is the set of all attributes, and $\bigotimes$ is the cross product operator. E_C represents a set of graph edges such that $\forall\ E_{(a,b)} \in E_C$, $E_{(a,b)} = \langle V_a, V_b, \Pi_{(a,b)}, \Psi_{(a,b)}, \Omega_{(a,b)}\rangle$, where $\Pi_{(a,b)}$ is the set of substitution constraints that must be satisfied such that the scope represented by the node V_a (source node) can replace (functionally substitute) the scope represented by the node V_b (target node), $\Psi_{(a,b)}$ is the corresponding conversion function, and $\Omega_{(a,b)}$ is the corresponding operator mapping matrix.*

Figure 3.8 depicts a representation for the concepts substitutability graph. The figure shows that for every pair of concepts; the substitutable attributes are defined as well as their substitution constraints, conversion functions, and operator mapping matrices with respect to every application domain operation. Table 3.2 indicates an example of a segment of a concepts substitutability graph in the logistics application domain that corresponds to the *CargoTransportation* operation (more details about meanings of the listed attributes will be given in the next section), to simplify the representation of the segment, we did not include the operator mapping matrices in the table. A row in Table 3.2 represents an edge in a segment in the substitutability graph. For example, the first row indicates there is an edge from the scope `Cargo.Details` into the scope `Freight.Details`, and also indicates the corresponding conversion function. A set of substitution constraints could be empty.

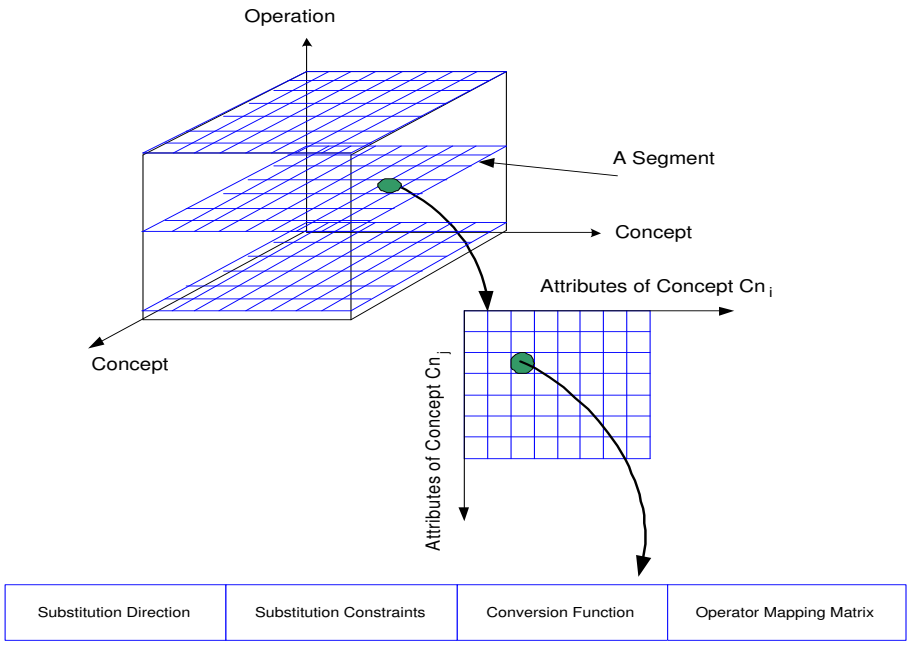

Figure 3.8: A CSG Representation

The quality of the data captured in this graph is the responsibility of the ontology engineer (more details about the responsibilities of an ontology engineer are given in Appendix A). The ontology mapping process is performed by creating a common concepts substitutability graph from the concepts and operations of both ontologies, hence the matchmaker does not

Source (V_a)	**Target** (V_b)	**Conversion Function** ($\Psi_{(a,b)}$)	**Substitution Constraints** ($\Pi_{(a,b)}$)
Cargo.Details	Freight.Details	Freight.Details = Cargo.Details	
Freight.Details	Cargo.Details	Cargo.Details = Freight.Details	
Cargo.POL	Origin.Details	Origin.Details = Cargo.POL	
Origin.Details	Cargo.POL	Cargo.POL = Origin.Details	
Cargo.POD	Destination.Details	Destination.Details = Cargo.POD	
Destination.Details	Cargo.POD	Cargo.POD = Destination.Details	
Cargo.Course	Freight.Course	Freight.Course = Cargo.Course	
Freight.Course	Cargo.Course	Cargo.Course = Freight.Course	

Credit.Period	Payment.Type	IF (Credit.Period $>$ 0) THEN
		Payment.Type = Credit
		ELSE
		Payment.Type = Cash-Against-Doc
		END IF
			Credit.Period $\geq$ 0

Payment.Type	Credit.Period	IF (Payment.Type = Credit) THEN
		Credit.Period $\in \{15.30, 45, 60\}$
		ELSE
		Credit.Period = 0
		END IF
			Payment.Type $\in$ {Credit,
			Cash-Against-Doc}

ShippingOrder.Status	Cargo.Status	Switch (ShippingOrder.Status) Case Fulfilled: Cargo.Status = Accomplished Case Created: Cargo.Status = Received End Case	ShippingOrder.Status $\in$ {Fulfilled, Created}
Cargo.Status	ShippingOrder.Status	Switch (Cargo.Status) Case Accomplished: ShippingOrder.Status = Fulfilled Case Received: ShippingOrder.Status = Created End Case	Cargo.Status $\in$ {Accomplished, Received}
Proposal.Status	Offer.Status	Offer.Status = Proposal.Status	Proposal.Status $\in$ {Sent,Approved}
Offer.Status	Proposal.Status	Proposal.Status = Offer.Status	Offer.Status $\in$ {Sent,Approved}

ShippingOrder.Status	Offer.Status	IF (ShippingOrder.Status = Approved) THEN
		Offer.Status = Accepted
		ElSE
		Offer.Status = Executed
		END IF
			ShippingOrder.Status $\in$ {Approved,
			Executed}

Offer.Status	ShippingOrder.Status	IF (Offer.Status = Accepted) THEN
		ShippingOrder.Status = Approved
		ElSE
		ShippingOrder.Status = Executed
		END IF
			Offer.Status $\in$ {Accepted, Executed}

Payment.Status	Cargo.Status	IF (Payment.Status = Received) THEN
		Cargo.Status = Accomplished
		END IF
			Payment.Status = Received

Cargo.Status	Payment.Status	IF (Cargo.Status = Accomplished) THEN
		Payment.Status = Received
		END IF
			Cargo.Status = Accomplished

Table 3.2: Substitutability Graph Segment for CargoTransportation *Operation*

need to adopt any changes in the adopted service matching techniques when new ontology mappings are added.

3.5 The G^+ Goal Model

This section introduces the G^+ model to provide the means to describe a goal, its achievement, and the corresponding required services' capabilities (functional features supported/offered by a service to achieve its goals) adopting the proposed meta-ontology.

A real-world status in a given application domain could be described via a set of facts over the concepts involved at a given point of time. As we describe facts via constraints, the real-world status could be represented by a set of constraints over the concepts involved at a given point of time. The achievement of a given goal implies a change in the real-world status. Such change could be captured by monitoring the constraints over the involved concepts at different time transition points; starting from the constraints over a set of initial concepts (before achieving the goal) reaching to the constraints over a set of final concepts (after achieving the goal); going through different sets of constraints over the appeared intermediate concepts. Hence, a goal achievement could be described via different sets of constraints appeared over the involved concepts at different transition points of time.

The constraints over the set of initial concepts are known as the goal *Pre-Constraints*. The constraints over the set of final concepts are known as the goal *Post-Constraints*. The appeared intermediate concepts are mainly based on the steps taken to achieve the goal. As goals are achieved by invoking services, achievement of a given goal could require certain capabilities to be supported/offered by the invoked services, such requirements will be described via a set of constraints over services' capabilities, known as the goal *Describing-Constraints*. Hence, describing a goal and its achievement requires capturing the pre-constraints, the post-constraints, the describing-constraints, the steps taken to achieve the goal and the constraints over the intermediate concepts. We introduce the G^+ model to fulfill these requirements.

Definition 11 (G^+ Model) *A goal G is defined in the G^+ format as $\langle Op, Ctxt, SN\rangle$, where Op is an operation representing G, $Ctxt$ is the achievement context of G defined as the tuple $\langle Pre, Desc, Post\rangle$, Pre is the set of pre-constraints (denoted as $Ctxt^{Pre}$), $Desc$ is the set*

of describing constraints (denoted as $Ctxt^{Desc}$) and Post is the set of the post-constraints (denoted as $Ctxt^{Post}$), and SN is the scenarios-network used for achieving G.

A G^+ model a goal by an application domain operation. It captures the goal pre-constrains, post-constraints, and describing-constraints in the goal's achievement context, known as the goal's High-Level Functional Context (HLFC). It adopts the notion of "scenario" to describe the steps taken to achieve the goal. A scenario shows only one story about how to achieve the goal, and it is modelled as a sequence of application domain operations. However, a goal could be described by multiple scenarios that can differ in their abstraction levels and could have different triggering constraints, forming a network of scenarios that plot the possible ways that a given goal can be achieved (see Figure 3.11). In G^+ model a scenario is *tightly coupled* with a goal model, that every operation in a given scenario is seen as a goal to be achieved and could be realized by another different scenarios. The constraints over the appeared intermediate concepts will be automatically extracted according to the defined scenarios-network and the defined HLFC (details in the next chapter).

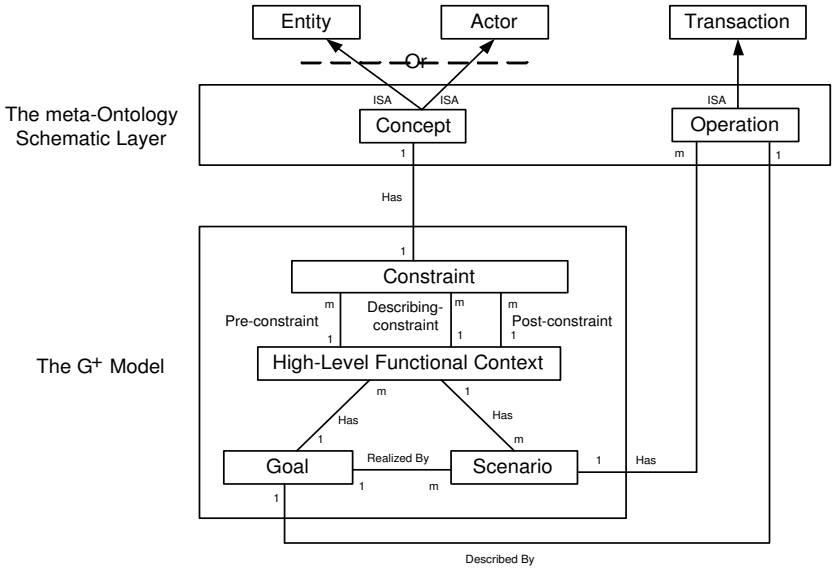

Figure 3.9: Relationships between the G^+ Model and the Meta-Ontology

Figure 3.9 indicates the relations between the components of a G^+ model and the components of the proposed meta-ontology. It shows that both goals and scenarios are described

via the application domain operations, while HLFCs are described via constraints that are based on the application domain concepts.

Users need to describe the required goals, the required scenarios for goals' achievement, and the required services' capabilities from their own perspective. Also, service providers need to describe the supported goals of their services, the supported scenarios for achievements of these goals, and the supported capabilities from their own perspective. Hence, there are two different perspectives for describing a goal and its achievement: a *User's Perspective* and a *Service's Perspective*. A comparison between the user perspective and the service perspective is given in Table 3.3.

Constraint Set	User Perspective	Service Perspective
Pre-Constraints	Guaranteed to be satisfied	Required to be satisfied
Post-Constraints	Required to be satisfied	Guaranteed to be satisfied
Describing-Constraints	Required to be satisfied	Guaranteed to be satisfied

Table 3.3: User Perspective versus Service Perspective

In the user's perspective, a user guarantees the satisfaction of the pre-constraints and requires the returned service to satisfy the required describing-constraints and the required post-constraints. However, in the service's perspective, a service guarantees the satisfaction of the describing-constraints and the post-constraints (after its successful execution), but it requires the pre-constraints to be satisfied in order to be invoked.

For example, a user who wants to transport a car cargo from `Melbourne-Australia` to `Alexandria-Egypt`. The user provides all the cargo details and the required international commercial terms (that is a world standard for transportation fees and charges). Constraints used to describe such information are guaranteed to be satisfied by the user, hence they should appear in his/her pre-constraints. However, he/she would like to have a service that can handle motor vehicles transportation and offer credit payment. Constraints used to describe such information are required to be satisfied by the service, hence they should appear in his/her describing-constraints. The user requires the transportation operation to be accomplished once he send the payments. Constraints used to describe such information are required to be satisfied after the service finishes its execution, hence they should appear in

his/her post-constraints. Also the user would like to carry out the cargo transportation in the following manner. He/She would like to send the cargo and charging scheme details first, then get an offer, negotiate it, then accept it. Finally, he/she would like the offer to be executed and send his/her payments. So such information should appear as the required scenario to achieve the goal. Figure 3.10 depicts a G^+ model for the user's goal. It indicates that the goal need to be achieved is a `Cargo Transportation` operation. The user wants to achieve this goal by the scenario depicted in the figure, also the user provides the required HLFC, where `Cargo.POL` is the cargo port of loading, `Cargo.POD` is the cargo port of discharge, `Cargo.Course` indicate the nature of the required transportation operation, `IncoTerm.Type` is the standard international commercial term, and `FOB` (Freight On Board) is its required value.

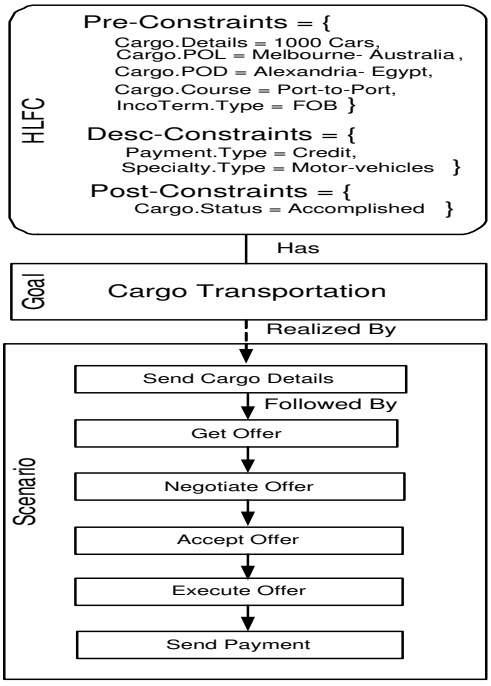

Figure 3.10: Example of a User G^+ Model

Another example from a service's perspective, a service provider that can do freight movement operations, and would like to advertise his/her service. The service provider requires some constraints to be satisfied in order to invoke the service, such constraints should

appear in the service pre-constraints. The supported/offered service's capabilities for freight movement operations should appear as the service describing-constraints. The information about the concepts appeared after the service finishes its execution should appear in the service post-constraints. Also different scenarios for achieving the goal should be defined. Figure 3.11 depicts a G^+ model for the service's goal. It indicates the service goal is *Freight Movement*, and also depicts the various scenarios (with different abstraction levels) required for realizing this goal.

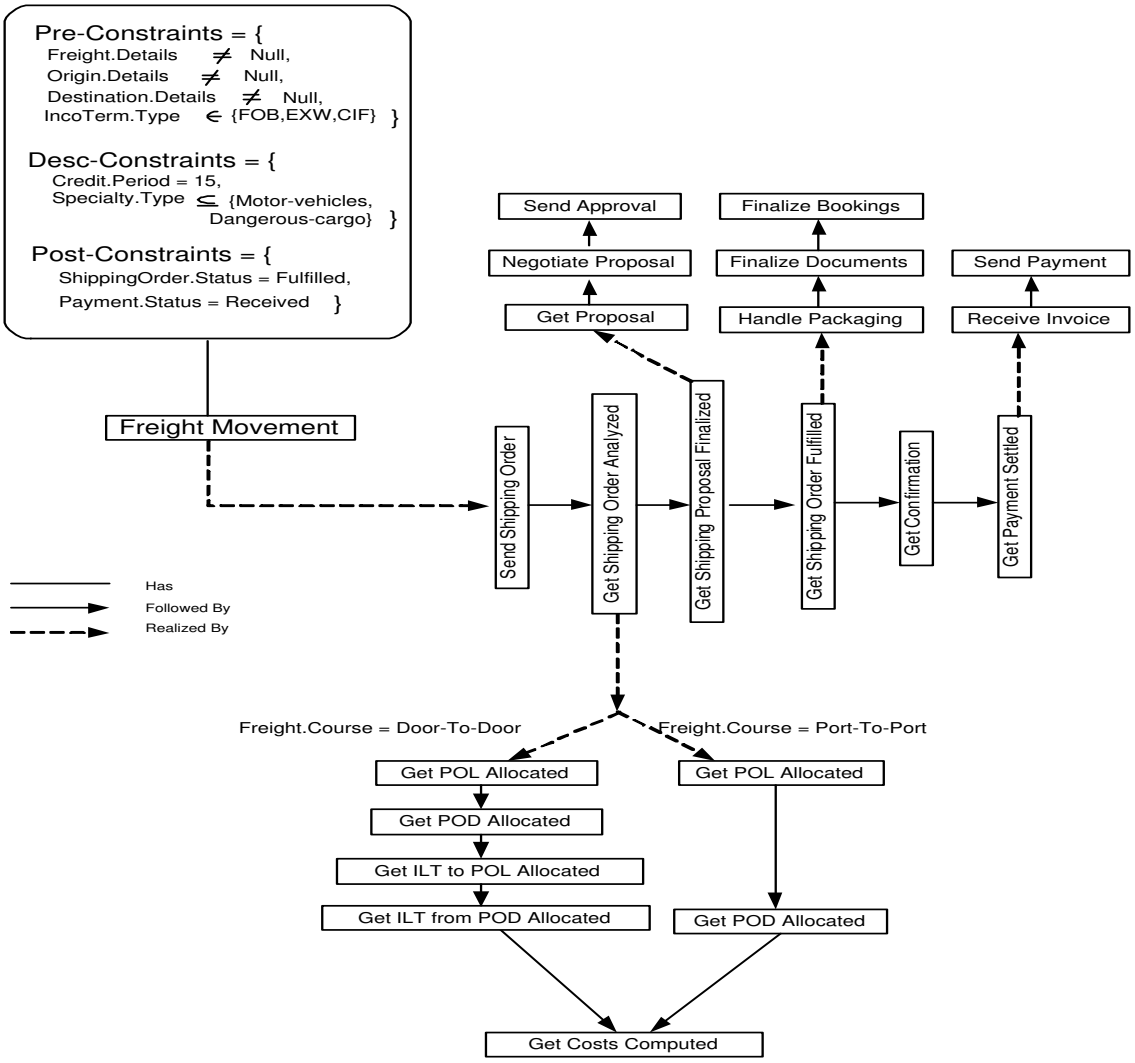

Figure 3.11: Example of a Service G^+ Model

For simplicity, we removed the *Port-To-Door*, and the *Door-to-Port* cases from the branching over the *Freight.Course* values, as the idea of branching and its different realization scenarios is clearly illustrated by the *Port-to-Port* and the *Door-to-Door* cases. In the *Door-to-Door* case additional In Land Transportation (ILT) operation is required. The service requires the `IncoTerm.Type` to be either a `FOB` (Freight On Board), `EXW` (EX-Works), or `CIF` (Cost insurance and Freight) in order to be invoked. The service offers a 15 days credit period, and can handle motor vehicles and dangerous cargo transportation.

Both users and service providers should choose the operations forming their scenarios from the adopted ontology. Their corresponding HLFCs should not violate the semantics of the chosen operations (their pre and post constraints defined in the adopted ontology). Table 3.4 indicates the pre-constraints and post-constraints of the operations selected by the user from the ontology to form the required scenario. Table 3.5 indicates the pre-constraints and post-constraints of the operations selected by the service provider from the adopted ontology to form the required scenarios-network. Hence, a correctness check must be done, to make sure correct scenarios are created, more details are given in the end of this chapter. For human users, a set of templates could be provided for the possible scenarios of achieving different application domain operations, a user can modify these templates as required to form his/her request (more details about the responsibilities of users and service providers are given in Appendix A).

As indicated earlier, users could specify two different categories of constraints to represent their desires: mandatory/essential and optional/preferred. Hence, each HLFC set of constraints could be divided into two subsets a mandatory/essential subset and an optional/preferred subset. However, in this thesis, we restrict user's constraints to be only mandatory/essential constraints. That any constraints given by the user (pre, post or describing) must be satisfied by the returned matching results. Also, a scenarios-network could contain *negative scenarios*, which are the scenarios a service should follow when an error or exception is occurred during its execution. In our approach, negative scenarios are handled exactly as the positive scenarios in terms of modelling and matching. Hence, without loss of generality, this thesis will focus only on *positive scenarios*.

Operation	Pre-constraints	Post-Constraints
SendCargoDetails	{Cargo.Details ≠ Null, Cargo.POL ≠ Null, Cargo.POD ≠ Null, IncoTerm.Type ≠ Null }	{Cargo.Status = Received}
GetOffer	{Cargo.Status = Received , Cargo.Course ≠ Null }	{Offer.Status = Sent}
NegotiateOffer	{Offer.Status = Sent}	{Offer.Status = Approved}
AcceptOffer	{Offer.Status = Approved}	{Offer.Status = Accepted}
ExecuteOffer	{Offer.Status = Accepted}	{Offer.Status = Executed}
Sendpayment	{Offer.Status = Executed}	{Cargo.Status = Accomplished}

Table 3.4: Part of the Ontology Operations' Definitions Selected by the User

Operation	Pre-constraints	Post-Constraints
SendShippingOrder	{Freight.Details ≠ Null, Origin.Details ≠ Null, Destination.Details ≠ Null, Freight.Course ≠ Null, IncoTerm.Type ≠ Null }	{ShippingOrder.Status = Created}
GetShippingOrderAnalyzed	{ShippingOrder.Status = Created}	{ShippingOrder.Status = Analyzed }
Get-POL-Allocated	{ShippingOrder.Status = Created}	{POL.Status = Allocated}
Get-POD-Allocated	{POL.Status = Allocated}	{POL.Status = Allocated, POD.Status = Allocated}
Get-ILT-To-POL-Allocated	{POL.Status = Allocated}	ILT.ToStatus=Allocated
Get-ILT-From-POD-Allocated	{POD.Status = Allocated}	ILT.FromStatus=Allocated
GetCostsComputed	{POL.Status = Allocated, POD.Status = Allocated}	{ShippingOrder.Status = Analyzed}
GetShippingProposalFinalized	{ShippingOrder.Status = Analyzed}	{ShippingOrder.Status = Approved}
GetProposal	{ShippingOrder.Status = Analyzed}	{Proposal.Status = Sent}
NegotiateProposal	{Proposal.Status = Sent}	{Proposal.Status = Approved}
SendProposal	{Proposal.Status = Approved}	{ShippingOrder.Status = Approved}
GetShippingOrderFulfilled	{ShippingOrder.Status = Approved}	{ShippingOrder.Status = Executed}
HandlePackaging	{ShippingOrder.Status = Approved}	{Packaging.Status = Accomplished}
FinalizeDocuments	{Packaging.Status = Accomplished}	{Documentation.Status = Accomplished}
FinalizeBookings	{Documentation.Status = Accomplished}	{ShippingOrder.Status = Executed}
GetConfirmation	{ShippingOrder.Status = Executed}	{ShippingOrder.Status = Confirmed}
GetPaymentSettled	{ShippingOrder.Status = Confirmed}	{ShippingOrder.Status = Fulfilled, Payment.Status = Received}
ReceiveInvoice	{ShippingOrder.Status = Confirmed}	{ShippingOrder.Status = Pending}
SendPayment	{ShippingOrder.Status = Pending}	{ShippingOrder.Status = Fulfilled, Payment.Status = Received}

Table 3.5: Part of the Ontology Operations' Definitions Selected by the Service Provider

When matching the services' constraints against the users' constraints, the matching direction should be from the party that guarantees the satisfaction of its constraints (the source) to the party that required its constraints to be satisfied (the target). As we have two different perspectives in defining a G^+ model, the direction in G^+ matching is based on the type of the defined constraints (pre, post and describing), as indicated in Table 3.6 (more details about service matching are given in Chapter 5).

Constraint Set	Source	Target
Pre-Constraints	User	Service
Post-Constraints	Service	User
Describing-Constraints	Service	User

Table 3.6: HLFC Matching

3.6 From a G^+ Model to a GAP-Forest

It is the responsibility of the matchmaker to analyze all the possible scenarios provided by users and service providers in order to find a mach. This section indicates how this could be achieved. One path from a goal node to a leaf operation node (no successors) represents one way to achieve the goal, such a path is known as a *Goal Achievement Pattern* (GAP) (see Figure 3.12). A GAP provides a global (end-to-end) snapshot of how the service's goal is expected to be accomplished. This global snapshot provides information that helps the matchmaker to anticipate the external behavior of the service in a given context in order to achieve the service goal. Extracting GAPs from a scenarios-network implies tracing all the possible paths in the defined network, as each different path represents a different GAP.

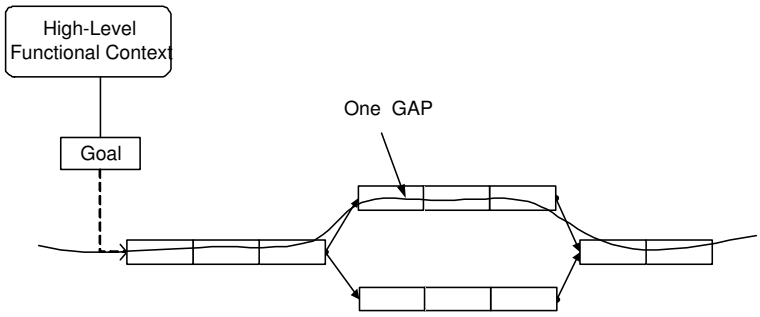

Figure 3.12: Example of a GAP

For each trace of the scenarios-network, a different set of branches are visited. At the point where a branch starts, a group of constraints must be true in order to visit that branch. This group of constraints acts as a *sub-context* for this GAP. This sub-context will be added to the pre-constraints of the high-level functional context of the G^+ model (we refer to this a conjunction to the HLFC). This forms the high-level functional context of the GAP.

For example, in Figure 3.11, we can identify two sub-contexts are formed when the values of `Freight.Course` is examined: one sub-context when (`Freight.Course = Door-to-Door`), and the other when (`Freight.Course = Port-to-Port`). Figure 3.13 depicts these formed GAPs and their corresponding high-level functional contexts. The figure indicates that the HLFCs of abstraction level-2 (corresponding to the GAPs resulting from the branching) are formed from the conjunction of HDLC of the G^+ model and the corresponding sub-contexts. A GAP is defined as indicated in Definition 12.

Definition 12 (Goal Achievement Pattern) *Given a goal* $G = \langle Op, Ctxt, SN \rangle$*, a goal achievement pattern of* G *is defined as* $Gp = \langle\ Op,\ GCtxt,\ OpSeq \rangle$*, where* $GCtxt = \langle Ctxt^{Pre} \wedge SubCtxt,\ Ctxt^{Desc},\ Ctxt^{Post}\ \rangle$*, and* $SubCtxt$ *is the conjunction of the constraints that must be satisfied along the path that forms the operations sequence* $OpSeq$ *resulting from tracing* SN*; starting from the goal node until a leaf operation node is reached.*

As a scenarios-network has scenarios defined in different abstraction levels, the extraction process of different GAPs from a given scenarios-network starts by tracing all of the possible paths in the scenarios-network, then replacing the operations by their realizing scenarios (if they exist). Every replacement process creates more detailed GAPs from the previously generated GAPs. Hence, every abstraction level has a group of GAPs that describes the realization of the goal at this abstraction level, as indicated in Figure 3.13. The figure indicates that the realization of the goal *Freight Movement* of the service G^+ model depicted in Figure 3.11 is described using three abstraction level.

The GAP of abstraction level-0 is simply representing the goal and its HLFC. Abstraction level-1 contains the GAPs that are directly realizing the goal, in this case there is only one GAP. The GAPs of abstraction level-2 are resulting from the GAPs of abstraction level-1 by replacing the GAPs operations by their direct realizing scenarios. When branching is visited during the extraction of the GAPs of abstraction level-2, the GAP of abstraction level-1 is duplicated, and for every copy the corresponding sub-context is added to the HLFC of abstraction level-1 GAP. The collection of these sets of GAPs forms the *GAP-Forest* corresponding to the G^+ model, as indicated in Definition 13.

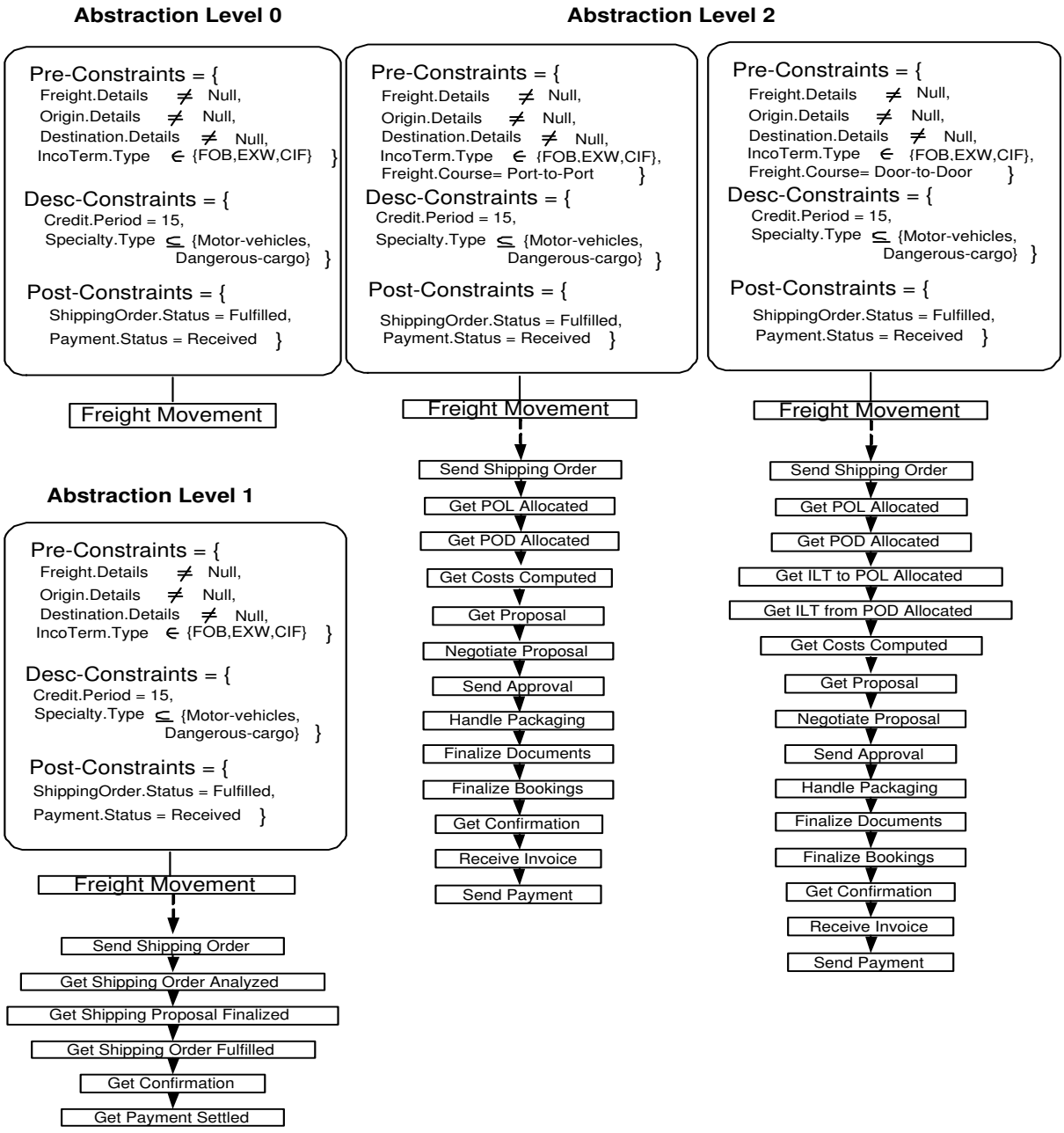

Figure 3.13: Example of a GAP Forest

Definition 13 (GAP Forest) *A GAP forest is defined as* $Gpf = \{\langle L_i, GAP_{L_i}\rangle \mid 0 \leq i \leq n\}$*, where* L_i *is the* i^{th} *abstraction level,* L_0 *represents the most abstracted realization level,* L_n *is the most detailed realization level, and* GAP_{L_i} *is the set of GAPs belongs to the abstraction level* L_i*.*

The process of GAP-forest extraction can be performed during the design time or during the matching time. However, in our approach, we apply this extraction process during the design time in order to simplify the matching process, and to improve the matching response time. Hence, the matching process between two G^+ models is actually a matching process between their corresponding GAP-forests (all the extracted GAPs). When a GAP forest of a given G^+ model has only one GAP, this G^+ model is known as a *simple* G^+ *model.*

3.6.1 GAP Correctness

As users and service providers create their scenarios from selecting operations from the adopted application domain ontology, this section defines a GAP correctness criterion that must hold in order to use a given GAP in the automated matching process.

The operations sequence of a given GAP can be seen as a transformation function that transforms the pre-constraints of the GAP ($GCtxt^{Pre}$) into the corresponding post-constraints ($GCtxt^{Post}$) via a number of transition points such that every transition point has a corresponding set of constraints, as indicated in Figure 3.14.

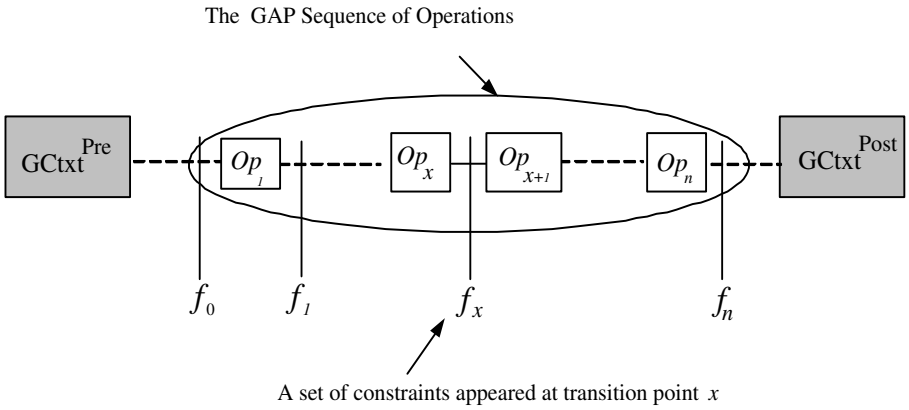

Figure 3.14: GAP Transformation Function

Such transformation function is formally defined as $(\sigma : \Delta \rightarrow \Delta)$ such that $\sigma(f_0) = f_n$, where $f_0, f_n \subseteq \Delta$, and Δ is the set of all possible constraints, f_0 is the $GCtxt^{Pre}$, and f_n is the $GCtxt^{Post}$. As σ consists of a sequence of operations, every transition point x between operations Op_x and Op_{x+1} has a set of constraints f_x resulting from invoking all the preceding operations[4].

To invoke an operation, its pre-constraints must be satisfied, hence a GAP is considered correctly defined when $GCtxt^{Pre}$ is transformed into $GCtxt^{Post}$ via σ, and every set of constraints at a given transition point satisfying the pre-constraints of its following operation until $GCtxt^{Post}$ is reached. Otherwise the GAP is considered incorrectly defined as the invocation of its operations cannot be guaranteed. We formally define a GAP correctness as follows.

Definition 14 (GAP Correctness) *Given a goal achievement pattern* $Gp = \langle\, Op, GCtxt, OpSeq \,\rangle$, *Gp is considered correctly defined if* $(\forall_{x=0}^{x=n-1}\ f_x \vDash Op_{x+1}^{Pre}) \wedge (f_n \vDash GCtxt^{Post})$, *where n is the number of operations in OpSeq,* $f_0 = GCtxt^{Pre}$, *and* f_x *is the set of constraints at the transition point x between* Op_x *and* Op_{x+1}, *such that* $Op_x, Op_{x+1} \in OpSeq$, *and* $\vDash$ *is the constraint satisfiability operator.*

Definition 14 requires determining constraints satisfiability. A constraint $Cnst_i$ satisfies a $Cnst_j$ when $Cnst_i$ subsumes $Cnst_j$, which could be determined via constraint implication [Jeavons and Cooper, 1995; Pearson and Jeavons, 1997]. However, existing approaches for determining constraint implication (such as the ones discussed in [Jeavons and Cooper, 1995; Pearson and Jeavons, 1997]) require the constraints to have the same scope to be able to determine their implication. We summarize such approach in the following proposition. However, we will extend such approach later in the next chapter to determine constraint satisfiability between constraints with different scopes.

Proposition 1 (Constraint Satisfiability) *Given two constraints* $Cnst_i$ *and* $Cnst_j$, $Cnst_i$ *satisfies* $Cnst_j$ *(denoted as* $Cnst_i \vDash Cnst_j$*) if* $(\Xi(Cnst_i) = \Xi(Cnst_j)) \wedge (Cnst_i \Rightarrow Cnst_j)$.

[4]The formula for computing f_x is given in Chapter 4.

Proof: *$Cnst_i$ satisfies $Cnst_j$ when $Cnst_i$ subsumes $Cnst_j$. As implication is one of the common methods for determining constraint subsumption, $Cnst_i$ satisfies $Cnst_j$ when ($Cnst_i \Rightarrow Cnst_j$). Also, as implication considers any constraints with different scopes as independent constraints [Jeavons and Cooper, 1995; Pearson and Jeavons, 1997], we require the constraints to have the same scopes in order to be examined for satisfiability.*

Proposition 1 indicates that a constraint $Cnst_i$ (a source constraint) satisfies a constraint $Cnst_j$ (a target constraint) when the constraints have the same scope, and when the source constraint implies/subsumes the target constraint. For example, the constraint (`Cargo.Weight` $>$ 100) satisfies the constraint (`Cargo.Weight` $>$ 10), as (`Cargo.Weight` $>$ 100) implies (`Cargo.Weight` $>$ 10). Similarly, by applying the standard implication rules, the constraint (`Cargo.Weight` $\subseteq \{100, 200\}$) satisfies the constraint (`Cargo.Weight` $=$ 100). While, the constraint (`Cargo.Weight` $=$ 100) does not satisfy the constraint (`Cargo.Weight` $\subseteq \{100, 200\}$), but it satisfies the constraint (`Cargo.Weight` $\in \{100, 200\}$). Determining constraints satisfiability is the basic building block of the service matching process, as services' descriptions and users' requests contain different types of constraints sets, we will need to determine the satisfiability status between two sets of constraints. Constraints-sets satisfiability is a complex process, as the there could exists different types of correlations between sets' elements. For example, the conjunction of two source constraints could satisfy a target constraint, while the target constraint cannot be satisfied by either source constraints individually. Another example that shows how complex is the process is that when a conjunction of multiple source constraints satisfies a conjunction of multiple target constraints, where source constraints cannot satisfy target constraints individually. So to simplify things we assume a set of constraints is satisfied when each of its elements is satisfied individually, as indicated in Assumption 1.

Assumption 1 (Constraint-Set Satisfiability) *Given a set of constraints f_i, f_i is satisfied if $\forall$ $Cnst_u$, $Cnst_u \in f_i$, $Cnst_u$ is satisfied by only one constraint at a time.*

By Adopting Assumption 1, satisfiability between two sets of constraints is simply determined by individually determining the satisfiability status between their elements, as follows.

Proposition 2 (Constraint-Set Satisfiability) *Given two sets of constraints f_i and f_j, f_i satisfies f_j (denoted as $f_i \vDash f_j$) if $(\Xi(f_i) \supseteq \Xi(f_j))$ and $\forall\ Cnst_u \in f_j$, $\exists\ Cnst_v \in f_i$ such that $(Cnst_v \vDash Cnst_u)$.*

Proof: *As the constraints of any given set will be examined individually (according to Assumption 1), f_j can be satisfied when each constraint of its elements has a corresponding constraint in f_i with the same scope and satisfied by such constraint.*

For example, the set {`Cargo.Weight` = 100, `Cargo.Type` = Cars, `Cargo.Course` = Port-to-Port} satisfies the set {`Cargo.Weight` > 10, `Cargo.Type` ∈ {Cars, Buses, Trucks}}. According to Proposition 2, f_i could have more constraints than f_j, and the scopes of these constraints do not belong to the scope of f_j, we define these constraints as the *semantic difference* between f_i and f_j. We use the concept of semantic difference to compute behavior models' states (more details are in the next chapter).

Definition 15 (Semantic Difference) *Given two sets of constraints f_i and f_j such that f_i and f_j belong to the same GAP. The semantic difference between f_i and f_j (denoted as $f_i \diamond f_j$) is f_k, where f_k is the maximal subset of f_i such that $(f_i - f_k) \vDash f_j$.*

For example, the semantic difference between {`Cargo.Weight` = 100, `Cargo.Type` = Cars, `Cargo.Course` = Port-to-Port}, and {`Cargo.Weight` > 10, `Cargo.Type` ∈ {Cars, Buses, Trucks}} is {`Cargo.Course` = Port-to-Port}. The semantic difference between two sets of constraints is always unique, as indicated in Theorem 1.

Theorem 1 (Semantic Difference Uniqueness) *The semantic difference between two sets of constraints f_i and f_j is always unique.*

Proof: *In order to determine the satisfiability of f_j, f_i will be divided into two independent subsets with no common elements, a subset that have the same scope as f_j, and a subset that has completely different scope from f_j. As the subset that has completely different scope from f_j is a maximal subset, this subset will always be unique.*

Determining the satisfiability between two sets of constraints using Proposition 2 is mainly based on syntactical matching for their scopes. Such approach could be acceptable when the

constraints belong to the same GAP, as common vocabulary provided by the adopted ontology is used. But adopting the same approach when examining the satisfiability of constraints belonging to different GAPs (for example, a service GAP and a request GAP) is considered a very strict approach that leads to the appearance of false negatives. As this assumes that both services' descriptions and users' requests are defined via the same vocabulary, which is not a practical assumption. To overcome such problem we propose the concept of *constraint substitutability* in the next chapter. It determines the satisfiability between constraints with different scopes using the captured semantics of the involved application domain. Also, Definition 14 will be extended in Chapter 5 to determine the correctness of a service's GAP with respect to users' goals.

3.7 Conclusion

This chapter identified the characteristics required to be supported in the matching framework, enabling the matchmaker to automatically determine the correctness of the matching results. We illustrated the limitations of existing matching framework. We proposed the SWSMF framework to be the matching framework for the semantic web services. We proposed a meta-ontology for application domains in order to overcome the semantic interoperability problem. We proposed the concepts substitutability graph to capture the functional substitution semantics of application domains' concepts. We proposed the G^+ model to capture the goal-based high-level functional specifications for both services and users, adopting the concept of scenarios-network. Finally, we indicated how different goal achievement patterns will be generated from the G^+ model providing the criteria for GAP correctness. SWSMF succeed to fully support all the characteristics required in a matching framework that enables the matchmaker to automatically determine the correctness of the matching results. A comparison between the SWSMF framework and existing matching frameworks is given in Table 3.7, adopting the evaluation approach discussed in Section 3.2.

Characteristic	UDDI	EbXML	E-speak	LARKS	OWL-S	WSMO	SWSMF
1- **Separates Semantics From Presentation**	N	N	F	F	F	F	F
2- **Handles Semantic Interoperability**	N	N	F	F	P	F	F
3- **Adopts Goal-Based Matching**	N	N	N	N	N	F	F
4- **Adopts Role-Based Matching**	N	N	N	N	N	F	F
5- **Adopts External Behaviors**	N	N	N	N	N	N	F
6- **Adopts Different Contexts**	N	S	N	S	N	S	F
7- **Facilitates Service Aggregation**	N	N	N	N	N	F	F
8- **Supports non-functional/non-technical req.**	S	S	N	N	P	F	F
9- **Adopts User Semantics**	N	N	N	N	N	F	F

Table 3.7: Comparison between SWSMF and Existing Matching Frameworks

Chapter 4

The Functional Substitutability Matching Scheme

Capturing the semantics of web services, users, and application domains in a machine-understandable format is not enough to enable the matchmaker to automatically determine the correctness of the matching results, as the matchmaker needs to know how to use these semantics to determine the correct results. The matchmaker needs to adopt various matching schemes that specify how these semantics will be used within the matching process. As the focus is on the high-level functional specifications, this chapter proposes a matching scheme for matching the high-level functional specifications of semantic web services, known as the Functional Substitutability Matching Scheme (FSMS). FSMS uses *substitutability* as the matching rule and *goal achievement* as the correctness criterion. Finally, this chapter provides the algorithms required for realizing FSMS. These algorithms are the basic building blocks required for implementing the direct and the aggregate service matching techniques.

4.1 Preliminaries

A given specifications' type can be matched using different matching rules. For example, the high-level functional specifications can be matched either by using strict syntactic matching rules or by using flexible semantic matching rules. When a matchmaker uses a matching

rule that is different from the one required or anticipated by users, this will lead to matching results that may not achieve users' goals. Hence, the matchmaker needs to know which matching rules are required to be applied in order to make sure that the returned matching results achieve the required users' goals. Therefore, we propose the concept of a *matching scheme* to regulate the matching process, it specifies the comparison aspect, the matching rule, and the correctness criterion that must be adopted by the matchmaker for matching a given type of specifications (see Figure 4.1).

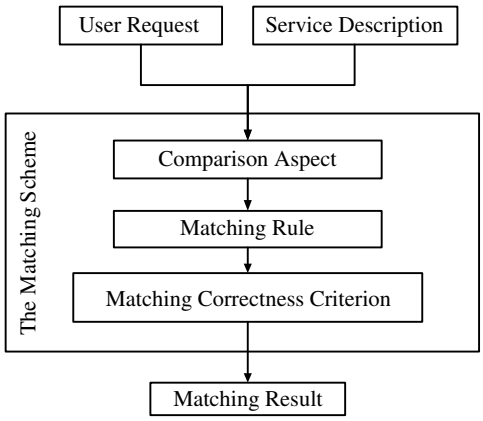

Figure 4.1: The Matching Scheme Architecture

We define a matching scheme as a tuple ⟨comparison aspect, matching rule, correctness criterion⟩. The comparison aspect component indicates the type of specifications to be examined. The matching rule component shows how a given comparison aspect is going to be examined. The correctness criterion component validates the results returned after applying the matching rule. According to this architecture, the matching process works for each type of specifications in the following manner (see Figure 4.1): the information related to the comparison aspect is extracted from users' requests and from services' descriptions. Later the matching rule is applied over the extracted information. Finally, the obtained answers are checked using the correctness criterion in order to return only the correct answers that fulfill users' goals.

As there exist different types of specifications, users need to specify which matching scheme should be used for each type of specifications, such that the matchmaker knows which

matching scheme will be applied when examining a given type of specifications. By doing so, both the matchmaker and users will have the same understanding of how the matching process will be carried out, hence the matchmaker will be able to eliminate any matching results that cannot achieve users' goals. As this thesis is primarily concerned with the high-level functional matching, the proposed matching scheme is about the high-level functional specifications, which we call as the Functional Substitutability Matching Scheme (FSMS). Section 4.2 discusses the components of FSMS, providing the algorithms required for realizing the FSMS. These algorithms must be adopted by any service matching technique that adopts FSMS. Section 4.3 indicates how FSMS is used to match external behavior models. Finally, Section 4.4 concludes the chapter.

4.2 FSMS Components

FSMS uses *Functionality* as the comparison aspect, where services are examined according to their capability of accomplishing a given function required for achieving users' goals. Information about a service functionality is extracted from its G^+ model. Similarly, information about the required users' functionality is extracted from the G^+ model provided by the users.

As users are interested in services that can achieve their goals the way they specified, any service with a functionality that can substitute the functionality specified by users will be considered a correct match. Hence, FSMS uses *Substitutability* as the matching rule. As a service is considered a correct match only when the users' goals are achieved after it finishes its execution, FSMS uses *User Goal Achievement* as the correctness criterion. we therefore define FSMS as $\langle Functionality, Substitutability, UserGoalAchievement \rangle$. When FSMS is adopted, the matchmaker will check whether the functionality specified by a user can be substituted by the functionality of a given web service. Furthermore, the matchmaker must determine whether users' goals are achieved or not by the returned matching results without executing any of the returned services.

In what follows we discuss the components of FSMS in more detail, indicating how the comparison aspect, the matching rule and the correctness criterion will be applied in the matching process.

4.2.1 Functionality as a Comparison Aspect

Functionality of a web service is mostly determined according to its behavior for achieving its goal. Hence, capturing such behavior in a machine-understandable format is mandatory for representing the service functionality. The behavior model will be extracted from the extracted GAP-forest. A behavior model corresponding to a given GAP is determined according to the GAP's dynamic component, which is the corresponding sequence of operations.

Behavior Model Extraction: When an operation is invoked, it changes the facts about the involved concepts, either by changing their attributes' values, by creating new concepts, or by vanishing some concepts (that appeared in its input and did not appear in its output). By monitoring the constraints representing the facts about the involved concepts appearing before and after the invocation of a given operation, we can determine the effect of invoking such an operation over the involved concepts, which reflects the behavior of the operation.

In general, when a given operation Op_x successfully finishes its execution, the constraints at this point of time can be classified as:

- *New*: the constraints created over new concepts generated by Op_x and that appeared in its post-constraints,
- *Evolved*: the constraints created by the operation Op_x over the input concepts and that appeared in its post-constraints,
- *Unchanged*: the constraints that appeared in Op_x pre-constraints and still appear in Op_x post-constraints, and
- *Independent*: the constraints that appeared before invoking Op_x, but do not appear in Op_x^{Pre} nor appear in Op_x^{Post}.

The union between the new, evolved, unchanged, and independent constraints represents the set of the persisting constraints after Op_x finishes its execution. The set of persisting constraints at a transition point x is denoted as f_x.

Definition 16 (Persisting Constraint) *A persisting constraint at a given transition point x is a constraint that must be satisfied at x.*

In order to have a correctly defined GAP ($Gp = \langle Op, GCtxt, OpSeq \rangle$), as indicated in Definition 14 (see Section 3.6.1), f_x must satisfy the pre-constraints of operation Op_{x+1}. However, when f_x satisfies Op_{x+1}^{Pre}, this does not mean all the constraints in f_x are needed to satisfy Op_{x+1}^{Pre}, as it might contain constraints that are independent from Op_{x+1}^{Pre}. Therefore we classify the constraints of f_x into *effective constraints* (denoted as f_x^e) and *idle constraints* (denoted as f_x^i). An effective constraint at transition point x is a constraint that can affect the invocation of Op_{x+1} by having a semantically related scope to the scopes of the pre-constraints of Op_{x+1}, as indicated in Definition 17. While an idle constraint at transition point x is a constraint that cannot affect the invocation of Op_{x+1} by having an independent scope from the scopes of the pre-constraints of Op_{x+1}, as indicated in Definition 18.

Definition 17 (Effective Constraint) *Given a constraint $Cnst_i$ such that $Cnst_i \in f_x$, $Cnst_i$ is considered an effective constraint at a transition point x when $\Xi(Cnst_i) \in \Xi(Op_{x+1}^{Pre})$.*

Definition 18 (Idle Constraint) *Given a constraint $Cnst_i$ such that $Cnst_i \in f_x$, $Cnst_i$ is considered an idle constraint at a transition point x when $\Xi(Cnst_i) \notin \Xi(Op_{x+1}^{Pre})$.*

According to Table 3.4 (page 67), in the user G^+ model depicted in Figure 3.10 (page 64), we can see that the constraint {`Cargo.Course = Port-to-Port`} (that is belonging to the context pre-constraints) is an idle constraint at transition point 0 (that is before operation `SendCargoDetails`), while the constraints {`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `IncoTerm.Type` = FOB} are effective.

Figure 4.2 shows the various types of constraints appearing before invoking Op_{x+1} and after executing Op_x. It indicates that f_x^e should satisfy Op_{x+1}^{Pre} in a correctly defined GAP, while f_x^i will be part of the persisting constraints at the next transition point.

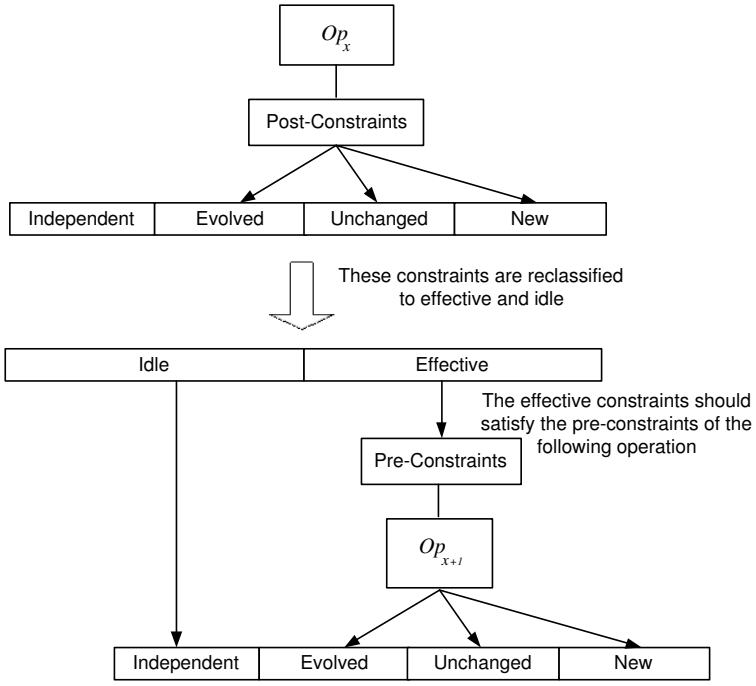

Figure 4.2: Relation between the Different Types of Constraints

Theorem 2 shows how the persisting constraints at a given transition point are computed. Theorem 2 indicates that the persisting constraints at a given transition point are computed as the union between the idle constraints at the transition point and the post-constraints of the operation following this transition point. Table 4.1 shows an example of the distribution of the persisting constraints of the user G^+ model shown in Figure 3.10.

Theorem 2 (Persisting Constraints Correctness) *Given a correct GAP defined as* $Gp = \langle Op, GCtxt, OpSeq \rangle$*, the set of persisting constraints* f_{x+1} *at transition point* $(x+1)$ *is correctly computed if* $f_{x+1} = f_x^i + Op_{x+1}^{Post}$*, where* $0 \leq x \leq n-1$*,* n *is the number of operations in OpSeq,* $(f_0 = GCtxt^{Pre})$*,* $(f_x^i = f_x \diamond Op_{x+1}^{Pre})$*, and* $(f_x^e = f_x - f_x^i)$*.*

Proof: *When a service is invoked, the constraints of* $GCtxt^{Pre}$ *are guaranteed to be satisfied. Hence, the persisting constraints before starting the invocation of the corresponding sequence of operations will equal to* $GCtxt^{Pre}$*, hence* $f_0 = GCtxt^{Pre}$*...(1). As Gp is correctly defined,* $(\forall_{x=0}^{x=n-1}\ f_x \vDash Op_{x+1}^{Pre})$ *according to Definition 14, and* f_x^e *is the subset that satisfies* Op_{x+1}^{Pre}*,* f_x^i*,* f_x^e *are determined according to Definitions 15, 17, 18...(2). At a given transition*

point x, f_x will be divided into two independent subsets f_x^e and f_x^i, hence ($f_x^e = f_x - f_x^i$)...(3). Also, at a given transition point x, f_x^e will be consumed by Op_{x+1}, resulting Op_{x+1}^{Post}, while f_x^i remains unchanged. Therefore, at the transition point (x+1), the persisting set of constraints will be formed from f_x^i and Op_{x+1}^{Post}, as indicated in Figure 4.2, hence $f_{x+1} = f_x^i + Op_{x+1}^{Post}$...(4).

Transition Point	**Persisting Constraints**	**Effective Constraints**	**Idle Constraints**
0	{`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `Cargo.Course` = Port-to-Port, `IncoTerm.Type` = FOB}	{`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `IncoTerm.Type` = FOB}	`Cargo.Course` = Port-to-Port
1	{`Cargo.Course` = Port-to-Port, `Cargo.Status` = Received}	{`Cargo.Course` = Port-to-Port, `Cargo.Status` = Received}	
2	{`Offer.Status` = Sent}	{`Offer.Status` = Sent}	
3	{`Offer.Status` = Approved}	{`Offer.Status` = Approved}	
4	{`Offer.Status` = Accepted}	{`Offer.Status` = Accepted}	
5	{`Offer.Status` = Executed}	{`Offer.Status` = Executed}	
6	{`Cargo.Status` = Accomplished}	{`Cargo.Status` = Accomplished}	

Table 4.1: Persisting Constraints Distribution

In service direct matching, we consider all the persisting constraints at the last transition point as *effective* in order to be considered during behavior matching (more details are given in Section 4.3). However, Theorem 2 as well as Definitions 17 and 18 will be extended in Chapter 5 to compute the set of persisting constraints with respect to users' goals. However, in general the persisting constraints at every transition point represent a behavior state of the service. As the effective constraints and the idle constraints represent different semantics, we need to differentiate between them in the behavior state model. For this reason, we define a state as follows.

Definition 19 (Behavior State) *A behavior state S_x at a transition point x between operations Op_x and Op_{x+1} is defined as the tuple $\langle f_x^e, f_x^i \rangle$.*

For example, the state S_0 in Table 4.1 is $\langle$ {`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `IncoTerm.Type` = FOB}, {`Cargo.Course` = Port-to-Port }$\rangle$.

The behavior of a web service according to one of its GAPs is constituted by tracing the persisting constraints at the transition points of the corresponding sequence of operations. This creates a sequence of states defining the behavior of a web service according to a given GAP.

Definition 20 (Behavior Model) *A behavior model of a web service according to a given GAP (denoted as β) is defined as the sequence of states $\langle S_0, S_1, ..., S_n \rangle$, where n is the number of operations defined in the GAP.*

The adopted behavior model is a linear model, as any nonlinearity is eliminated when GAPs are extracted to simplify the matching process. A linear behavior model must be extracted for every goal achievement pattern in the extracted GAP-forest of a service. A behavior model defined according to Definition 20 has a corresponding function σ that transforms the context's pre-constraints into its corresponding post-constraints (σ is discussed before in subsection 3.6.1, page 71). As a high-level functional context contains three different sets of constraints: pre-constraints, post-constraints, and describing-constraints, and the extraction of a behavior model is based only on the pre-constraints and the post-constraints of the context, we represent the functionality of a given service according to Definition 21 to take the describing-constraints into consideration.

Definition 21 (Service Functionality) *Given a goal achievement pattern $Gp = \langle Op, GCtxt, OpSeq \rangle$, the functionality of a web service according to Gp is defined as $\langle GCtxt^{Desc}, \beta \rangle$, where $GCtxt^{Desc}$, β are the describing-constraints and the behavior model of Gp, respectively.*

For example, the required functionality of the user G^+ model depicted in Figure 3.10 is indicated in Table 4.2. First, the matchmaker requires determining the functionality needed by users as well as the functionality of each GAP of a given service (Definition 21). Then the matching rule is applied over these extracted functionalities to find the correct matching result.

Describing-Constraints	{`Specialty.Type` = Motor-vehicles, `Payment.Type` = Credit}
S_0	⟨ {`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `IncoTerm.Type` = FOB}, {`Cargo.Course` = Port-to-Port}⟩
S_1	⟨ {`Cargo.Course` = Port-to-Port, `Cargo.Status` = Received} , {} ⟩
S_2	⟨ {`Offer.Status` = Sent}, {} ⟩
S_3	⟨ {`Offer.Status` = Approved}, {} ⟩
S_4	⟨ {`Offer.Status` = Accepted} , {} ⟩
S_5	⟨ {`Offer.Status` = Executed} , {} ⟩
S_6	⟨ {`Cargo.Status` = Accomplished} , {} ⟩

Table 4.2: User Required Functionality

4.2.2 Substitutability as a Matching Rule

As the functionality required by a user is represented via different sets of constraints (describing-constraints, pre-constraints, post-constraints and persisting constraints), the matchmaker tries to find a service that can satisfy these constraints. Hence, the service matching problem is mapped into a constraint satisfiability problem. When examining the satisfiability between user's constraints and service's constraints, we have to take into consideration that the direction of the satisfiability, as it varies according to the type of involved constraints (that is describing-constraints, pre-constraints, post-constraints and persisting constraints). As indicated in Chapter 3, G^+ models are defined according to two different perspectives: a user's perspective and a service's perspective (see Table 3.3 for more details). That, the user's pre-constraints will be checked if they can satisfy the service's pre-constraints, the service's describing-constraints will be checked if they can satisfy the user's describing-constraints, and the service's post-constraints will be checked if they can satisfy the user's post-constraints. So in general, we need to determine if a given set of constraints (that is known as the *source constraint-set*) can satisfy another set of constraints (that is known as the *target constraint-set*). In what follows, we show how this will be determined.

As we indicated before, determining constraint satisfiability using Proposition 1 is a very strict approach when adopted to match service's constraints and user's constraints, as it requires the constraints to have the same scope. For example, a service's functional scope could be described via the constraint (`City.name = Cairo`), and the user's required functional scope described via the constraint (`Country.name = Egypt`). In spite of the constraint (`City.name = Cairo`) satisfies the constraint (`Country.name = Egypt`), Proposition 1 cannot identify this case, as it depends on the existing techniques for determining constraint implication such as the ones described in [Jeavons and Cooper, 1995; Pearson and Jeavons, 1997], which consider these constraints as independent constraints as they have different scopes. We identify the constraint satisfiability approaches that require the source constraint and the target constraint to have the same scope as *Constraint Direct Satisfiability* approaches.

To overcome the limitation of these direct satisfiability approaches, we propose the concept of *Constraint Indirect Satisfiability*. The idea behind constraint indirect satisfiability is to find a transformation $\urcorner$ using the semantics of the involved application domain in order to mediate between the constraints when they have different scopes. The transformation transforms the *source constraint* ($Cnst_i$) into an *intermediate constraint* ($Cnst_k$) when a given set of *pre-constraints* are satisfied (which are defined according to the semantics of the involved application domain) such that the intermediate constraint will have the same scope as the *target constraint*($Cnst_j$). By doing so, existing techniques for determining constraint direct satisfiability could be used to determine the satisfiability status between the generated intermediate constraint and the target constraint. When the intermediate constraint satisfies the target constraint, this indicates that the source constraint indirectly satisfies the target constraint, as indicated Definition 22.

Definition 22 (Constraint Indirect Satisfiability) *Given two constraints $Cnst_i$ and $Cnst_j$ such that $\Xi(Cnst_i) \neq \Xi(Cnst_j)$, $Cnst_i$ indirectly satisfies $Cnst_j$ if there exists a transformation $\urcorner$ that transforms $Cnst_i$ into $Cnst_k$ such that $Cnst_k \vDash Cnst_j$.*

Figure 4.3 depicts the difference between direct and indirect constraint satisfiability. It indicates that when the source constraint $Cnst_i$ has a different scope from the target constraint $Cnst_j$, the source constraint will be transformed into an intermediate constraint $Cnst_k$ using a transformation $\urcorner$ such that existing direct constraint satisfiability techniques can be used to check the satisfiability of the target constraint by the intermediate constraint.

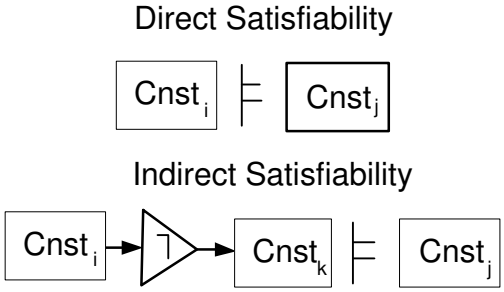

Figure 4.3: Direct versus Indirect Constraint Satisfiability

Finding such a transformation ($\urcorner$) is not an easy task, as there could be an infinite number of transformations that can transform one constraint into another. Therefore, finding such a transformation must be based on the semantics of the involved application domain, then it must be validated according to the achievement semantics of the required goals in order to make sure that the invocation of such a transformation does not violate any of the involved semantics (that is semantics-preserving transformation). First, we provide a general definition for a transformation, as indicated in Definition 23. Then we will provide the criteria that must hold in order to consider a transformation as a valid semantics-preserving transformation. As only valid semantics-preserving transformations must be used during the service matching process.

Definition 23 (Transformation) *A transformation* $\urcorner$ *is a function* $(\Delta \times 2^{\Delta} \rightarrow \Delta)$ *such that* $\urcorner(S, P) = T$*, where* Δ *is the set of all possible constraints,* S *is the source constraint* $(S \in \Delta)$*,* T *is the target constraint* $(T \in \Delta)$*,* P *is the set of pre-constraints that must be satisfied before invoking the transformation* $(P \subseteq \Delta)$*,* $(S \notin P)$*, and* $(S \neq T)$*.*

As substitutability is the adopted matching rule, the substitution semantics between the concepts of application domains need to be captured in a machine-understandable format such that they can used by the matchmaker to generate the possible transformations that can be used to determine constraints indirect satisfiability, then determines the validity of such transformations before invoking them. We will show how the concepts substitutability graph (CSG), introduced in Chapter 3, is used to generate these transformations.

As a CSG contains information that indicates which scopes can be substitutable with respect to application domain operations, also it indicates the mapping between scopes' values via the defined conversion functions, and the mappings between different operators via the defined operator mapping matrices. By using the scopes of the source and target constraints and the concepts substitutability graph, one will be able to determine if there exists a possible transformation that can transform the source constraint into the target constraint via an intermediate constraint that has the same scope as the target constraint. This is performed by checking if there exists a path from the scope of the source constraint into the scope of the target constraint in the segment of the involved user's goal. Having no path indicates that the scope of the target constraint cannot be substituted by the scope of the source constraint with respect to the captured semantics, meaning there is no transformation that can be found to transform the source constraint into the target constraint according to the captured semantics. In this case, the source and target constraints are considered independent. However, when there exists a path from the source scope into the target scope, determination of the transformation depends on the reachability status between the scopes. When the path contains only one edge, this is known as *Direct Reachability*. When the path contains more than one edge, this is known as *Indirect Reachability*.

Definition 24 (Scopes Direct Reachability) *Given a scope* Ξ_i*, a scope* Ξ_j*, a goal* $G = \langle Op, Ctxt, SN\rangle$ *, and a concepts substitutability graph* $CSG = \bigcup_{u=1}^{n}\{\langle Op_u, V_C, E_C\rangle\}$*.* Ξ_i *is directly reachable to* Ξ_j *with respect to* G *(denoted as* $\Xi_i \rightharpoonup_G \Xi_j$*) when* $\exists Seg,\ Seg \in CSG$ *such that* $Op_u = Op$*, and* $\exists E_{(i,j)},\ E_{(i,j)} \in E_C$ *in the corresponding* Seg*, where* $E_{(i,j)} = \langle V_i, V_j, \Pi_{(i,j)}, \Psi_{(i,j)}, \Omega_{(i,j)}\rangle$ *such that* $(V_i = \Xi_i)$ *and* $(V_j = \Xi_j)$*.*

Definition 25 (Scopes Indirect Reachability) *Given a scope Ξ_i, a scope Ξ_j, a goal $G = \langle Op, Ctxt, SN \rangle$, and a concepts substitutability graph $CSG = \bigcup_{u=1}^{n} \{\langle Op_u, V_C, E_C \rangle\}$. Ξ_i is indirectly reachable to Ξ_j with respect to G (denoted as $\Xi_i \curvearrowright_G \Xi_j$) when $\exists Seg$, $Seg \in CSG$ such that $Op_u = Op$, and $\exists$ a path $P = (E_{(i,k_1)}, E_{(k_1,k_2)}, ..., E_{(k_{m-1},k_m)}, E_{(k_m,j)})$ from Ξ_i to Ξ_j in Seg, where $E_{(x,y)} \in E_C$ is an edge in the corresponding Seg, and $m \geq 1$ is the number of intermediate nodes between V_i and V_j, where $(V_i = \Xi_i)$ and $(V_j = \Xi_j)$.*

When the scopes are reachable (directly or indirectly), then there is a possibility that there exists a transformation that can transform the source constraint to the target constraint. We check this possibility by generating a group of intermediate constraints according to the defined conversion functions and operator mapping matrices along the edges of the path. Then checking the existence of a direct satisfiability between any of the generated intermediate constraints and the target constraint using the Proposition 1. When there exists at least one intermediate constraint that directly satisfies the target constraint, this means there exists a transformation that transforms the source constraint into the target constraint according to the captured semantics. Generation of these intermediate constraints depends on the reachability status between the scopes. Figure 4.4 depicts the realization process of constraints indirect satisfiability via scopes direct reachability.

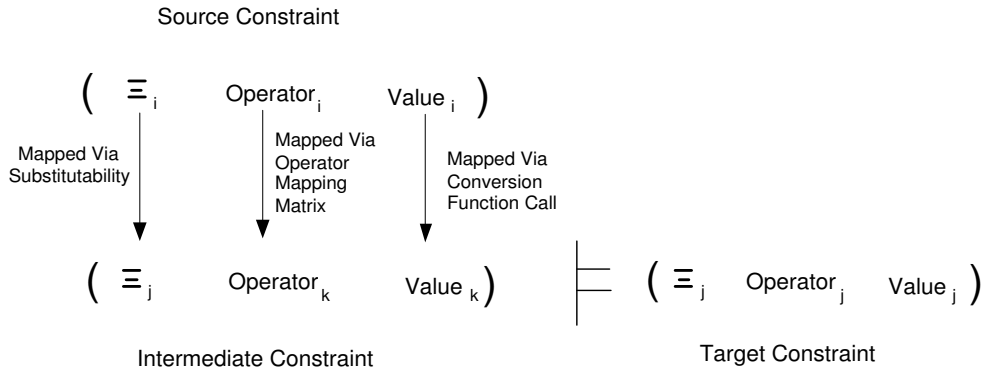

Figure 4.4: Constraint Indirect Satisfiability via Scopes Direct Reachability

Figure 4.4 indicates how an intermediate constraint is created for this case. An intermediate constraints is created in the following manner:

1. Its scope is as the scope of the target constraint,

2. Its value is a value resulting from invoking the corresponding conversion function with the value of the source constraint (note that the values returned by the conversion function are finite, according to Definition 9 (page 57)),

3. Its operator is an operator corresponding to the operator of the source constraint in the corresponding operator mapping matrix.

Theorem 3 shows how a transformation $\urcorner$ is formed when the scope of the source constraint is directly reachable to the scope of the target constraint. The theorem indicates that the input for the transformation is the source constraint, and the output is the target constraint, while the pre-constraints of the transformation are the substitution constraints attached to the edge from the source scope into the target scope. We used the notation ϑ_i to refer to a constraint's operator in general, as it could be $\in$, $\subseteq$, or λ. Similarly, we used the notation χ_i to refer to a constraint's value, as it could be a single value of a comparative constraint τ or it could be a single value belonging to the set of values of a bounding constraint η.

Theorem 3 (Transformation Formation via Scopes Direct Reachability) *Given two constraints $Cnst_i$, $Cnst_j$, a goal G. There exists a transformation $\urcorner$ such that $\urcorner(Cnst_i, \Pi_{(i,j)}) = Cnst_j$ if $(\Xi(Cnst_i) \rightharpoonup_G \Xi(Cnst_j))$ and $\exists\ Cnst_k$ such that $(Cnst_k \vDash Cnst_j)$, where $Cnst_k = (\ \Xi(Cnst_j)\ \vartheta_k\ \chi_k\ )$, $\vartheta_k \in \Omega_{(i,j)}(\vartheta_i)$, ϑ_i is the operator of $Cnst_i$, $\chi_k \in \Psi_{(i,j)}(\chi_i)$, and χ_i is the value of $Cnst_i$.*

Proof: *According to Definition 10, concepts substitutability graph indicates the substitutability status of application domain scopes. Every edge defined in the graph represents a mapping between two given scopes, hence the existence of an edge indicates the existence of a transformation from one of its scopes into the other scope (according to the edge direction). As the substitution constraints of a given edge must be satisfied in order to do the substitution process, hence such substitution constraints act as the pre-constraint of the transformation...(1). When the corresponding conversion function and operator mapping matrix*

lead to the generation of an intermediate constraint that can satisfy $Cnst_j$*, the transformation* $\urcorner$ *can be formed according to Definition 22 such that* $\urcorner(Cnst_i, \Pi_{(i,j)}) = Cnst_j$.

For example, the constraint (`Credit.Period` = 15) can be transformed into the constraint (`Payment.Type` ∈ {`Credit`,`Cash-Against-Doc`}) according to the concepts substitutability graph segment defined in Table 3.2, and the operator mapping matrix depicted in Figure 3.7. As one can notice, there exists an edge from `Credit.Period` into `Payment.Type` with a substitution constraint (`Credit.Period` ≥ 0). The conversion function restricts the values of `Payment.Type` to `Credit`, as (`Credit.Period` = 15). This forms an intermediate constraint (`Payment.Type` = `Credit`), which satisfies the constraint (`Payment.Type` ∈ {`Credit`,`Cash-Against-Doc`}). The substitution constraint (`Credit.Period` ≥ 0) will be the pre-constraint of the transformation, and the transformation will be defined as $\urcorner$(`Credit.Period` = 15, {`Credit.Period` ≥ 0})= `Payment.Type` ∈ {`Credit`,`Cash-Against-Doc`}. Algorithm 1 shows how Theorem 3 is applied to find a transformation that transforms the source constraint into a target constraint using CSG. The algorithm simply generate the possible intermediate constraints as indicated before, then check the direct satisfiability between the each intermediate constraint and the target constraint.

In case the source constraint is a bounding constraint, and the corresponding defined conversion function does not support the $(\in, \subseteq)$ operators but it supports the $(=)$ operator. We still can check the indirect satisfiability status, by invoking the conversion function with every value defined in the constraint. When the source constraint is a disjunction bounding constraint, having only one value that leads to formation of an intermediate constraint that satisfies the target constraint is enough to consider the existence of indirect satisfiability between the source constraint and the target constraint. However, when the source constraint is a conjunction bounding constraint, all of its values must lead to formation of intermediate constraints that satisfy the target constraint in order to consider the existence of indirect satisfiability between the source constraint and the target constraint. For example, the constraint (`Credit.Period` ∈ $\{0, 15\}$) is not supported by the OMM depicted in Figure 3.7, hence this case will be handled as if we have two equality comparative constraints (`Credit.Period` = 0, `Credit.Period` = 15). Each constraint will be examined individu-

ally, leading to the formation of these two intermediate constraints (`Payment.Type` = Cash-Against-Doc), (`Payment.Type` = Credit), respectively. Both of these intermediate constraints can satisfy the target constraint (`Payment.Type` ∈ {`Credit`,`Cash-Against-Doc`}), however having only one intermediate constraint satisfying the target constraint will be enough to accept the existence of indirect satisfiability between the source constraint and the target constraint.

Algorithm 1 Transformation Formation via Scopes Direct Reachability

Input: $Cnst_i$ (Source constraint), $Cnst_j$ (Target constraint), G (A goal).

Output: True if there exists a transformation that allows $Cnst_i$ to indirectly satisfy $Cnst_j$ via direct reachability with respect to G, False otherwise.

1: **Begin**
2: Z[ ]= $\Psi_{(i,j)}(\chi_i)$
3: O[ ]= $\Omega_{(i,j)}(\vartheta_i)$
4: **for each** χ, $\chi \in$ Z **do**
5: **for each** ϑ, $\vartheta \in$ O **do**
6: Create $Cnst_k$ as ($\Xi(Cnst_j)$ ϑ χ).
7: **if** ($Cnst_k \vDash Cnst_j$) **then**
8: Return True
9: **end if**
10: **end for**
11: **end for**
12: Return False
13: **End**

When there exists indirect reachability between the scopes, a sequence of transformations needs to be determined to decide the indirect satisfiability property. Each node in the path from the source's scope to the target's scope represents a scope of an intermediate constraint that needs to be indirectly satisfied by the predecessor constraint until the required target constraint is reached (see Figure 4.5). In other words, the transformation corresponding

to the indirect reachability case consists of a sequence of transformations formed via direct reachability between the intermediate scopes. As a conversion function could have multiple finite output values, a source constraint could be transformed into a finite number of intermediate constraints at a given stage. This forms a finite tree of the possible intermediate constraints that can be obtained from the source constraint using the defined finite conversion function. When one of the intermediate constraints of the final stage directly satisfies the target constraint this implies that the source constraint can indirectly satisfy the target constraints, as indicated in Figure 4.5.

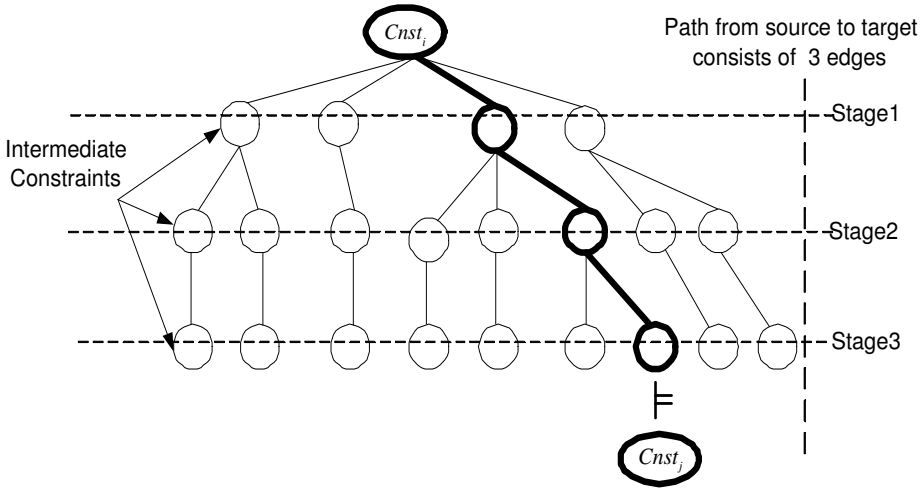

Figure 4.5: Indirect Satisfiability Decision Tree

The pre-constraints of the transformation $\neg$ corresponding to the scopes indirect reachability case are formed from the conjunction of the pre-constraints of the transformations forming $\neg$, as indicated in Theorem 4. Theorem 4 shows how a transformation $\neg$ is formed when the scope of the source constraint is indirectly reachable to the scope of the target constraint. The theorem indicates that the input for the transformation is the source constraint, and the output is the target constraint, while the pre-constraints of the transformation are the conjunction of the substitution constraints attached to the edges of the path from the source scope into the target scope. The theorem also indicates every intermediate constraints generated along the path must be satisfied by the preceding intermediate constraint in order to have such transformation.

Theorem 4 (Transformation Formation via Scopes Indirect Reachability) *Given two constraints $Cnst_i$, $Cnst_j$, a goal G. There exists a transformation $\urcorner$ such that $\urcorner(Cnst_i, \Pi_{(i,k_1)} \wedge \Pi_{(k_1,k_2)} \wedge ... \wedge \Pi_{(k_{m-1},k_m)} \wedge \Pi_{(k_m,j)}) = Cnst_j$ if $(\Xi(Cnst_i) \curvearrowright_G \Xi(Cnst_j))$ and $\exists$ $Cnst_{k_1}$, $Cnst_{k_2}$,...,$Cnst_{k_m}$ such that $(Cnst_{k_1} \vDash Cnst_{k_2}) \wedge (Cnst_{k_2} \vDash Cnst_{k_3}) \wedge .... \wedge (Cnst_{k_m} \vDash Cnst_j)$, where $Cnst_{k_1}$, $Cnst_{k_2}$,...,$Cnst_{k_m}$ are the intermediate constraints formed along the path from $\Xi(Cnst_i)$ to $\Xi(Cnst_j)$ using transformations formed via direct reachability between the intermediate scopes.*

Proof: *As the transformation from $Cnst_i$ to $Cnst_j$ consists of a sequence of transformations formed via direct reachability between the intermediate scopes using Theorem 3 and such transformations can be invoked only when their pre-constraints are satisfied, the conjunction of these pre-constraints must be satisfied in order to invoke $\urcorner$. Also, $(Cnst_{k_1} \vDash Cnst_{k_2}) \wedge (Cnst_{k_2} \vDash Cnst_{k_3}) \wedge .... \wedge (Cnst_{k_m} \vDash Cnst_j)$ must hold in order to reach $Cnst_j$, according to the transitive property of constraint satisfiability.*

For example, the constraint (`Credit.Period` = 15) can be transformed into the constraint (`Payment.Interest` = 0.17) via the constraint (`Payment.Type` = `Credit`). As we indicated before the constraint (`Credit.Period` = 15) can be transformed into (`Payment.Type` = `Credit`) when the pre-constraint (`Credit.Period` $\geq$ 0) is satisfied. Assuming, the constraint (`Payment.Type` = `Credit`) can be transformed into the constraint (`Payment.Interest` = 0.17) when the pre-constraint (`Payment.Type` = `Credit`) is satisfied. Hence, the pre-constraints of the transformation that transforms the constraint (`Credit.Period` = 15) into the constraint (`Payment.Interest` = 0.17) are the conjunction of (`Credit.Period` $\geq$ 0) and (`Payment.Type` = `Credit`). Algorithm 2 shows how Theorem 4 is applied to find a transformation that transforms the source constraint into a target constraint using CSG. Algorithm 2 starts by checking the reachability of the scopes such that if there no path from the source's scope to the target's scope, this implies the source constraint cannot satisfy the target constraint.

In case there exists a path from the source's scope to the target's scope, it processes the first following node to the source's scope by generating all the intermediate constraints corresponding to the output value of the conversion function and the operator mapping

Algorithm 2 Transformation Formation via Scopes Indirect Reachability

Input: $Cnst_i$ (Source constraint), $Cnst_j$ (Target constraint), G (A goal).

Output: True if there exists a transformation that allows $Cnst_i$ to indirectly satisfy $Cnst_j$ via indirect reachability with respect to G, False otherwise.

1: **Begin**
2: **if** (There is no path from $\Xi(Cnst_i)$ to $\Xi(Cnst_j)$ in the segment of G) **then**
3: Return False
4: **else**
5: Ξ_k = GetFollowingNode($\Xi(Cnst_i)$)
6: **if** ($\Xi_k = \Xi(Cnst_j)$) **then**
7: Apply Algorithm 1 for ($Cnst_i$,$Cnst_j$, G)
8: **else**
9: Z[ ]= $\Psi_{(i,k)}(\chi_i)$
10: O[ ]= $\Omega_{(i,k)}(\vartheta_i)$
11: **for each** χ, $\chi \in$ Z **do**
12: **for each** ϑ, $\vartheta \in$ O **do**
13: Create $Cnst_k$ as (Ξ_k ϑ χ).
14: Apply Algorithm 2 for ($Cnst_k$,$Cnst_j$, G)
15: **end for**
16: **end for**
17: **end if**
18: Return False
19: **end if**
20: **End**

matrix, then makes a recursive call for every intermediate constraint as an intermediate source. This process is continued until the target scope is reached (that is only one edge is between an intermediate source and the target). When one of the final intermediate constraints directly satisfies the target constraint, this implies that the source constraint indirectly satisfies the target constraint.

Having a transformation that can transform a source constraint into a target constraint is not enough to determine the existence of indirect satisfiability between the constraints, as the pre-constraints of the transformation need to be satisfied at the time of transformation invocation (that is context-based satisfiability). Therefore, when the persisting constraints at the time of invocation satisfies the pre-constraints of the transformation, the transformation known as *valid* and can be invoked. When the pre-constraints of the transformation cannot be satisfied, this means the source constraint cannot indirectly satisfy the target constraint with respect to the captured semantics of the involved goal. Therefore, a transformation could be valid with respect to a given goal and the same transformation could be invalid with respect to the semantics of another goal. Constraint indirect satisfiability is determined according to Theorem 5. The theorem shows when a transformation is considered valid. That when the conjunction of the persisting constraints at the invocation time and the generated intermediate constraints satisfies the pre-constraints of the transformation.

Theorem 5 (Constraint Indirect Satisfiability) *Given two constraints $Cnst_i$, $Cnst_j$, and a goal G such that $(Cnst_i \in f_i)$, $(Cnst_j \in f_j)$, and $(\Xi(Cnst_i) \curvearrowright_G \Xi(Cnst_j))$. $Cnst_i$ indirectly satisfies $Cnst_j$ with respect to G via a transformation $\urcorner$ (denoted as $Cnst_i \gg_G Cnst_j$) if $(f_i \wedge f_j \wedge Cnst_{k_1} \wedge ... \wedge Cnst_{k_m}) \vDash (\Pi_{(i,k_1)} \wedge \Pi_{(k_1,k_2)} \wedge ... \wedge \Pi_{(k_{m-1},k_m)} \wedge \Pi_{(k_m,j)})$, where $\urcorner(Cnst_i, \Pi_{(i,k_1)} \wedge \Pi_{(k_1,k_2)} \wedge ... \wedge \Pi_{(k_{m-1},k_m)} \wedge \Pi_{(k_m,j)}) = Cnst_j$, and $Cnst_{k_1}$,...,$Cnst_{k_m}$ are the corresponding intermediate constraints.*

Proof: *As $(\Xi(Cnst_i) \curvearrowright_G \Xi(Cnst_j))$, $\urcorner$ will be created according to Theorem 4. According to Definition 23, the transformation pre-constraints must be satisfied in order to be able to apply the transformation. The only way to satisfy these pre-constraints at invocation time is when they are satisfied by the existing constraints at invocation time, which are f_i, f_j, and the involved intermediate constraints.*

Theorem 5 requires the scopes of the constraints to be indirectly reachable. However, determining the validity of a transformation when the scopes are directly reachable is a special case of Theorem 5, that when only one intermediate constraint is used.

In what follows we will show how the proposed constraint indirect satisfiability approach is adopted during constraint matching. As substitutability is the adopted matching rule, we propose the concept of *constraint substitutability* for matching constraints. In which, a source constraint matches a target constraint when the target constraint can be substituted by the source constraint, as indicated in Theorem 6. The theorem simply matches a source constraint to a target constraint when the source constraint directly satisfies the target constraint (if they have the same scope) or when it indirectly satisfies the target constraint (if they have different scopes).

Theorem 6 (Constraint Substitutability) *Given two constraints $Cnst_i$, $Cnst_j$ and a goal G. $Cnst_j$ can be substituted by $Cnst_i$ with respect to G (denoted as $Cnst_i \trianglerighteq_G Cnst_j$) if $(Cnst_i \vDash Cnst_j) \vee (Cnst_i \gg_G Cnst_j)$.*

Proof: *A target constraint can be substituted by a source constraint, when the source constraint satisfies the target constraint. When the source and target constraints have the same scope, Proposition 1 is used to determine the satisfiability status. When the source and the target have different but reachable scopes, Theorem 5 is used to determine the satisfiability status.*

For example, the constraint (`Credit.Period` = 15) substitutes the constraint (`Credit.Period` > 0) as it directly satisfies it. Also the constraint (`Credit.Period` = 15) substitutes the constraint (`Payment.Type` = `Credit`) as it indirectly satisfies it, as we indicated before. Now, we will extend Proposition 2 for matching sets of constraints as it is based on constraints direct satisfiability, which leads to the appearance of false negatives as indicated before. To avoid this problem, we match constraints-sets using constraint substitutability instead of constraint direct satisfiability, as indicated in Proposition 3.

Proposition 3 (Constraint-Set Substitutability) *Given two sets of constraints f_i, f_j, and a goal G. f_j can be substituted by f_i with respect to G (denoted as $f_i \trianglerighteq_G f_j$) if $\forall\ Cnst_u$*

$\in f_j$, $\exists\ Cnst_v \in f_i$ *such that* ($Cnst_v \trianglerighteq_G Cnst_u$).

Proof: *As a given set of constraints is satisfied when each of its constraints is individually satisfied (according to Assumption 1),* f_j *can be satisfied when each constraint of its elements has a corresponding substituting constraint in* f_i.

For example, the set of the describing-constraints of the user G^+ model depicted in Figure 3.10 can be substituted by the set of the describing-constraints of the service G^+ model depicted in Figure 3.11, as the constraint (`Specialty.Type` $\subseteq$ {Motor-vehicles, Dangerous-Cargo}) directly satisfies the constraint (`Specialty.Type` = Motor-vehicles), and the constraint (`Credit.Period` = 15) indirectly satisfies the constraint (`Payment.Type` = Credit). Proposition 3 does not prevent a given source constraint from substituting multiple target constraints individually. Hence, to know which source constraints substitute which target constraints, a *Constraints Mapping Matrix(CMM)* is built. CMM's rows are the constraints of the source set, CMM's columns are the constraints of the target set, and the elements of CMM are either 1 (indicating that the target constraint can be substituted by the source constraint) or 0 (indicating the target constraint cannot be substituted by the corresponding source constraint).

Algorithm 3 indicates how the substitutability status between two sets of constraints is determined according to the constructed CMM. For example, the CMM corresponding to the set of the describing-constraints of the user G^+ model depicted in Figure 3.10 and the set of the describing-constraints of the service G^+ model depicted in Figure 3.11 is as follows.

	`Specialty.Type` = Motor-vehicles	`Payment.Type` = Credit
`Specialty.Type` $\subseteq$ {Motor-vehicles, Dangerous-cargo}	1	0
Credit.Period = 15	0	1

A column of zeros indicates that the corresponding target constraints cannot be substituted by any constraint in the source set. A column with multiple ones indicates the corresponding target constraint is substituted by the conjunction of the corresponding source constraints, while a row with multiple ones implies the corresponding source constraint can substitute the conjunction of the corresponding target constraints. For example, the CMM

corresponding to the set of the post-constraints of the user G^+ model depicted in Figure 3.10 and the set of the post-constraints of the service G^+ model depicted in Figure 3.11 is as follows.

	(Cargo.Status = Accomplished)
(ShipmentOrder.Status = Fulfilled)	1
(Payment.Status = Received)	1

Algorithm 3 Constraint-Set Substitutability

Input: f_i (A source set of constraints), f_j (a target set of constraints), and G (A goal).

Output: True when f_j can be substituted by f_i with respect to G, and False otherwise.

1: **Begin**
2: **for each** $Cnst_{(j,y)}$, $Cnst_{(j,y)} \in f_j$ **do**
3: **for each** $Cnst_{(i,x)}$, $Cnst_{(i,x)} \in f_i$ **do**
4: **if** $(Cnst_{(i,x)} \trianglerighteq_G Cnst_{(j,y)})$ **then**
5: $CMM[x, y] = 1$
6: **else**
7: $CMM[x, y] = 0$
8: **end if**
9: **end for**
10: **end for**
11: **if** (CMM has a column of zeros) **then**
12: Return (False)
13: **else**
14: Return (True)
15: **end if**
16: **End**

During the matching process a different CMM is constructed for comparing the pre-constraints, the post-constraints, the describing-constraints, and for every set of persisting

constraints in order to determine the matching status between a service functionality and a user's required functionality, more details about the matching process are given in Chapter 5.

4.2.3 Goal Achievement as a Correctness Criterion

The achievement of users' goals via the returned matching results indicates the correctness of these matching results. Hence, the matchmaker needs to be able to determine when a user goal is achieved such that it can decide whether a given service can achieve the required goal or not. According to the G^+ model, a user goal is considered achieved when the pre-constraints of HLFC is transformed into the post-constraints of HLFC via the required behavior model, and the describing-constraints of HLFC are satisfied. We define the achievement of a given goal as follows.

Definition 26 (Goal Achievement) *Given a goal G described via a goal achievement pattern* $Gp_i = \langle\ Op,\ GCtxt,\ OpSeq\ \rangle$, *G is achieved when* T_i *and* D_i *are satisfied provided that* $\sigma_i(S_i) = T_i$, *where* $S_i = GCtxt^{Pre}$, $T_i = GCtxt^{Post}$, $D_i = GCtxt^{Desc}$, σ_i *is the transformation function corresponding to* β_i. *This is denoted as* $(G \vdash \langle S_i, T_i, \beta_i, D_i \rangle)$, *where* $\vdash$ *is the goal achievement operator.*

According to Definition 26, the matchmaker needs to find services with behavior models that can transform the required pre-constraints into the required post-constraints substituting the required behavior model, also these services must satisfy the required describing-constraints. As the goal achievement is the correctness criterion adopted in the matching process, the matching process is based on the properties of the goal achievement operator that define its semantics. The goal achievement operator has the following properties.

1- Pre-constraints Substitutability Invariance: This is denoted as

$$(S_j \trianglerighteq_{G_i} S_i) \wedge (G_i \vdash \langle S_i, T_i, \beta_i, D_i \rangle) \Rightarrow (G_i \vdash \langle S_j, T_i, \beta_i, D_i \rangle)$$

This property indicates G_i is still considered to be achieved when its pre-constraints are substituted by another subsuming set of pre-constraints.

Proof: As $(S_j \trianglerighteq_G S_i)$, this implies correctness of the GAP is still guaranteed to be satisfied according to Definition 14 when S_i is replaced by S_j. Hence, σ_i is guaranteed

to be invoked by S_j such that $\sigma_i(S_j) = T_i$. Therefore, the goal can be achieved via $(G_i \vdash \langle S_j, T_i, \beta_i, D_i \rangle)$.

2- Serial Composition: This is denoted as

$(G_i \vdash \langle S_i, T_i, \beta_i, D_i \rangle) \wedge (G_j \vdash \langle T_i, T_j, \beta_j, D_j \rangle) \Rightarrow (G_k \vdash \langle S_i, T_j, \beta_k, D_i \cup D_j \rangle)$, where D_i and D_j are independent.

This property indicates a composite goal is guaranteed to be achieved when its subgoals are achieved, provided that the goals have independent describing-constraints. Two sets of constraints are considered independent when they have unreachable scopes.

Proof: As $(\sigma_i(S_i) = T_i)$, and $(\sigma_j(T_i) = T_j)$, this implies that $\sigma_j(\sigma_i(S_i)) = T_j$. Hence, a behavior model β_k (that is formed form a sequential composition of β_i and β_j) that can transform S_i into T_j is guaranteed to exist. Hence, a new composite goal G_k (that is formed form a sequential composition of G_i and G_j) is guaranteed to be achieved via a correct GAP, as the correctness constraints indicated in Definition 14 hold (the output of σ_i satisfies the input of σ_j). As D_i and D_j are independent and guaranteed to be satisfied, the composite describing-constraints is formed as $D_i \cup D_j$.

3- Transitivity: This is denoted as

$(G_i \vdash \langle S_i, T_i, \beta_i, D_i \rangle) \wedge (G_i \vdash \langle T_i, T_j, \beta_j, D_i \rangle) \Rightarrow (G_i \vdash \langle S_i, T_j, \beta_k, D_i \rangle)$

This property indicates when a goal is achieved via two different achievement processes, the same goal is considered to be achieved via a third achievement process that is a sequential composition of these two processes when the post-constraints of one achievement process is the pre-constraints of the other process, providing that the two process have the same describing-constraints.

Proof: Transitivity is a special case of the goal serial composition property. Hence, by applying Property (2), we reach that $(G_i \vdash \langle S_i, T_j, \beta_k, D_i \rangle)$.

4- Irreflexive: This is denoted as

$\nexists\ (G_i \vdash \langle S_i, S_i, \beta_i, D_i \rangle)$

This property indicates that in a goal achievement process, the pre-constraints must

not equal to the post-constraint.

Proof: A service invocation whether it is an information service or a transaction service should lead to a change in the facts appeared before and after the invocation. Otherwise, this means such service is doing nothing, which is not practical.

5- Asymmetric: This is denoted as

$(G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle) \nRightarrow (G_i \vdash \langle T_i, S_i, \beta_j, D_i\rangle)$

This property indicates that a given goal cannot be achieved by transforming its post-constraints into its pre-constraints.

Proof: Assuming both $(G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle)$ and $(G_i \vdash \langle T_i, S_i, \beta_j, D_i\rangle)$ hold...(1). This implies that $\sigma_i(S_i) = T_i$, and $\sigma_j(T_i) = S_i$ hold, hence $\sigma_i(\sigma_j(T_i)) = T_i$ could be reached by applying the transitivity property. This implies that $(G_i \vdash \langle T_i, T_i, \beta_j, D_i\rangle)$, which indicates that G_1 is a reflexive goal, which contradicts Property (4)...(2). Hence, assumption in (1) is not valid.

These properties guide how the matching process will be performed. For example, the composition property shows that a user request could be answered via invoking a sequence of services. The pre-constraints substitutability invariance property shows that the pre-constraints of a user and the pre-constraints of a service can be different, and the service still considered as a prospective matching result when its pre-constraints are substituted by the user's pre-constraints. The asymmetric property confirms that the pre-constraints and the post-constraints must be examined separately. Table 4.3 summarizes the properties of the goal achievement operator.

Property	Notation
1- **Pre-constraints Substitutability Invariance**	$(S_j \trianglerighteq_{G_i} S_i) \wedge (G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle) \Rightarrow (G_i \vdash \langle S_j, T_i, \beta_i, D_i\rangle)$
2-**Serial Composition**	$(G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle) \wedge (G_j \vdash \langle T_i, T_j, \beta_j, D_j\rangle) \Rightarrow (G_k \vdash \langle S_i, T_j, \beta_k, D_i \cup D_j\rangle)$
3- **Transitivity**	$(G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle) \wedge (G_i \vdash \langle T_i, T_j, \beta_j, D_i\rangle) \Rightarrow (G_i \vdash \langle S_i, T_j, \beta_k, D_i\rangle)$
4- **Irreflexive**	$\nexists\ (G_i \vdash \langle S_i, S_i, \beta_i, D_i\rangle)$
5- **Asymmetric**	$(G_i \vdash \langle S_i, T_i, \beta_i, D_i\rangle) \nRightarrow (G_i \vdash \langle T_i, S_i, \beta_j, D_i\rangle)$

Table 4.3: Goal Achievement Operator Properties

However, there are other properties that do not hold for the goal achievement operator. For example, the goal achievement operator is not distributive over conjunctive sources.

In other words, $(G_i \vdash \langle (S_i \wedge S_j), T_i, \beta_i, D_i \rangle) \not\Rightarrow (G_i \vdash \langle S_i, T_i, \beta_i, D_i \rangle) \wedge (G_i \vdash \langle S_j, T_i, \beta_i, D_i \rangle)$, this is because $(S_i \wedge S_j)$ is required to be satisfied in order to have a correct GAP, which cannot be obtained by only S_i or S_j. Also the goal achievement operator is not post-constraints substitutability invariant nor describing-constraints substitutability invariant, as having different post-constraints or describing-constraints changes the semantics of the goal achievement. For example, when a goal G_i is considered to be achieved by transforming $(X.a > 10)$ into $(Y.b > 20)$, this is not the same as a goal G_j that is considered to be achieved by transforming $(X.a > 10)$ into $(Y.b > 10)$. Because having value like $Y.b = 15$ does not lead to the achievement of G_i, but leads to the achievement of G_j.

4.3 Behavior Models Matching using FSMS

This section indicates how behavior models are matched according to FSMS. As only linear behavior models are adopted, the proposed behavior model matching is based on their state sequence matching, which in turn is based on matching their corresponding states. Matching state sequences could be determined either by using one-to-one state matching approach or by using m-to-n state matching approach. In the one-to-one state matching approach, a source state is examined against only one target state at a time. In the m-to-n state matching approach, a group of consecutive source states is examined against a group of consecutive target states, and each group could have a different length (that is the number of states in the group).

Existing approaches such as Magee et al. [1997], Pierce and Sangiorgi [2000], Findler et al. [2001], and Berardi [2005] adopt the one-to-one state matching approach. They require the state sequences to have the same length in order to be matched. Furthermore, they match the states syntactically, as they assume a given goal can only be achieved by invoking the same sequence of operations. Indeed, such an approach is a very strict approach that leads to the appearance of the false negatives, as a goal could be achieved via different sequences of operations that vary in their length. Therefore, we argue that states should be matched on a semantic basis and state sequences should be matched adopting an m-to-n state matching approach in order to minimize the appearance of false negatives. As none of the currently

existing behavior model matching techniques adopts an m-to-n state matching approach, in this section we indicate how states will be matched on a semantic basis, then indicate how behavior models will be matched using an m-to-n state matching approach, adopting FSMS.

4.3.1 State Matching

As substitutability is the adopted matching rule, a source state S_x matches a target state S_t when S_x substitutes S_t with respect to the involved goal. However, as indicated in Definition 19, a behavior state consists of two subsets of constraints: effective and idle. We argue that state matching should be based only on the effective constraints, as the idle constraints do not have any effect on the sequence dynamics at the corresponding transition point. However, this does not mean that the idle constraints are completely ignored, as they will be taken into consideration when they become effective at other successive transition points. During state matching, we will take into consideration the effect of state substitution only at the corresponding transition point. The overall effect of state substitution will be taken into consideration when the whole behavior model is examined. More details will be given later in this section. We match behavior states according to the substitutability of their effective constraints, as indicated in Proposition 4.

Proposition 4 (State Matching) *Given a source state $S_x = \langle f_x^e, f_x^i \rangle$, a target state $S_t = \langle f_t^e, f_t^i \rangle$, and a given goal G such that $S_x \in \beta_i$, and $S_t \in \beta_j$. A source state S_x matches a target state S_t with respect to G (denoted as $S_x \trianglerighteq_G S_t$) if $(f_x^e \trianglerighteq_G f_t^e)$.*

Proof: *As we are dealing with correct GAPs, $f_t^e \vDash Op_{t+1}^{Pre}$ must hold in order to be able to invoke Op_{t+1}, according to Definition 14 and Definition 17 ...(1). When S_t is substituted by S_x, Op_{t+1}^{Pre} must be guaranteed to be satisfied after the substitution in order to be able to invoke Op_{t+1}, hence $f_x^e \trianglerighteq_G Op_{t+1}^{Pre}$ must hold...(2). This can be achieved when $(f_x^e \trianglerighteq_G f_t^e)$, as $((f_x^e \trianglerighteq_G f_t^e) \wedge (f_t^e \vDash Op_{t+1}^{Pre}) \Rightarrow f_x^e \trianglerighteq_G Op_{t+1}^{Pre}$ (according to the transitive property of constraint satisfiability).*

For example, according to the substitutability graph segment defined in Table 3.2, the state ⟨{`Frieght.Details` ≠ NULL, `Origin.Details` ≠ NULL, `Destination.Details` ≠ NULL, `IncoTerm.Type` ∈ {FOB,EXW,CIF}}, {}⟩ (a target state) can be substituted by the state ⟨ { `Cargo.Details` = `1000 Cars`, `Cargo.POL` = `Melbourne-Australia`, `Cargo.POD` = `Alexandria-Egypt`, `IncoTerm.Type` = FOB}, {`Cargo.Course` = `Port-to-Port`}⟩ (a source state), as the constraints (`Cargo.Details` = `1000 Cars`), (`Cargo.POL` = `Melbourne-Australia`), (`Cargo.POD` = `Alexandria-Egypt`), and (`IncoTerm.Type` = FOB) substitute the constraints (`Frieght.Details` ≠ NULL), (`Origin. Details` ≠ NULL), (`Destination.Details` ≠ NULL), and (`IncoTerm.Type` ∈ {FOB,EXW,CIF}) respectively.

During the matching process, the matchmaker needs to identify a given state to be able to determine whether this state is reached or not. Hence, we need a state identification technique to recognize states. We define the concept of *state scope* in order to identify a given state during the matching process. As Proposition 4 indicates that the effective constraints of a state are the only constraints to be used in the matching process, we use the scopes of these effective constraints to identify the state, a state scope is defined as follows.

Definition 27 (State Scope) *Given a state* $S_x = \langle f_x^e, f_x^i \rangle$. *The scope of* S_x *(denoted as* $\Xi(S_x)$*) is* $\Xi(f_x^e)$.

The scope of the state ⟨{`Cargo.Details` = 1000 Cars, `Cargo.POL` = Melbourne-Australia, `Cargo.POD` = Alexandria-Egypt, `IncoTerm.Type` = FOB}, {`Cargo.Course` = Port-to-Port}⟩ is the set { `Cargo.Details`, `Cargo.POL`, `Cargo.POD`, `IncoTerm.Type`}. The matching status between two states could be determined according to their scopes. That when two states have unreachable scopes with respect to the involved goal this implies that the states cannot be matched. However, if they are reachable, this implies the substitutability status between their effective constraints should be determined to get the final matching status.

We will adopt Proposition 4 for state matching during behavior models matching. First we will propose a behavior matching approach using one-to-one state matching. Later in this chapter, we will propose a behavior matching approach using m-to-n state matching. Theorem 7 shows how behavior models will be matched using one-to-one state matching. It

simply matches two behavior models when they have the same number of states and when every state in the source behavior model substitutes the corresponding target state in the target behavior model. We identify this approach for behavior matching as a one-to-one semantic behavior matching approach, as states will be semantically matched according to the substitutability of their effective constraints. A comparison between the proposed one-to-one semantic behavior matching against one-to-one syntactic behavior matching approach will be given at the end of Chapter 5.

Theorem 7 (One-to-One Semantic Behavior Matching) *Given two behavior models* $\beta_x = \langle S_{(x,0)}, S_{(x,1)}, ..., S_{(x,n)} \rangle$, $\beta_t = \langle S_{(t,0)}, S_{(t,1)}, ..., S_{(t,m)} \rangle$, *and a given goal G,* β_x *matches* β_t *with respect to G using one-to-one state matching approach (denoted as* $\beta_x \sim_G \beta_t$*) if* $(m = n) \wedge (\forall_{i=0}^{i=n}\ S_{(x,i)} \trianglerighteq_G S_{(t,i)})$.

Proof: *As we are dealing with correct GAPs, at every transition point the effective constraints must satisfy the pre-constraints of its following operation. Hence,* β_t *implies a sequence of effective constraints* $\langle f^e_{(t,0)}, f^e_{(t,1)}, ..., f^e_{(t,m)} \rangle$ *that must hold in order to transform* $f_{(t,0)}$ *into* $f_{(t,m)}$*. A source behavior model can substitute a target behavior model when the source behavior model implies a sequence of effective constraints that can substitute the required sequence of effective constraints, as indicated in Figure 4.6. This can be achieved when the two behavior models have the same number of transitions and every target state is matched to the corresponding source state according to Proposition 4.*

In the remainder of this chapter, for simplicity we will refer to constraints by alphabet letters such that two constraints will be considered substitutable when they are represented by the same alphabet letter. For example, the state S_x=⟨{a,b}, {c,d}⟩ has a, b as its effective constraints and c, d as its idle constraints. Also S_x matches a state S_t, where S_t =⟨{a,b}, {g,h}⟩. In what follows we will discuss the m-to-n behavior matching approach.

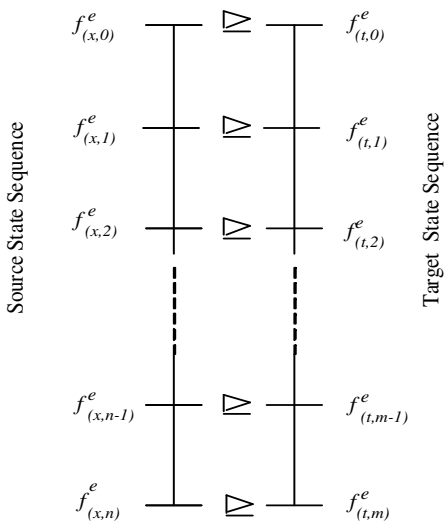

Figure 4.6: One-to-One State Sequence Matching

4.3.2 State Expansion

In the m-to-n approach, when a source state S_x cannot match a target state S_t with respect a given goal G (denoted as $S_x \not\trianglerighteq_G S_t$), this case is examined further for any possibility for a match case that could result from merging S_x and S_t with other states in their corresponding state sequences. When states are merged, different effective constraints would appear in the resulting states, which gives the opportunity for a match case to happen. When a state is merged with other states in its sequence, we define this case as a *state expansion*. First we will define what is meant by state expansion and introduce the different types of state expansions, as they represent the actions to be done during m-to-n state matching. Second, we will discuss what is meant by state merge and provide the theorems and propositions that show how states are expanded. Finally, we will show how state sequences of behavior models will be re-clustered using these different types of state expansions in order to be matched. First, a state expansion is defined as follows.

Definition 28 (State Expansion) *Given a state S_i such that $S_i \in \beta_x$, S_i is expanded to the state S'_i when it is merged with other adjacent states in β_x forming a cluster of states.*

As both the source and the target state sequences are involved in the matching process, both *source state expansion* and *target state expansion* could be required during the matching process. In the source state expansion, a source state will be expanded in order to match a target state. In the target state expansion, a target state will be expanded in order to be matched to a source state. When a state is expanded, it could be merged with either its successor states (which is known as *Down Expansion*) or its predecessor states (which is known as *Reverse Expansion*), as indicated in Figure 4.7.

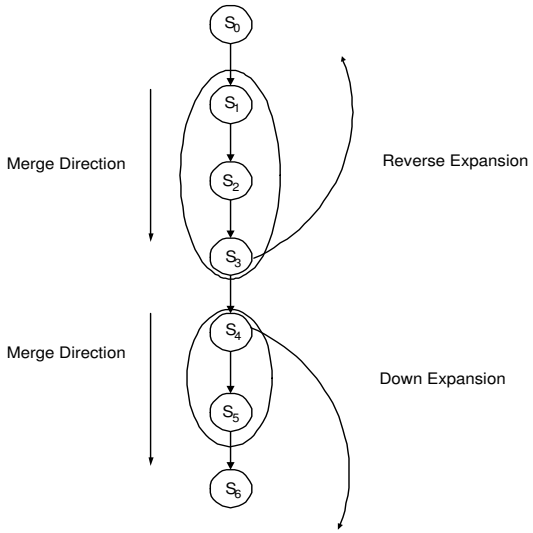

Figure 4.7: Expansion Direction versus Merge Direction

However, the merge direction is always from the lower index state to the higher index state regardless of the expansion direction. Figure 4.7 indicates that S_4 is down expanded by being merged with S_5, and S_3 is reverse expanded by merging the states S_1, S_2, and S_3, this is done by merging S_1 with S_2 and the results will be merged with S_3. There are four types of state expansion could be adopted during the state matching process, as indicated in Definition 29. More details about which type of state expansion will be applied and when during the matching process will be given in Section 4.3.3.

Definition 29 (State Expansion Types) *Given a source state S_x, a target state S_t, and a goal G. There are four types of state expansions could be adopted during the matching process between S_x and S_t with respect to G. These types are:*

Source Down Expansion *When $S_x \not\trianglerighteq_G S_t$, S_x is merged with some of its successors states in order to match S_t.*

Source Reverse Expansion *When $S_x \not\trianglerighteq_G S_t$, S_x is merged with some of its predecessor states in order to match S_t.*

Target Down Expansion *When $S_x \trianglerighteq_G S_t$, S_t is merged with some of its successors states until cannot be matched with S_x.*

Target Reverse Expansion *When $S_x \not\trianglerighteq_G S_t$, S_t is merged with some of its predecessor states in order to be matched.*

Now, we discuss what is meant by state merge, and prove the correctness of the proposed merge approach. In our merge approach, merging two consecutive states S_x and S_{x+1}, forming a new state S_m, means that their successor operations Op_{x+1} and Op_{x+2} are merged forming a new operation Op_m, as indicated in Figure 4.8.

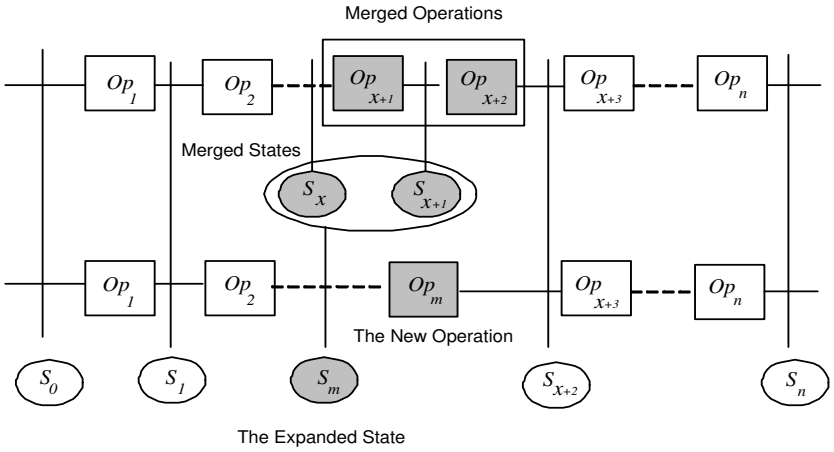

Figure 4.8: Consecutive States Merge

S_m is computed as if there is a new operation Op_m in the sequence replacing the operations Op_{x+1} and Op_{x+2}. Op_m is formed by cascading Op_{x+1} and Op_{x+2}, as depicted in Figure 4.9.

Figure 4.8 indicates that the input of Op_m is the union between the sets of concepts A and B, its output is the union between the sets of concepts C and E, while the set of concepts D

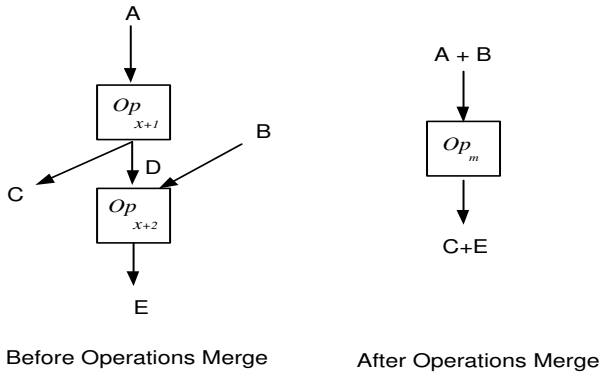

Figure 4.9: Consecutive Operations Merge

will not appear neither in Op_m input nor in Op_m output. As a result of that, any constraint over D is discarded when the expanded state is computed. This discarded constraints are known as the *vanishing constraints*. By discarding these vanishing constraints, the resulting expanded state will have different effective constraints, which creates an opportunity for a match case to happen. The pre-constraints of Op_m will be the constraints based on the concepts of A and B, and the post-constraints will be the constraints based on the concepts of C and E, as indicated in Proposition 5.

Proposition 5 (Operations Merge Correctness) *Given two consecutive operations Op_{x+1}, Op_{x+2} in an operations sequence of a given GAP. Op_m is a correct merge between Op_{x+1} and Op_{x+2} if $(Op_m^{Pre} = Op_{x+1}^{Pre} + (Op_{x+2}^{Pre} \diamond Op_{x+1}^{Post})) \wedge (Op_m^{Post} = Op_{x+2}^{Post} + (Op_{x+1}^{Post} \diamond Op_{x+2}^{Pre}))$.*

Proof: *To have a correct merge between Op_{x+1} and Op_{x+2}, f_{x+2} (the persisting constraints are the transition point x+2) should not be changed after Op_m execution. Therefore, we will compute the persisting constraints at the transition point x+2 before and after the merge using Theorem 2, and we will show that f_{x+2} has not changed due to the merge when Op_m^{Pre} and Op_m^{Post} are computed as indicated.*

Before Merge:

$f_{x+2} = f_{x+1}^i + Op_{x+2}^{Post}$...(1). $f_{x+1}^i = (f_{x+1} \diamond Op_{x+2}^{Pre})$...(2). $f_{x+1} = f_x^i + Op_{x+1}^{Post}$....(3). $f_x^i = (f_x \diamond Op_{x+1}^{Pre})$...(4). From (1),(2),(3),(4), we can write f_{x+2} as a function of f_x, Op_{x+1}^{Pre}, Op_{x+1}^{Post}, Op_{x+2}^{Pre}, and Op_{x+2}^{Post}. Hence, $f_{x+2} = (((f_x \diamond Op_{x+1}^{Pre}) + Op_{x+1}^{Post}) \diamond Op_{x+2}^{Pre})$

+ Op^{Post}_{x+2}....(5), which is represented by the shaded area in the Venn diagram depicted in Figure 4.10, taking into consideration that the semantic difference is used instead of the set-difference.

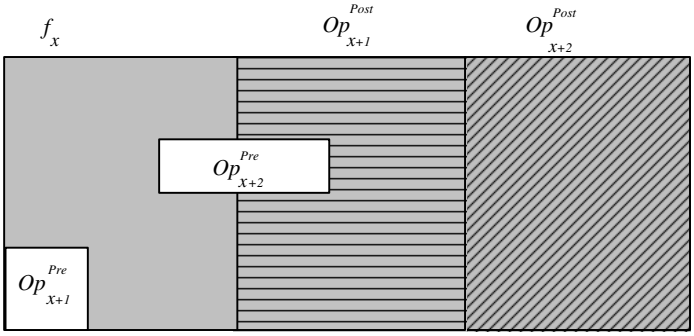

Figure 4.10: f_{x+2} Elements

After Merge:

$f_{x+2} = f^i_x + Op^{Post}_m$...(6). $f^i_x = f_x \diamond Op^{Pre}_m$.....(7). From (6) and (7), we reach $f_{x+2} = (f_x \diamond Op^{Pre}_m) + Op^{Post}_m$.....(8). By substituting the values of $Op^{Pre}_m = Op^{Pre}_{x+1} + (Op^{Pre}_{x+2} \diamond Op^{Post}_{x+1})$, and $Op^{Post}_m = Op^{Post}_{x+2} + (Op^{Post}_{x+1} \diamond Op^{Pre}_{x+2})$ in (8). We reach to $f_{x+2} = (f_x \diamond (Op^{Pre}_{x+1} + (Op^{Pre}_{x+2} \diamond Op^{Post}_{x+1}))) + (Op^{Post}_{x+2} + (Op^{Post}_{x+1} \diamond Op^{Pre}_{x+2}))$.....(9). By rearranging the terms, we reach $f_{x+2} = (f_x \diamond (Op^{Pre}_{x+1} + (Op^{Pre}_{x+2} \diamond Op^{Post}_{x+1})))+ (Op^{Post}_{x+1} \diamond Op^{Pre}_{x+2})) + Op^{Post}_{x+2}$. Which also represent the same shaded area in the Venn diagram depicted in Figure 4.10. Therefore, Equation (9) equals to Equation (5), which indicates that the assigned values for Op^{Pre}_m and Op^{Post}_m does not change f_{x+2}.

Having Op_m formed this way guarantees the states appeared before and after the merge are not changed. Figure 4.11 shows the effect of merging Op_2 and Op_3 by depicting an operation sequence before and after the merge. The following table indicates the state sequence before the merge.

State	Persisting Constraints	Effective Constraints	Idle Constraints
S_0	$\{a, d, z\}$	$\{a\}$	$\{d, z\}$
S_1	$\{b, c, d, z\}$	$\{c, d\}$	$\{b, z\}$
S_2	$\{b, e, z\}$	$\{e, z\}$	$\{b\}$
S_3	$\{b, g\}$	$\{b, g\}$	$\{\}$
S_4	$\{h\}$	$\{h\}$	$\{\}$

The following table indicates the state sequence after the merge.

State	Persisting Constraints	Effective Constraints	Idle Constraints
$S_0^{'}$	$\{a, d, z\}$	$\{a\}$	$\{d, z\}$
$S_1^{'}$	$\{b, c, d, z\}$	$\{c, d, z\}$	$\{b\}$
$S_2^{'}$	$\{b, g\}$	$\{b, g\}$	$\{\}$
$S_3^{'}$	$\{h\}$	$\{h\}$	$\{\}$

As we can see S_0, S_3, S_4 are not affected by the merge, as the pre-constraints of Op_m are $\{c, d, z\}$ and post-constraint of Op_m is $\{g\}$. The effective constraints of S_m do not contain the constraint e (that is a vanishing constraint). Additionally, the expanded state version of S_1 (that is $S_1^{'}$) includes more number of effective constraints than S_1.

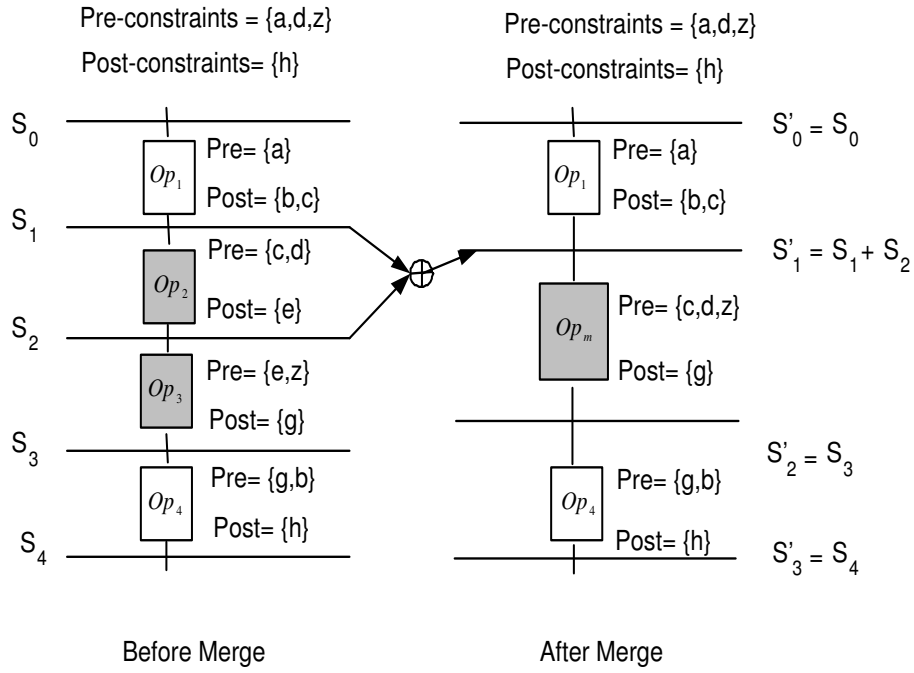

Figure 4.11: State Merge Example

Based on the proposed operations merge approach, two consecutive states are merged as indicated in Theorem 8. Theorem 8 indicates that f_{x+1}^{i} is not involved in the state merge operation as it does not affect the invocation of Op_m, however f_{x+1}^{i} still appears in the state S_{x+2}. If there exists no successor operation for S_{x+1}, this implies that S_x and S_{x+1} cannot be merged. This case happens when the operation sequence consists of only one operation.

Theorem 8 (State Merge Correctness) *Given two consecutive states* $S_x = \langle f_x^e, f_x^i \rangle$, *and* $S_{x+1} = \langle f_{x+1}^e, f_{x+1}^i \rangle$. *The state* S_m *is a correct merge of* S_x *and* S_{x+1} *if it is computed as* $\langle f_m^e, f_m^i \rangle$, *where* $f_m^i = (f_x^i \diamond f_{x+1}^e)$ *and* $f_m^e = (f_x$ - $f_m^i)$.

Proof: *Due to the operations merge,* f_x *will be redivided into different effective and idle subsets with respect to* Op_m^{Pre} *to compute* S_m *according to the behavior state definition (Definition 19). Hence,* $f_m^i = f_x \diamond Op_m^{Pre}$....*(1). By substituting the value of* Op_m^{Pre} *indicated in Proposition 5 in (1), we reach that* $f_m^i = f_x \diamond (Op_{x+1}^{Pre} + (Op_{x+2}^{Pre} \diamond Op_{x+1}^{Post}))$...*(2), which is represented by the shaded area in the Venn diagram depicted in Figure 4.12, taking into consideration that the semantic difference is used instead of the set-difference.*

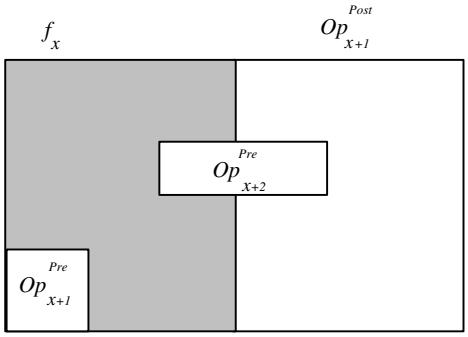

Figure 4.12: f_m^i *Elements*

By rearranging the terms of Equation (2) using the semantic difference definition (Definition 15), we reach $f_m^i = (f_x \diamond Op_{x+1}^{Pre}) \diamond (Op_{x+2}^{Pre} \diamond Op_{x+1}^{Post})$....*(3). As* $f_{x+1}^e \vDash Op_{x+2}^{Pre}$, *this implies that* $f_{x+1}^e \vDash (Op_{x+2}^{Pre} \diamond Op_{x+1}^{Post})$. *Hence* f_{x+1}^e *can substitute* $(Op_{x+2}^{Pre} \diamond Op_{x+1}^{Post})$. *Therefore, we can rewrite Equation (3) as* $f_m^i = (f_x \diamond Op_{x+1}^{Pre}) \diamond f_{x+1}^e$...*(4). As* $f_x^i = f_x \diamond Op_{x+1}^{Pre}$ *(Theorem 2), we can rewrite Equation (4) as,* $f_m^i = (f_x^i \diamond f_{x+1}^e)$. *Also, as* $f_m = f_x$, *and* $f_m^e = f_m$ - f_m^i *according to Theorem 2, this implies that* $f_m^e = f_x$ - f_m^i.

During state sequence matching using m-to-n state matching approach, a target state is examined against a source state at a time. When a target state cannot be matched with a source state, the possibility of a source state down expansion is examined to see if the target state can be matched. To down expand a source state S_x, first a target state S_t is required to be specified, as it will be the reference for the expansion. Second, the states that

will be merged with S_x to match the target state should be identified. Third, these states should be merged into a new state; starting from the state that has the least sequence index reaching to the state that has the highest sequence index. Finally, the resulting merge state is examined against the target state, that if it matches the target state this means S_x is down expandable with respect to S_t. Proposition 6 indicates how a down expandable source state is determined.

Proposition 6 (Source Down Expansion) *Given a source state S_x, a target state S_t, and a goal G such that $S_x \in \beta_i$ and $S_t \in \beta_j$. S_x is down expandable with respect to S_t and G (denoted as $S_x \downarrow_G S_t$) if ($S_x \not\trianglerighteq_G S_t$) and $\exists S_e$, $S_e \in \beta_i$ such that ($e > x$) and ($S'_x \trianglerighteq_G S_t$), where S'_x is the resulting state from merging the states from S_x to S_e.*

Proof: *The proposition is a direct realization for Definition 29.*

The *expansion step* of a state S_x with respect to S_t and G is the number of states merged to S_x in order to match S_t. Figure 4.13 depicts a source down expansion case. It indicates that S_t cannot be matched with S_x, as $\Xi(S_t) = \{a, d, z\}$ and $\Xi(S_x) = \{a\}$.

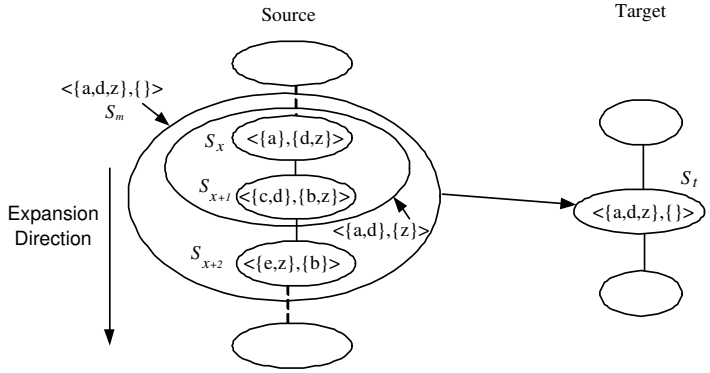

Figure 4.13: Source Down Expansion

S_x is expanded one state at a time until the expanded state can match S_t. This occurs when S_x, S_{x+1}, and S_{x+2} are merged. As merging S_x with $S_{x+1} = \langle\{a, d\}, \{z\}\rangle$, and merging S_x, S_{x+1}, $S_{x+2} = \langle\{a, d, z\}, \{\}\rangle$.

When $S_x \not\trianglerighteq_G S_t$ and S_x is not down expandable with respect to S_t, a matching cluster could be obtained if S_x is merged with predecessor states. We identify this case as a source

reverse expansion, as indicated in Proposition 7. However, this requires a Clustering Restructuring Operation (CRO) to take place, as depicted in Figure 4.14. It shows that S_x cannot match S_t, but when S_x is reversely expanded with respect to S_t, a new cluster that can match S_t is found. CRO is performed as follows. A new cluster will be created by merging the new required matching cluster (seen in the figure as dotted) with the old clusters located within the expansion step of the source state. Same merge operation will be performed for the corresponding target matching states. Then all these old clusters and their matching pairs will be deleted.

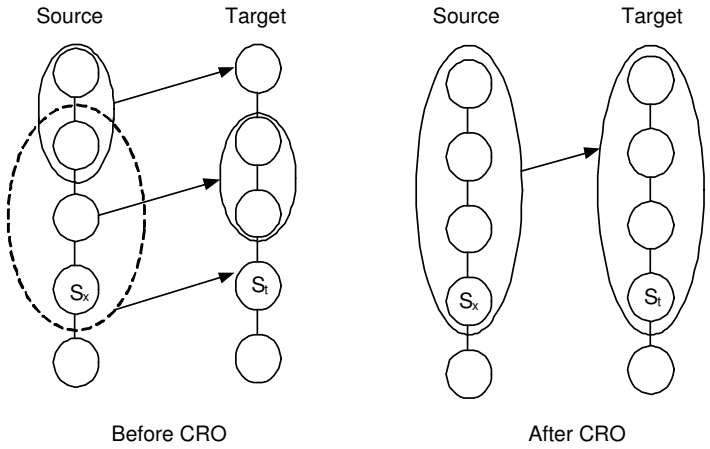

Figure 4.14: Clusters Restructuring Operation

Proposition 7 (Source Reverse Expansion) *Given a source state S_x, a target state S_t, and a given goal G such that $S_x \in \beta_i$ and $S_t \in \beta_j$. S_x is reverse expandable with respect to S_t and G (denoted as $S_x \uparrow_G S_t$) if ($S_x \not\trianglerighteq_G S_t$) and $\exists S_e$, $S_e \in \beta_i$ such that ($e < x$) and ($S'_x \trianglerighteq_G S_t$), where S'_x is the resulting expanded source state from merging the states from S_e to S_x.* ***Proof:*** *The proposition is a direct realization for Definition 29.*

For example, the state S_{x+2} in Figure 4.13 is a reverse expandable state with respect to S_t. When a target state cannot be matched with a source state, and the source state cannot be expanded in either directions. We check the possibility of merging the target state with previously matched target states, hoping to get rid of any vanishing constraints belonging to

the effective constraints of the target state. However, this requires CRO to be applied. This case is known as target reverse expansion, as indicated in Proposition 8.

Proposition 8 (Target Reverse Expansion) *Given a source state S_x, a target state S_t, and a given goal G such that $S_x \in \beta_i$ and $S_t \in \beta_j$. S_t is reversely expandable with respect to S_x and G (denoted as $S_t \Uparrow_G S_x$) if ($S_x \not\trianglerighteq_G S_t$) and $\exists S_e$, $S_e \in \beta_j$ such that ($e < t$) and ($S'_x \trianglerighteq_G S'_t$), where S'_t is the resulting state from merging the states from S_e to S_t, and S'_x is the state resulting from CRO.*

Proof: *The proposition is a direct realization for Definition 29.*

Figure 4.15 depicts a reverse target expansion case. It indicates that S_t cannot be matched with S_x, also it indicates that the expansion of S_x fails to match S_t. However, S_t is matched when it is reverse expanded, which requires S_x to be reverse expanded as well.

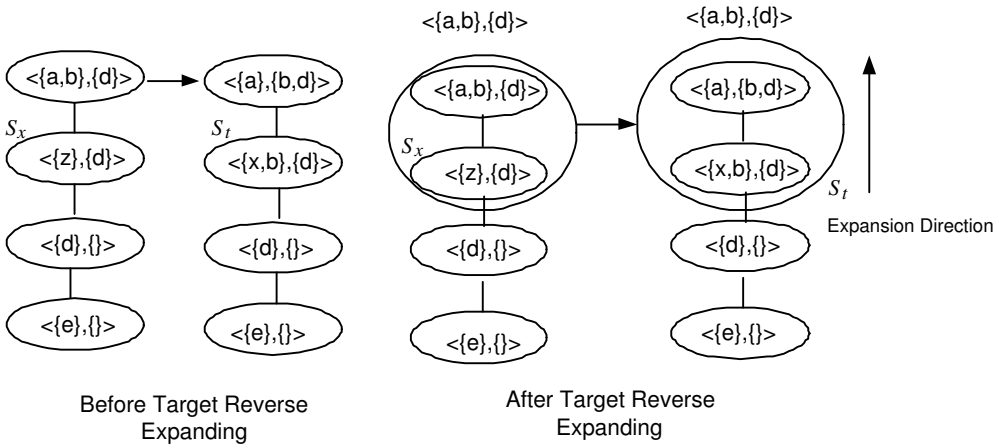

Figure 4.15: Example of Target Reverse Expansion

When a source state matches a target state, there is a possibility that this source state can match more target states. We check this possibility by merging the target state with its successors. We identify this case as target down expansion, as indicated in Proposition 9.

Proposition 9 (Target Down Expansion) *Given a source state S_x, a target state S_t, and a given goal G such that $S_x \in \beta_i$ and $S_t \in \beta_j$. S_t is down expandable with respect to S_x and G (denoted as $S_t \Downarrow_G S_x$) if ($S_x \trianglerighteq_G S_t$) and $\exists S_e \in \beta_j$ such that ($e > t$) and ($S_x \trianglerighteq_G S'_t$), where S'_t is the resulting expanded target state from merging the states from S_t to S_e.*

Proof: *The proposition is a direct realization for Definition 29.*

Figure 4.16 depicts a target down expansion case. It indicates S_x matches S_t, also indicates S_x matches the cluster from S_t, S_{t+1}, and S_{t+2} forming a state S_m.

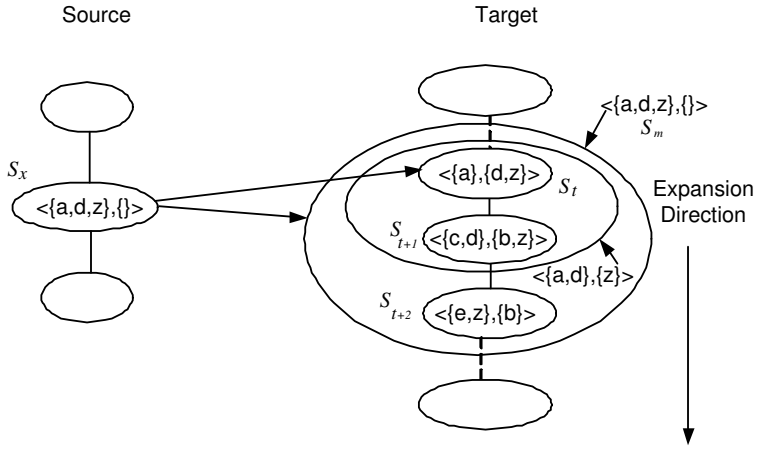

Figure 4.16: Target Down Expansion

4.3.3 State Sequence Clustering

This section first proposes a Sequence Mediator Procedure (SMP) that uses the different types of state expansions to re-cluster unmatched state sequences in order to reach a matching case (if possible). Then proposes the behavior substitutability matching approach that adopts SMP. SMP is the core of m-to-n state matching, it accepts two state sequences that could have different length, then returns two re-clustered state sequences with the same length (if possible), as indicated in Figure 4.17. The figure indicates the status of state sequence before, during, and after applying SMP. Later, the matching status between the source and target sequences is determined according to their corresponding re-clustered sequences.

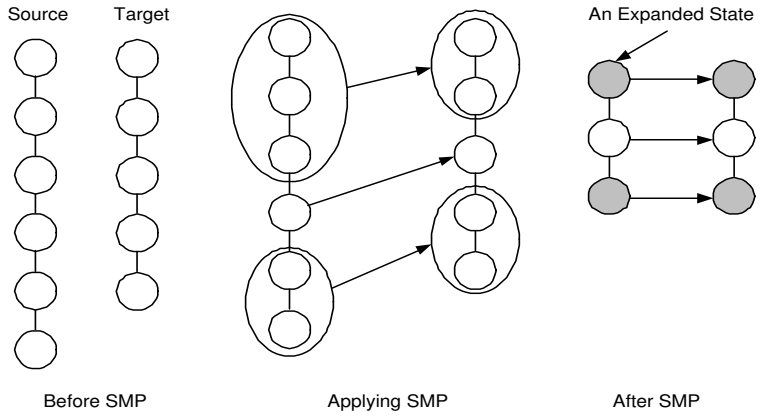

Figure 4.17: SMP Invocation Effect

SMP aims to find a matching source cluster for every target state. However, when a source state fails to match a target state, SMP checks if the source state could be down expandable with respect to the target state. When the source state cannot be down expanded with respect to the target state, SMP checks whether this source state could be reverse expanded with respect to the target state, which requires CRO to take place. When the source reverse expansion fails, SMP checks other successor source states against the target state before checking the target state reverse expansion, which requires CRO to take place as well. However, if this fails as well, SMP marks the target state as unmatched. SMP starts by examining the first state of the source target against the first state of the target sequence.

When the source state matches the target state according to Proposition 4, SMP applies Algorithm 4 to handle the matching case. Algorithm 4 indicates that SMP checks the possibility of target down expansion before registering the matching peer, then it proceeds to check the successor target and source states. However if there is no more successor source states, the remaining target states will be examined against the last state of the source sequence depending on reverse expansion for the last source state to match these target states.

When a source state cannot be expanded in either directions, SMP tries the successor source states to match the target state using the down and reverse source expansion scenarios, and it stores this unmatched source state for backtracking. If any of the successor source states succeeds to match the target state using any of the mentioned scenarios, a match case

is found and SMP handles the case as indicated earlier in Algorithm 4. When a target state cannot be matched to any source state, SMP tries reverse expanding the target state to find a match for it, when that fails this target state is considered unmatched, and the next target state will be examined. SMP starts to examine the next target state starting from the stored backtracked source state. Source backtracking is required to ensure that every target state got the chance to be examined against all unmatched source states.

Algorithm 4 SMP Matching Case Handling

Input: S[i] (the i^{th} source state), T[j] (the j^{th} target state), G (the required goal)
Output: True when sequences re-clustering is finished.

1: **Begin**
2: **if** (T[j] $\Downarrow_G$ S[i]) **then**
3: Expand T[j]
4: **end if**
5: Mark T[j], S[i] as matching peers
6: **if** ($j <$ ||T[ ]||) **then**
7: **if** ($i <$ ||S[ ]||) **then**
8: Apply Algorithm 5 over (S[i+1], T[j+1], G)
9: **else**
10: Apply Algorithm 5 over (S[i], T[j+1], G)
11: **end if**
12: **else**
13: Return True
14: **end if**
15: **End**

When every target state in the target sequence is examined and its matching peer is found, some of the source states could remain unmatched as indicated in Figure 4.18 (check state $\langle\{f\},\{\}\rangle$ in trace 6). SMP merges these unmatched source states to the predecessor matched source cluster. This will not change the matching status of the clusters, as their

effective constraints either remain the same (in case the effective constraints of the unmatched source states are vanishing constraints) or increase by new constraints that are independent from the effective constraints of the target cluster [1]. Algorithm 5 summarizes all the steps of SMP, the algorithm is initially applied over the first source state against the first target state. Example 2 illustrates a trace for SMP step by step.

The complexity of SMP is determined as indicated in Theorem 9.

Theorem 9 (SMP Complexity) *Complexity of SMP is $O(n^3)$.*

Proof: *The worst case for SMP to decide the matching status of a target state is to check source down expansibility, then check source reverse expansibility, then check target reverse expansibility...(1). A state down expansibility is simply $O(n)$ as it take states one by one and the merge it with the last obtained merged state. While the worst case of reversely expanding a state (source or target) is to check all the predecessors, but merging in this case has to be redone from the beginning of every prospective cluster, thus the worst case is to check the whole sequence which will be $1 + 2 + 3 + ... + n$, which equals to $\frac{n(n+1)}{2}$. Hence, reverse expansion costs $O(n^2)$. Therefore, according to (1), the worst case complexity to check the matching status of a target state will be $O(n^2)$. By repeating that for every state in the target sequence, the worst case complexity of SMP is $O(n^3)$.*

The complexity of SMP is relatively high, however we argue that in real-life scenarios the number of states of a given GAP will be relatively small (for example, 15 states).

Example 2 (A SMP Trace) *Figure 4.18 provides an example of matching a source operation sequence and a target operation sequence.*

SMP starts by checking the first source state with the first target state. It happens to be a match case, and SMP checks if the target state is down expandable with respect to the source state. It finds it is not down expandable. Thus, it proceeds to check the second target state with the second source state. It finds this is not a match case, so it checks the down expansibility of the source state, which fails. Thus, it checks its reverse expansibility, which fails as well. SMP checks other source states to be match that target state, which fails as well. So, SMP

[1] This is because, having X $\trianglerighteq$ Y, (X $\wedge$ Z) $\trianglerighteq$ Y holds as well when Z is independent from X and Y.

Algorithm 5 Sequence Mediator Procedure (SMP)

Input: S[i] (the i^{th} source state), T[j] (the j^{th} target state), G (the required goal)

Output: True when sequences re-clustering is finished.

```
1: Begin
2: if (S[i] ⊵_G T[j]) then
3:    Apply Algorithm 4 over (S[i], T[j],G)
4: else
5:    if (S[i] ↓_G T[j]) then
6:       Expand S[i]
7:       Apply Algorithm 4 over (S[i], T[j],G)
8:    else
9:       if (S[i] ↑_G T[j]) then
10:         Apply CRO over S[i]
11:         Apply Algorithm 4 over (S[i], T[j],G)
12:      else
13:         BackTrack =S[i]
14:         if (j < ||T[ ]||) then
15:            if (i < ||S[ ]||) then
16:               Apply Algorithm 5 over (S[i+1], T[j], G)
17:            else
18:               if (T[j] ⇑_G S[i]) then
19:                  Apply CRO over T[j]
20:                  Apply Algorithm 4 over (S[i], T[j],G)
21:               else
22:                  Mark T[j] as Unmatched
23:                  Apply Algorithm 5 over (BackTrack, T[j+1], G)
24:               end if
25:            end if
26:         else
27:            Merge unmatched source states with their predecessors.
28:            Return True
29:         end if
30:      end if
31:   end if
32: end if
33: End
```

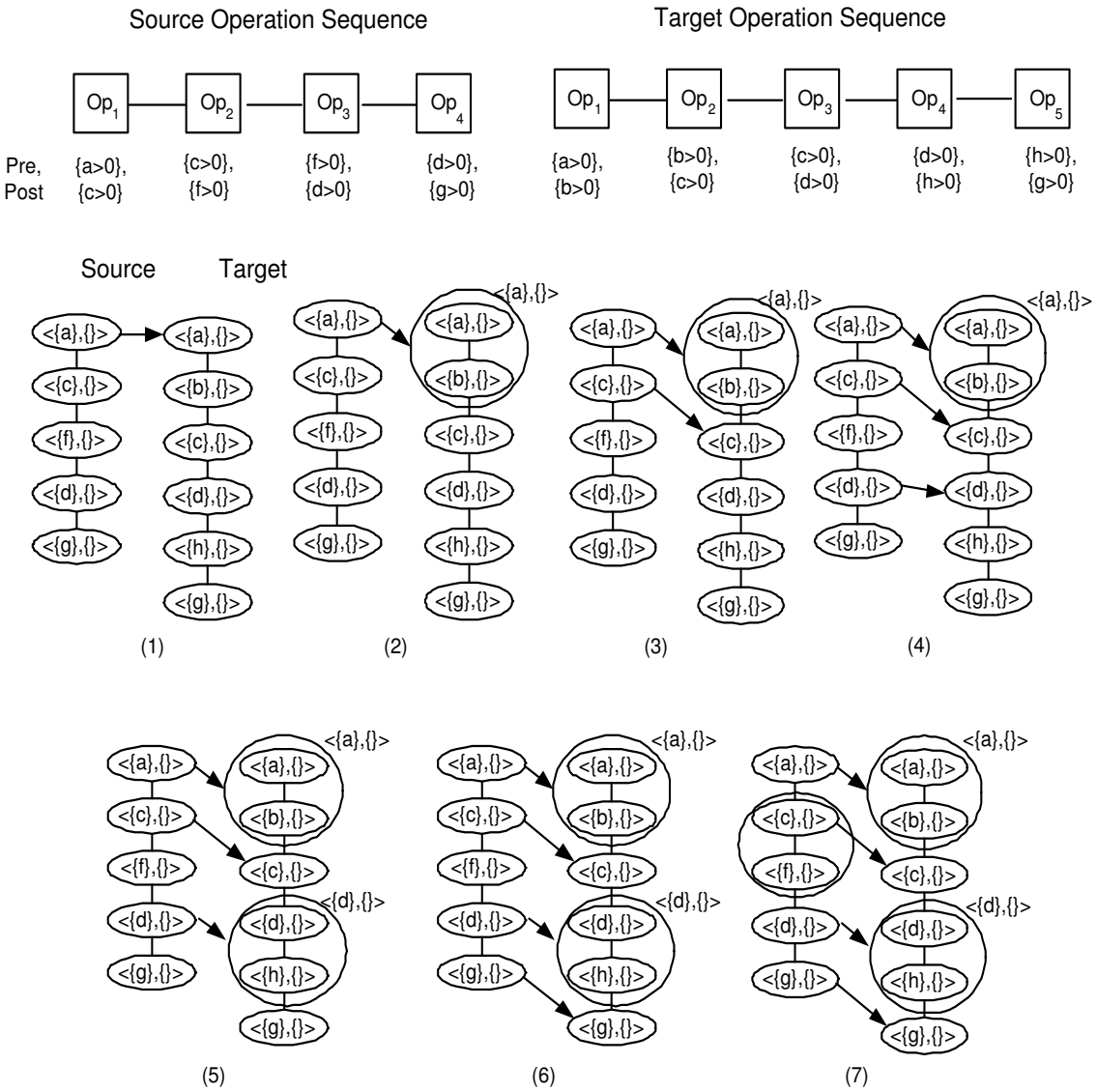

Figure 4.18: SMP Trace

starts to check the reverse expansibility of the target state. It finds it can be reverse expanded, so it restructures the target sequence to form a cluster from the first and second target states. Then, SMP checks the next target state $\langle\{c\},\{\}\rangle$ *against the next source state* $\langle\{c\},\{\}\rangle$*, it finds it is a matching case, so check the down expansibility of the target against the source state, which fails as the resulting scope is the same as the original state, so it moves to the next target state* $\langle\{d\},\{\}\rangle$ *and next source state* $\langle\{f\},\{\}\rangle$*. The source state cannot match the target state, also it is not expansible in either directions. Therefore, SMP starts to check the next source state* $\langle\{d\},\{\}\rangle$*, which happens to match the target state. Then it proceeds to the next target state after checking the expansibility of the target state*$\langle\{h\},\{\}\rangle$*, that will be examined against* $\langle\{g\},\{\}\rangle$*. As it is not a match case, and source state is not expandable in either directions and no more source states to be checked, SMP checks the reverse expansibility of the target state. It finds its reversely expandable, so it groups* $\langle\{d\},\{\}\rangle$*,* $\langle\{h\},\{\}\rangle$ *into one cluster and proceeds to the last target state, which happens to match the last source state. No backtracking is required as all the target sources are matched, as backtracking is only required when a target state cannot be matched. Finally, SMP find the source state* $\langle\{f\},\{\}\rangle$ *is unmatched, so it merges it to the predecessor matched source state, which is state* $\langle\{c\},\{\}\rangle$*, forming a new cluster with the same effective constraints* $\{c\}$ *as* f *is a vanishing constraint. When we switch the source with the target sequences, different matching clusters pairs will be obtained as indicated in Figure 4.19.*

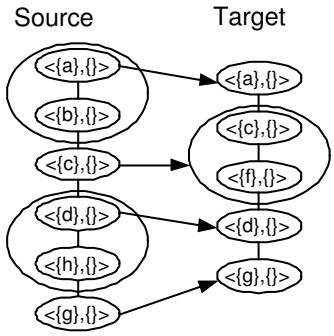

Figure 4.19: SMP Result when Source and Target are Switched

Adopting SMP, we propose an m-to-n semantic behavior matching approach. When two state sequences cannot be matched using Theorem 7, SMP is applied. Later, the matching status between the two re-clustered state sequences is examined again using Theorem 7. This is indicated in Theorem 10.

Theorem 10 (M-to-n Semantic Behavior Matching) *Given two behavior models* $\beta_x = \langle S_{(x,0)}, S_{(x,1)}, ..., S_{(x,n)} \rangle$, $\beta_t = \langle S_{(t,0)}, S_{(t,1)}, ..., S_{(t,m)} \rangle$, *and a given goal G,* β_x *matches* β_t *with respect to G using m-to-n state matching approach (denoted as* $\beta_x \approx_G \beta_t$*) if (*$\beta'_x \sim_G \beta'_t$*), where* β'_i, β'_j *are the behavior models resulting after applying SMP.*

Proof: *When SMP successfully re-clusters the source and target state sequences, the resulting state sequences will have the same length. Hence,* β'_i, β'_j *will be matched according to Theorem 7.*

Finally, a source behavior model substitutes a target behavior model when the behavior models are matched either by Theorem 7 or Theorem 10. Hence, behavior models substitutability is determined according to Theorem 11. Real-life examples for behavior model matching will be given in the next chapter.

Theorem 11 (Behavior Model Substitutability) *Given two behavior models* $\beta_x = \langle S_{(x,0)}, S_{(x,1)}, ..., S_{(x,n)} \rangle$, $\beta_t = \langle S_{(t,0)}, S_{(t,1)}, ..., S_{(t,m)} \rangle$, *and a given goal G,* β_x *substitutes* β_t *with respect to G (denoted as* $\beta_x \trianglerighteq_G \beta_t$*) if (*$\beta_x \sim_G \beta_t$*)* $\vee$ *(*$\beta_x \approx_G \beta_t$*).*

Proof: *When (n=m), Theorem 7 is used to match the behavior models. When Theorem 7 fails or (*$n \neq m$*), Theorem 10 will be used to determine the matching status between the behavior models. Hence, a source behavior model matches a target behavior model when either Theorem 7 or Theorem 10 holds.*

4.4 Conclusion

In this chapter, we introduced the concept of a matching scheme. We proposed a new matching scheme for matching the high-level specifications of semantic web services, known as the Functional Substitutability Matching Scheme (FSMS). We defined FSMS as ⟨ Functionality,

Substitutability, Goal Achievement ⟩, where functionality is the comparison aspect, substitutability is the matching rule and goal achievement is the matching correctness criterion. Functionality of a web service is represented by the supported describing-constraints and the supported behavior models. Similarly, the functionality of users is represented by the required describing-constraints and the required behavior models. Also in this chapter, we indicated how the behavior models are automatically extracted from the defined G^+ models. We proposed the constraints substitutability concept to determine users' constraints satisfiability. We discussed the different properties for the goal achievement operator which provide the matching process guidelines. We proposed the algorithms required for FSMS realization. We indicated how behavior models are matched using FSMS. Finally, we proposed an m-to-n state matching approach for matching state sequences, proposing a sequence mediator procedure that mediates between a source state sequence and a target state sequence such that the sequences could be semantically matched.

Chapter 5

Correctness-Aware Service Matching Approaches

This chapter proposes two new correctness-aware service matching techniques adopting FSMS: a direct matching technique and an aggregate matching technique. The direct matching technique primarily matches web services according to the substitutability of the GAPs extracted from their G^+ models. The aggregate matching technique adopts a dynamic sequential aggregate approach such that the returned solution is a sequence of web services that can substitute the required GAPs. The matching techniques require no user interaction during the matching process, and adopt the semantics of the involved application domain to mediate between services' descriptions and users' requests. Finally, this chapter evaluates the devised matching techniques against the service matching approaches discussed in Chapter 2 using simulation experiments.

5.1 Service Direct Matching Approach

In service direct matching, only one service's description is examined against the user's request. According to the goal achievement definition (Definition 26, page 100), a user's goal is considered achieved when the required post-constraints are satisfied by transforming the pre-constraints into the required post-constraints via the required behavior; satisfying the

required describing-constraints. As indicated in Chapter 3 (Table 3.3, page 63), the goal achievement semantics differs according to the involved perspective (that is a user's perspective or a service's perspective). In the service perspective, the service's pre-constraints need to be satisfied such that the service can be invoked, while the service guarantees the satisfaction of its post and describing-constraints. In contrast, in the user's perspective, the user guarantees the satisfaction of the required pre-constraints and needs the post and describing-constraints to be satisfied. Hence, a service that can be invoked by the user's pre-constraints and can satisfy the required post and describing-constraints via the required behavior is considered a correct match for the request. Therefore, the user's pre-constraints will be checked against the service's pre-constraints, to see if they can substitute the service's pre-constraints. Similarly, the service's post and describing-constraints will be checked against the user's post and describing-constraints, respectively, to see if they can substitute them, as indicated in Figure 5.1. However, in order to consider the service as a correct match for the request, another check needs to be done according to the goal achievement definition (Definition 26) that is the service's behavior model needs to substitute the user's behavior model.

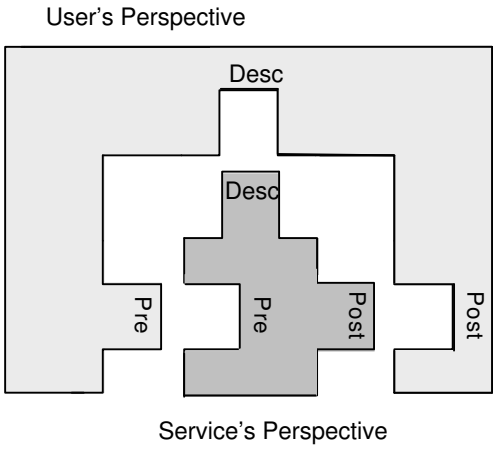

Figure 5.1: The Service Direct Matching Approach

When a service's behavior substitutes a user's behavior this means the service's sequence of operations will substitute the sequence of operations required by the user, as indicated in

Figure 5.2. It indicates that the substitution of operation sequences implies that a user has to follow a new behavior to achieve the required goal. This new behavior is known as the *anticipated User behavior*, which is mainly based on the service's behavior.

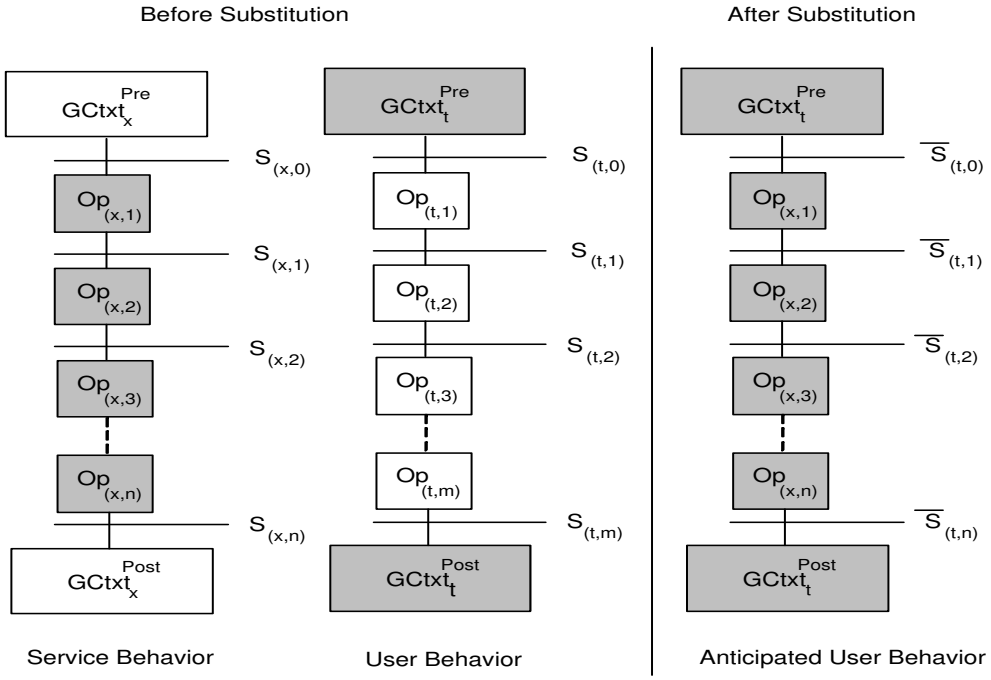

Figure 5.2: Anticipated User Behavior

Definition 30 (Anticipated User Behavior) *Given a service behavior model β_s, the anticipated user behavior corresponding to β_s (denoted as β_s^a) is the user behavior resulting from following the transitions indicated in β_s.*

When the anticipated user behavior substitutes the required user behavior according to Theorem 11, the service behavior model is considered a match for the required behavior. The difference between the service's behavior and the user's anticipated behavior is that the service's behavior is computed based on the pre-constraints and the operation sequence of the service, while the user's anticipated behavior is computed according to the pre-constraints of the user and the operation sequence of the service.

The computation of the anticipated behavior model requires to extend the definitions of

the effective and idle constraints given in Chapter 4 as well as the semantic difference and the GAP correctness definitions given in Chapter 3. This is because a user might use different concepts than the ones used in services' models, hence a mediation process with respect to the user's goal must be taken into consideration. Hence the computation of the persisting constraints as well as the classification of constraints into effective and idle will be based on the user's required goal. According to Definition 17, an effective constraint at transition point x is the constraint that its scope belongs to the scope of Op_{x+1}^{Pre}. This will be extended by considering a given constraint as effective at transition point x when its scope is reachable (directly or indirectly) to any element of the scope of Op_{x+1}^{Pre}.

Definition 31 (Goal-based Effective Constraint) *Given a constraint $Cnst_i$ such that $Cnst_i \in f_x$ and a goal G, $Cnst_i$ is considered an effective constraint at a given transition point x with respect to G when $(\Xi(Cnst_i) \rightharpoonup_G \Xi(Cnst_j)) \vee (\Xi(Cnst_i) \curvearrowright_G \Xi(Cnst_j))$, where $Cnst_j \in Op_{x+1}^{Pre}$.*

Similarly, an idle constraint at transition point x according to Definition 18 is the constraint that its scope does not belong to the scope of Op_{x+1}^{Pre}. This will be extended by considering a given constraint as idle at transition point x when its scope is not reachable (directly or indirectly) to any element of the scope of Op_{x+1}^{Pre}.

Definition 32 (Goal-based Idle Constraint) *Given a constraint $Cnst_i$ such that $Cnst_i \in f_x$ and a goal G, $Cnst_i$ is considered an idle constraint at a given transition point x with respect to G if $\forall\ Cnst_j \in Op_{x+1}^{Pre}$, $\nexists\ Cnst_j$ such that $(\Xi(Cnst_i) \rightharpoonup_G \Xi(Cnst_j)) \vee (\Xi(Cnst_i) \curvearrowright_G \Xi(Cnst_j))$.*

The semantic difference between two sets of constraints according to Definition 15 is determined using constraint direct satisfiability. This will be extended by using constraints substitutability instead.

Definition 33 (Goal-based Semantic Difference) *Given two sets of constraints f_i and f_j and a goal G. The semantic difference between f_i and f_j with respect to G (denoted as $f_i \diamond_G f_j$) is f_k, where f_k is the maximal subset of f_i such that $(f_i - f_k) \trianglerighteq_G f_j$.*

Similarly, GAP correctness indicated in Definition 14 is determined using constraint direct satisfiability. This will be extended by using constraints substitutability instead.

Definition 34 (Goal-based GAP Correctness) *Given a goal achievement pattern of a goal G defined as* $Gp = \langle\ Op,\ GCtxt,\ OpSeq\ \rangle$, *Gp is considered correctly defined with respect to G if* $(\forall_{x=0}^{x=n-1}\ f_x \trianglerighteq_G Op_{x+1}^{Pre}) \wedge (f_n \trianglerighteq_G GCtxt^{Post})$, *where n is the number of operations in OpSeq,* $f_0 = GCtxt^{Pre}$, *and* f_x *is the set of constraints at the transition point x between* Op_x, Op_{x+1} *such that* Op_x, $Op_{x+1} \in OpSeq$.

To compute the anticipated user behavior, the persisting constraints at a given transition point will be determined similarly as indicated in Theorem 2 by adopting the goal-based semantic difference operator instead, as indicated in Theorem 12.

Theorem 12 (Goal-based Persisting Constraints) *Given a goal achievement pattern of a goal G defined as* $Gp = \langle\ Op,\ GCtxt,\ OpSeq\ \rangle$. *The set of persisting constraints* f_{x+1} *at transition point* $(x+1)$ *is computed as* $f_{x+1} = f_x^i + Op_{x+1}^{Post}$, *where* $0 \leq x \leq n-1$, $(f_0 = GCtxt^{Pre})$, $(f_x^i = f_x \diamond_G Op_{x+1}^{Pre})$, $(f_x^e = f_x - f_x^i)$, *and n is the number of operations in OpSeq.*

Proof: *Same as Theorem 2 adopting Definition 34 instead of Definition 14.*

After computing the anticipated user behavior model, we can easily determine whether a given service is achieving the required user's goal or not, by checking if the service context is substituting the user's context and the user's anticipated behavior is substituting the required behavior model, as indicated in Theorem 13.

Theorem 13 (Service Direct Matching) *Given the goal achievement processes of a service's GAP (denoted as* GP_s*) and a user's GAP (denoted as* GP_u*) are described as* $G_s \vdash \langle GCtxt_s^{Pre}, GCtxt_s^{Post}, \beta_s, GCtxt_s^{Desc}\rangle$, *and* $G_u \vdash \langle GCtxt_u^{Pre}, GCtxt_u^{Post}, \beta_u, GCtxt_u^{Desc}\rangle$, *respectively.* GP_s *matches* GP_u *(denoted as* $GP_s \trianglerighteq_{G_u} GP_u$*) if* $(GCtxt_u^{Pre} \trianglerighteq_{G_u} GCtxt_s^{Pre}) \wedge (GCtxt_s^{Post} \trianglerighteq_{G_u} GCtxt_u^{Post}) \wedge (GCtxt_s^{Desc} \trianglerighteq_{G_u} GCtxt_u^{Desc}) \wedge (\beta_s^a \trianglerighteq_{G_u} \beta_u)$.

Proof: *As the user goal* G_u *is the one needs to be achieved, all the substitution checks will be performed according to* G_u. *In other words, the search for transformations that can satisfy the above conditions will be done in the corresponding segment of* G_u.

$G_s \vdash \langle GCtxt_s^{Pre}, GCtxt_s^{Post}, \beta_s, GCtxt_s^{Desc}\rangle$ implies that $\sigma_s(GCtxt_s^{Pre}) = GCtxt_s^{Post}$....(1). Similarly, $G_u \vdash \langle GCtxt_u^{Pre}, GCtxt_u^{Post}, \beta_u, GCtxt_u^{Desc}\rangle$ implies that $\sigma_u(GCtxt_u^{Pre}) = GCtxt_u^{Post}$ ....(2). In order to achieve G_u via the GP_s $\sigma_s(GCtxt_u^{Pre}) = GCtxt_u^{Post}$ must hold...(3). Hence, ($GCtxt_u^{Pre} \trianglerighteq_{G_u} GCtxt_s^{Pre}$) must hold in order to be able to invoke σ_s....(4). Assuming (4) holds and the service can be invoked, hence $GCtxt_s^{Post}$ and $GCtxt_s^{Desc}$ are guaranteed to be satisfied....(5). So in order to satisfy (3), ($GCtxt_s^{Post} \trianglerighteq_{G_u} GCtxt_u^{Post}$) must hold...(6). However, Definition 26 implies that $GCtxt_u^{Desc}$ must be satisfied in order to achieve G_u, according to (5) $GCtxt_s^{Desc}$ is guaranteed to be satisfied, hence ($GCtxt_s^{Desc} \trianglerighteq_{G_u} GCtxt_u^{Desc}$) must hold in order to achieve G_u...(7). When equations (4),(6),(7) hold, equation (3) will hold as $\sigma_s(GCtxt_u^{Pre}) = GCtxt_u^{Post}$ is guaranteed to be satisfied. However, in order to guarantee that β_s transforms $GCtxt_u^{Pre}$ into $GCtxt_u^{Post}$ substituting the required behavior ($\beta_s^a \trianglerighteq_{G_u} \beta_u$) needs to hold as well...(8). So in order to achieve G_u via GP_s (4),(6),(7), and (8) must hold.

According to the user's semantics, a service that can achieve the required goal via the same number of state transitions defined in the required GAP is better than a service that can achieve the required goal via a different number of state transitions. However, when the m-to-n state matching is adopted, the same number of state transitions required by the user is not guaranteed to be reached due to the application of CRO. Therefore, we propose to rank the returned matching results according to the *Sequence Similarity Ratio*.

Definition 35 (Sequence Similarity Ratio) *Given a service behavior model β_s, and a user behavior model β_u. The similarity between β_s and β_u (denoted as ρ_s) is computed as the ratio $\frac{\|\beta_s^{a\prime}\|}{\|\beta_u\|}$, where $\beta_s^{a\prime}$ is the anticipated user behavior after applying SMP.*

Example 3 (Service Direct Matching) *The user G^+ model depicted in Figure 3.10 has a matching GAP in the GAP-forest depicted in Figure 3.13 that corresponds to the service G^+ model depicted in Figure 3.11. For easy referencing, we depict the matching GAPs in Figure 5.3. Adopting the substitutability graph segment defined in Table 3.2 (page 60), we can see that the pre-constraints of the user substitute the pre-constraints of the service. Also,*

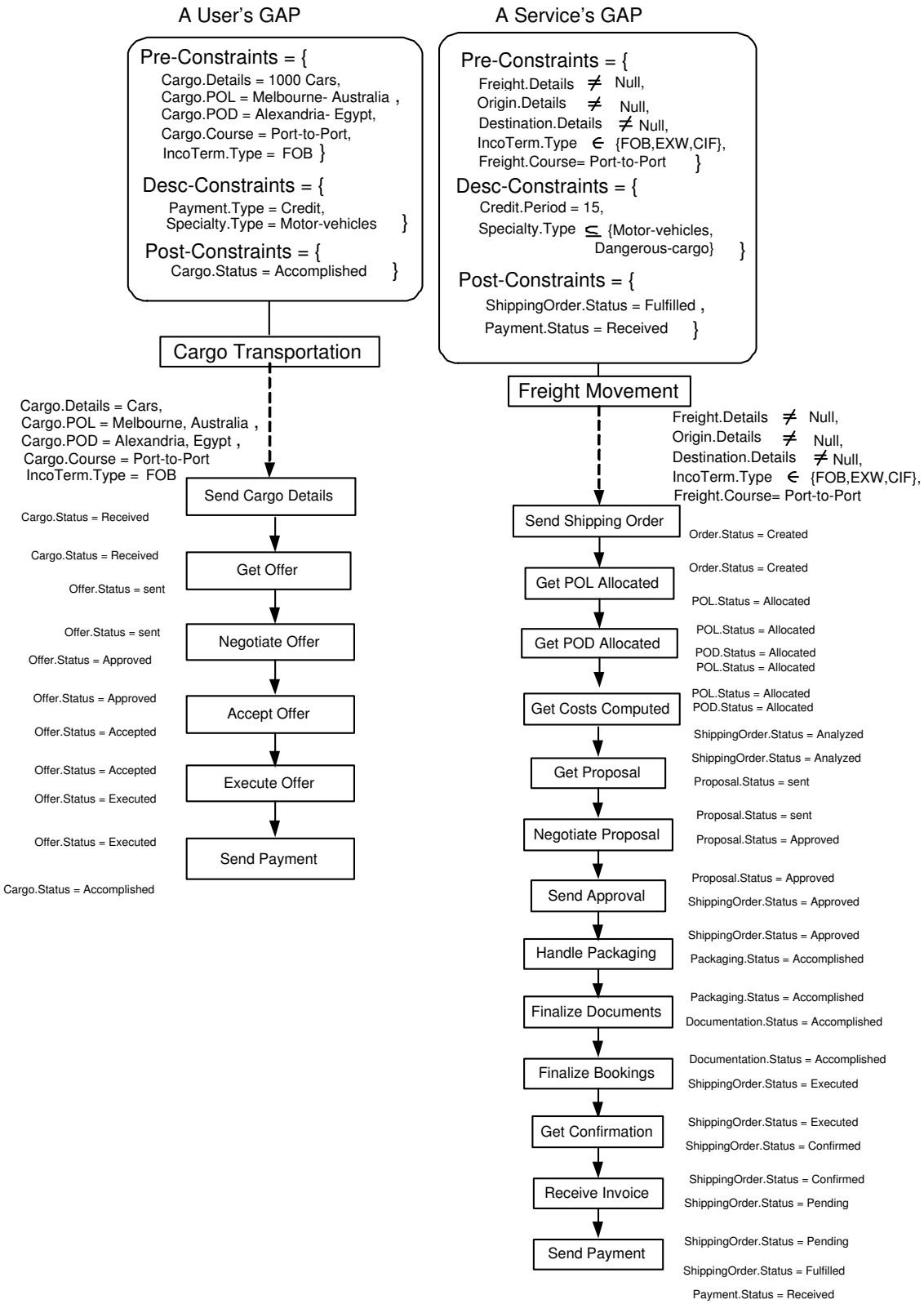

Figure 5.3: Service Direct Matching Example

the describing-constraints and the post-constraints of the service substitute the describing-constraints and the post-constraints of the user, respectively. The user's behavior model, the service's behavior model, and the anticipated behavior model are indicated in Table 5.1, Table 5.2, and Table 5.3, respectively. By applying SMP to mediate between the anticipated behavior and the user behavior, the resulting re-clustered sequences are depicted in Figure 5.4. This service matches the user request with ρ_s equal to $\frac{6}{7}$, meaning that it can achieve the user's goal following 85.7% *of the required steps.*

S_0	⟨ {*Cargo.Details* = 1000 Cars, *Cargo.POL* = Melbourne-Australia, *Cargo.POD* = Alexandria-Egypt, `IncoTerm.Type` = FOB}, {*Cargo.Course* = Port-to-Port}⟩
S_1	⟨ {*Cargo.Course* = Port-to-Port, *Cargo.Status* = Received} , {} ⟩
S_2	⟨ {*Offer.Status* = Sent}, {} ⟩
S_3	⟨ {*Offer.Status* = Approved}, {} ⟩
S_4	⟨ {*Offer.Status* = Accepted} , {} ⟩
S_5	⟨ {*Offer.Status* = Executed} , {} ⟩
S_6	⟨ {*Cargo.Status* = Accomplished} , {} ⟩

Table 5.1: User's Behavior Model

S_0	⟨ { *Freight.Details* ≠ Null, *Origin.Details* ≠ Null, *Destination.Details* ≠ Null, *Freight.Course* = Port-to-Port, `IncoTerm.Type` ∈ {FOB,EXW,CIF}} , {} ⟩
S_1	⟨ {*ShippingOrder.Status* = Created} , {} ⟩
S_2	⟨ {*POL.Status* = Allocated} , {} ⟩
S_3	⟨ {*POL.Status* = Allocated, *POD.Status* = Allocated} , {} ⟩
S_4	⟨ {*ShippingOrder.Status* = Analyzed} , {} ⟩
S_5	⟨ {*Proposal.Status* = Sent} , {} ⟩
S_6	⟨ {*Proposal.Status* = Approved} , {} ⟩
S_7	⟨ {*ShippingOrder.Status* = Approved} , {} ⟩
S_8	⟨ {*Packaging.Status* = Accomplished}, {} ⟩
S_9	⟨ {*Documentation.Status* = Accomplished}, {} ⟩
S_{10}	⟨ {*ShippingOrder.Status* = Executed} , {} ⟩
S_{11}	⟨ {*ShippingOrder.Status* = Confirmed} , {} ⟩
S_{12}	⟨ {*ShippingOrder.Status* = Pending} , {} ⟩
S_{13}	{*ShippingOrder.Status* = Fulfilled, *Payment.Status* = *Received*}, {} ⟩

Table 5.2: Service's Behavior Model

Later, the returned list of services returned are filtered using the LLFC, NFC and NTC contexts. In NTC filtering, services with prices greater than the required price are ignored. In LLFC filtering, services that do not support the required device types are ignored, also services that require more bandwidth than the specified by the user are ignored as well.
In NFC filtering, services with reliability values less than the required reliability probability are ignored, services with availability percentage less than the required availability percentage

S_0	⟨ {*Cargo.Details* = 1000 Cars, *Cargo.POL* = Melbourne-Australia, *Cargo.POD* = Alexandria-Egypt, *Cargo.Course* = Port-to-Port, `IncoTerm.Type` = FOB} , {}⟩
S_1	⟨ {*ShippingOrder.Status* = Created} , {} ⟩
S_2	⟨ {*POL.Status* = Allocated} , {} ⟩
S_3	⟨ {*POL.Status* = Allocated, *POD.Status* = Allocated} , {} ⟩
S_4	⟨ {*ShippingOrder.Status* = Analyzed} , {} ⟩
S_5	⟨ {*Proposal.Status* = Sent} , {} ⟩
S_6	⟨ {*Proposal.Status* = Approved} , {} ⟩
S_7	⟨ {*ShippingOrder.Status* = Approved} , {} ⟩
S_8	⟨ {*Packaging.Status* = Accomplished}, {} ⟩
S_9	⟨ {*Documentation.Status* = Accomplished}, {} ⟩
S_{10}	⟨ {*ShippingOrder.Status* = Executed} , {} ⟩
S_{11}	⟨ {*ShippingOrder.Status* = Confirmed} , {} ⟩
S_{12}	⟨ {*ShippingOrder.Status* = Pending} , {} ⟩
S_{13}	{*ShippingOrder.Status* = Fulfilled, *Payment.Status* = *Received*}, {} ⟩

Table 5.3: Anticipated Behavior Model

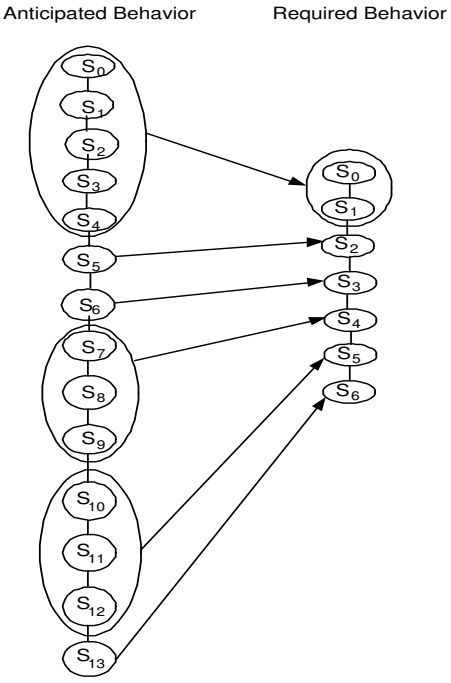

Figure 5.4: The Anticipated and the Required Behaviors after Applying SMP

are ignored, also services with less reputation rank than the required rank are ignored as well. When filtering services based on their roles (before matching the G^+ models), the user's role must satisfy one of the supported service's roles. As roles are concepts representing domain actors, concepts substitutability graph will be used to match the roles' constraints, such that roles' constraints substitutability will be determined according to the user's goal and the user's pre-constraints.

Theorem 13 indicates how a single service's GAP matches a single user's GAP, but when the user's request includes a complex G^+ model, the whole user's GAP-forest need to be examined against the service's GAP-forest, that when a GAP from the service GAP-forest matches a GAP from the user's GAP-forest using Theorem 13, regardless of their abstraction levels, the service is considered a match, as indicated in Proposition 10. We identify this approach for GAP-forest matching as a loose approach, as no restrictions required on the abstraction levels of the matched GAPs.

Proposition 10 (Loose Direct GAP-forest Matching) *Given a user's GAP-forest* ($Gpf_u = \{\langle L_i, Gp_{L_i}\rangle \mid 0 \leq i \leq n_u\}$) *of the goal* G_u*, and a service GAP-forest* ($Gpf_s = \{\langle L_j, Gp_{L_j}\rangle \mid 0 \leq j \leq n_s\}$) *of the goal* G_s*.* Gpf_s *matches* Gpf_u (*denoted as* $Gpf_s \trianglerighteq_G Gpf_u$) *if* $\exists\ Gp_s$, Gp_u *such that* ($Gp_s \trianglerighteq_{G_u} Gp_u$)*, where* $Gp_s \in Gp_{L_j}$, $Gp_u \in Gp_{L_i}$, $Gp_{L_j} \in Gpf_s$, *and* $Gp_{L_i} \in Gpf_u$.

Proof: *A user goal can be achieved via any of the GAPs extracted from the defined* G^+ *model, hence having at least one GAP matched to a service's GAP using Theorem 13 guarantees the user's goal is achieved, which indicates a correct match result is found.*

Example 3 indicates a case of loose GAP-forest matching. However, in the service-to-service matching problem[1], the whole GAP-forest need to be matched, as this is important for other problems. For example, to improve the reliability of a composite service, the composite service could maintain a list of services that can accomplish a given task. In case a component service failed, it can choose from other services in the list, that can accomplish the same goal. Hence, we need the component service to be able to cover all the required GAPs, not only just one case. In the case of service-to-service matching, GAP-forests will be matched strictly

[1]In the service-to-service matching problem, a service's functionality is examined against another service's functionality in order to determine if they can achieve the same goals.

that every GAP in the target GAP-forest has a corresponding matching GAP in the source GAP-forest, as indicated in Proposition 11.

Proposition 11 (Strict Direct GAP-forest Matching) *Given two GAP-forests* $Gpf_i = \{\langle L_i, Gp_{L_i}\rangle \mid 0 \leq i \leq n_i\}$, $Gpf_j = \{\langle L_j, Gp_{L_j}\rangle \mid 0 \leq j \leq n_j\}$, *and a goal G,* Gpf_j *can be substituted by* Gpf_i *with respect to G (denoted as* $Gpf_i \trianglerighteq_G Gpf_j$*) if* $\forall L_j$ *belongs to* Gpf_j, $\exists$ L_i *belongs to* Gpf_i *such that* $\forall$ Gp_x, $\exists$ Gp_y *such that* ($Gp_y \trianglerighteq_G Gp_x$), *where* $Gp_x \in Gp_{L_j}$, $Gp_y \in Gp_{L_i}$.

Proof: *The achievement of the goal G needs to be guaranteed for all the defined GAPs, hence all the GAPs in the target GAP-forest must be matched.*

Figure 5.5 indicates the difference between loose and strict matching definitions. Figure 5.5 indicates that in case of adopting loose matching, the source GAP-forest matches the target GAP-forest as a source GAP from abstraction level-2 matches a target GAP in abstraction level-1, but in case of strict matching, the the two forest are matched as every GAP in the target forest is matched to a GAP from the source forest. In this thesis, we only adopt loose GAP-forest matching, as we are only concerned with achieving users' goals.

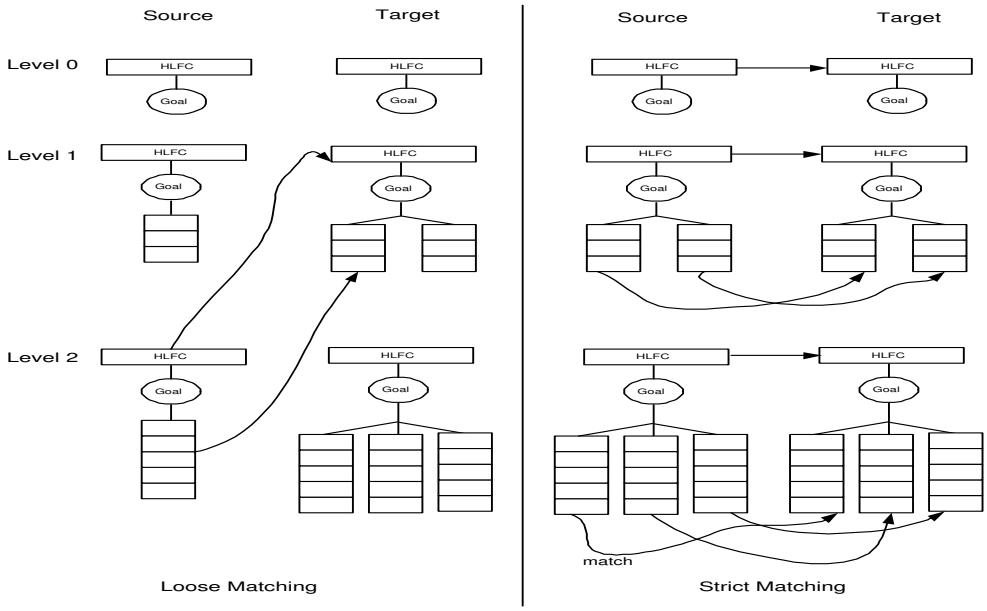

Figure 5.5: Strict versus Loose GAP Forest Matching

5.2 Service Aggregate Matching Approach

In this section, we propose a new service aggregate matching technique that is based on services' GAPs that are extracted from their G^+ model; adopting the semantics of the involved application domain to mediate between services' descriptions and users' requests. We adopt a sequential aggregation approach such that the solution for the aggregation problem will be a sequence of web services that functionally substitute the required GAP without interacting with users, as indicated in Figure 5.6.

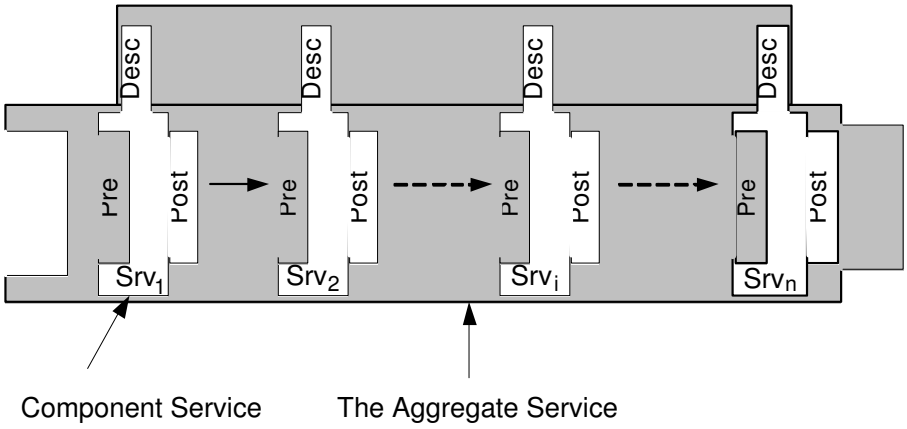

Figure 5.6: An Aggregate Service

The idea behind service sequential aggregation is to find a list of component services that can achieve different parts of the required goal achievement pattern. Later, from this list of component services, we determine the possible service sequences that can achieve the required goal achievement pattern. In order to determine these component services, we need to define the concepts of a *plug-in GAP*, and a *computation chunk*.

Definition 36 (Plug-in GAP) *A plug-in GAP is a service's GAP that can accomplish a continuous part of the required goal achievement pattern.*

Definition 37 (Computation Chunk) *A computation chunk is the continuous part of the required goal achievement pattern that can be achieved via a plug-in GAP of a given service Srv. A computation chunk is defined as $\langle ServiceID, Start, End \rangle$, where $ServiceID$ is a*

unique identifier for Srv, Start and End are the indices of the first and last states of the required behavior that are matched by the plug-in GAP of Srv.

Figure 5.7 depicts a plug-in GAP and its corresponding computation chunk. Figure 5.7 indicates that the behavior model of the service Srv with identifier equal to $Srv1000$[2] matches a computation chunk of the required behavior that starts from S_1 and ends at S_5. This computation chunk is defined as $< Srv1000, 1, 5 >$.

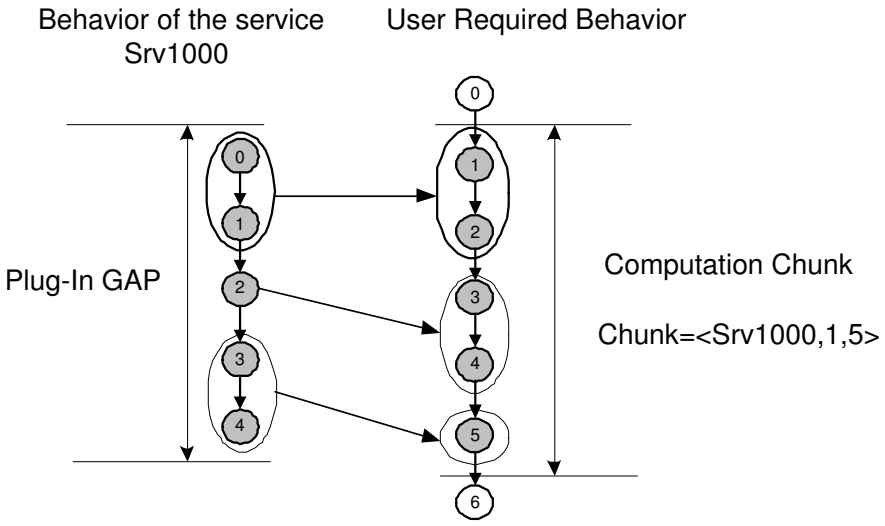

Figure 5.7: A Plug-in GAP and the corresponding Computation Chunk

Plug-in GAPs are simply determined by checking if the pre-constraints of a service GAP are substituted by a set of user's persisting constraints at a given transition point, and the corresponding service post-constraints substitute another set of user's persisting constraints at another following transition point, as indicated in Proposition 12.

Proposition 12 (Plug-in GAP) *Given the goal achievement processes of a service's GAP (denoted as GP_s) and a user's GAP (denoted as GP_u) are described as $G_s \vdash \langle\ GCtxt_s^{Pre}$, $GCtxt_s^{Post}$, β_s, $GCtxt_s^{Desc}\ \rangle$, and $G_u \vdash \langle\ GCtxt_u^{Pre}$, $GCtxt_u^{Post}$, β_u , $GCtxt_u^{Desc}\ \rangle$, respectively. Gp_s is a plug-in GAP of Gp_u (denoted as $Gp_s \preceq_{G_u} Gp_u$) if $\exists\ f_x$ and f_{x+y} such that*

[2]In practice, ServiceID will be the service's URI.

$(f_x \trianglerighteq_{G_u} GCtxt_s^{Pre}) \wedge (GCtxt_s^{Post} \trianglerighteq_{G_u} f_{x+y})$, where f_x and f_{x+y} are the persisting constraints at transition points x and $x+y$ of β_u, and $y \geq 1$.

Proof: *$G_s \vdash \langle\ GCtxt_s^{Pre},\ GCtxt_s^{Post},\ \beta_s,\ GCtxt_s^{Desc}\ \rangle$ implies that $\sigma_s(GCtxt_s^{Pre}) = GCtxt_s^{Post}$...(1). Having $(f_x \trianglerighteq_{G_u} GCtxt_s^{Pre}) \wedge (GCtxt_s^{Post} \trianglerighteq_{G_u} f_{x+y})$ implies that f_x can be transformed into f_{x+y} via Gp_s, as $\sigma_s(f_x) = f_{x+y}$ is guaranteed to hold. Hence, Gp_s is considered a plug-in of Gp_u according to Definition 37. As the minimum number of operations allowed to be defined in a given GAP is one, the corresponding behavior model will consist of two states, hence the minimum value allowed for y is one. Therefore, $y \geq 1$ must hold in order to have a meaningful behavior model that could be a plug-in of Gp_u.*

In order to obtain a list of the plug-in GAPs of a given required GAP, the matchmaker has to scan the service registry for services that satisfy the conditions indicated in Proposition 12, as indicated in Figure 5.8. It shows that there are two plug-in services for the required GAP: $Srv1000$ and $Srv2000$. $Srv1000$ covers the subsequence from S_0 to S_2, and $Srv2000$ covers the subsequence from S_1 to S_3.

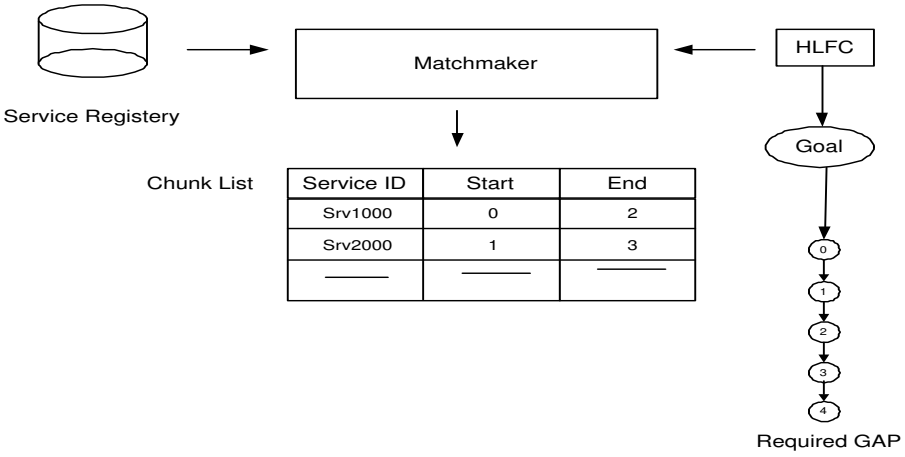

Figure 5.8: Chunk List

Having this chunk-list, the matchmaker needs to find if there exists a sequence of these chunks that can match the request. In order to be able to do so, we propose a special data structure that facilitates the aggregation process. This data structure organizes the chunk-list into array of stacks, where every stack contains the chunks that have the same *Start*

value, as indicated in Figure 5.9. The figure indicates that the array index is the same as the state sequence index of the request, and for every state, there is a stack of chunks that start with its index.

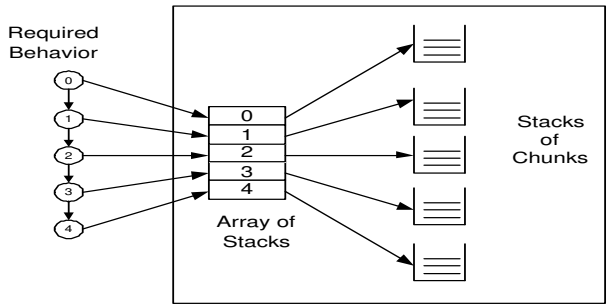

Figure 5.9: Array of Stacks

By doing so, the matchmaker can easily locate a sequence of chunks that fulfil the request, by tracing the values of the *Start* and *End* of the chunks such that the resulting sequence cover the whole state sequence of the request without any missing state transitions or overlaps, as indicated in Figure 5.10.

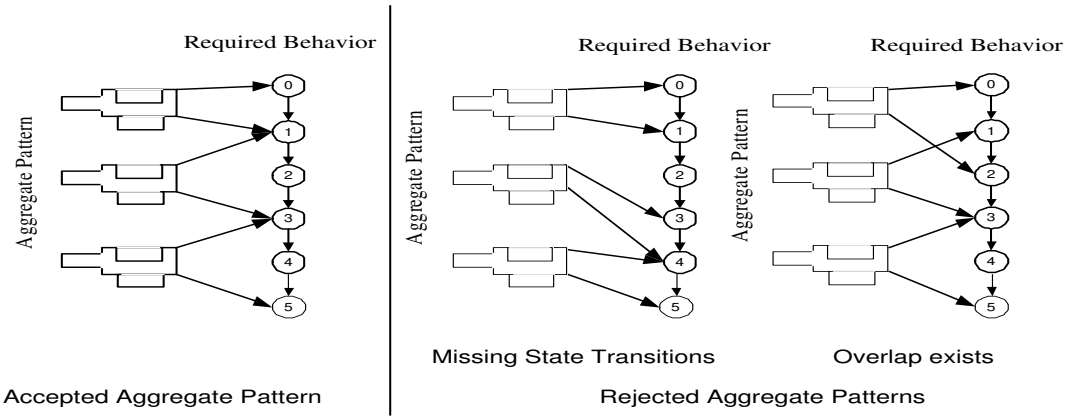

Figure 5.10: Accepted versus Rejected Aggregate Patterns

These accepted aggregate patterns will be filtered by the matchmaker such that an accepted pattern is considered a solution when its aggregate high-level functional context matches the required high-level functional context. First, we indicate how a sequence of high-level functional contexts will be aggregated with respect to a a given goal. Then, we

will provide the algorithms for GAP aggregate matching.

5.2.1 High-Level Functional Context Aggregation

Not any accepted aggregate pattern could form an aggregate service, as the invocation of the component services must be guaranteed, and the describing-constraints of these component services must be consistent to form an aggregate set of describing-constraints, as indicated in Figure 5.11. It indicates that the user's pre-constraints must satisfy the pre-constraints of the first service in the sequence, and the pre-constraints of every component service must be satisfied by the persisting constraints at their preceding transition points (a transition point is the point of time between two consecutive services). Also, the describing-constraints of component services need to be aggregated and compared to the describing-constraints of the user.

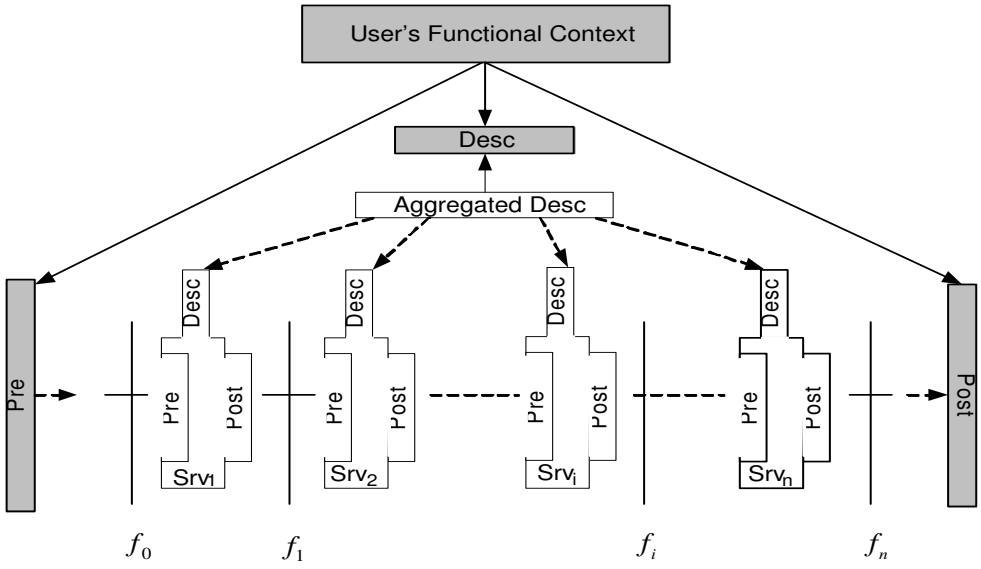

Figure 5.11: High-level Functional Context Aggregation

In general, when the describing-constraints of the component services are independent (that is they have unreachable scopes), the aggregated set of describing-constraints will be a simple union of the describing-constraints of the component services (that is $\bigcup_{i=1}^{n} GCtxt_i^{Desc}$). However, the describing-constraints of the component services could have con-

tradicting constraints and/or subsuming constraints, hence the aggregated set of describing-constraints cannot be determined via a simple union between the describing-constraints of the component services. To aggregate the describing-constraints, first we need to define the concept of the *weakest describing-constraint*.

Definition 38 (Weakest Describing-Constraint) *Given a set of describing-constraints with reachable scopes f_x, a constraint $Cnst_i$ and a goal G such that $Cnst_i \in f_x$. $Cnst_i$ is the weakest describing-constraint of f_x with respect to G if $\forall\ Cnst_j \in f_x$, $Cnst_j \trianglerighteq_G Cnst_i$.*

For example, the weakest constraint of the set { `Credit.Period` ≥ 30, `Credit.Period` ≥ 40, `Credit.Period` ≥ 60} is the constraint (`Credit.Period` ≥ 30).

In order to determine the aggregated set of describing-constraints, first we divide the union of the describing-constraints of the component services into a group of independent subsets, where every subset contains a group of describing-constraints that have reachable scopes. Then every subset will be reduced to its weakest constraint (if exists) such that the aggregated set of describing-constraints will be the union of these weakest constraints, as indicated in Figure 5.12.

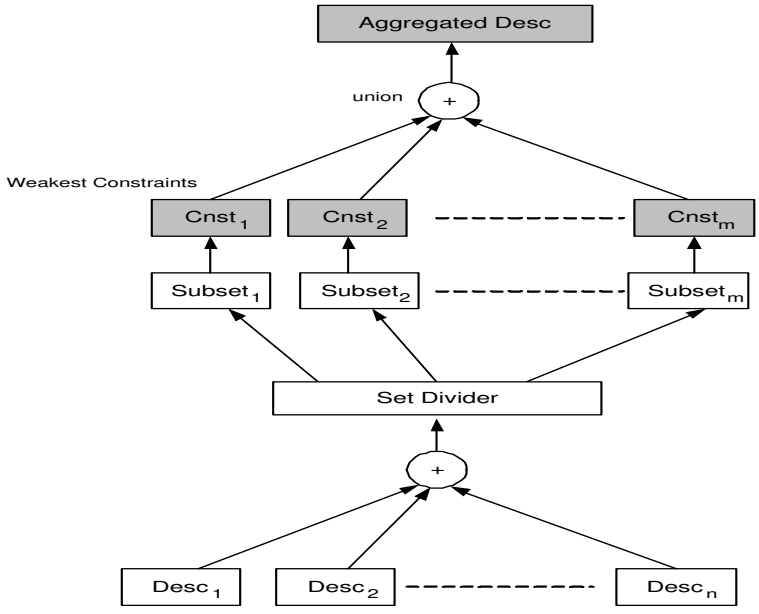

Figure 5.12: Describing-Constraints Aggregation

For example, the union of the following group of describing-constraints sets {`Credit.Period` $\geq$ 30, `Specialty.Type` = `Motor-vehicles`}, {`Credit.Period` $\geq$ 15, `Specialty.Type` $\subseteq$ {`Motor-vehicles`, `Food-Products`}} will be divided into the following independent two subsets {`Credit.Period` $\geq$ 15, `Credit.Period` $\geq$ 30} and {`Specialty.Type` = `Motor-vehicles`, `Specialty.Type` $\subseteq$ {`Motor-vehicles`, `Food-Products`}}.

Algorithm 6 Describing-Constraints Divider

Input: A group of describing-constraints $GCtxt_1^{Desc}$, $GCtxt_2^{Desc}$,...,$GCtxt_n^{Desc}$, and a goal G

Output: A group of independent subsets of describing-constraints with respect to G.

1: **Begin**
2: Create an array $X[]$ corresponding to the elements of $\bigcup_{i=1}^{n} GCtxt_i^{Desc}$.
3: Initialize $X[]$ elements as Unmarked.
4: **for** $i = 1$ to n **do**
5: **if** $X[i]$ is Unmarked **then**
6: Mark $X[i]$.
7: Create a subset containing $X[i]$.
8: **for** j = i+1 to n **do**
9: **if** $X[j]$ is Unmarked **then**
10: **if** $(X[i] \rightharpoonup_G X[j]) \vee (X[j] \rightharpoonup_G X[i]) \vee (X[i] \curvearrowright_G X[j]) \vee (X[j] \curvearrowright_G X[i])$ **then**
11: Mark $X[j]$.
12: Add $X[j]$ to $X[i]$ subset.
13: **end if**
14: **end if**
15: **end for**
16: **end if**
17: **end for**
18: Return the created subsets.
19: **End**

Algorithm 6 indicates how the union of the describing-constraints of component services will be divided into a group of independent subsets. It assigns any constraints with reachable scopes into the same subset. When a subset does not have a weakest constraint, this means its elements are contradicting, which implies that the describing-constraints of the component services cannot be aggregated. Hence, such aggregate pattern is considered as invalid and rejected. By having the aggregated describing-constraints formed from the weakest constraints, this guarantees that the aggregated describing-constraints are satisfied by the describing-constraints of component services. Algorithm 7 detects the existence of the weakest constraint of a given subset, by constructing a CMM from the elements of this subsets, in order to determine the substitutability status between its elements.

Algorithm 7 Describing-Constraints Aggregator

Input: A group of describing-constraints $GCtxt_1^{Desc}$,...,$GCtxt_n^{Desc}$, and a goal G

Output: An aggregated set of describing-constraints with respect to G if possible, False otherwise.

1: **Begin**
2: Apply Algorithm 6 over $GCtxt_1^{Desc}$,...,$GCtxt_n^{Desc}$, and G.
3: **for each** subset of the describing-constraints created subsets **do**
4: Construct a CMM, where the rows and the columns are the elements of the subset.
5: **if** the constructed CMM does not have a column of ones **then**
6: Return (False)
7: **else**
8: Mark the constraint corresponding to the column of ones as the weakest.
9: **end if**
10: **end for**
11: Construct a set $Desc_a$ as a union of all the marked weakest constraints.
12: Return ($Desc_a$)
13: **End**

For example, the following group of describing-constraints sets {`Credit.Period` $\geq$ 30, `Specialty.Type` = `Motor-vehicles`}, {`Credit.Period` $\geq$ 15, `Specialty.Type` $\subseteq$ {`Motor-vehicles`, `Food-Products`}}, are aggregated to {`Credit.Period` $\geq$ 15, `Specialty.Type` = `Motor-vehicles`}. The constraint (`Credit.Period` $\geq$ 15) is the weakest constraint of the subset {`Credit.Period` $\geq$ 15, `Credit.Period` $\geq$ 30}, as the corresponding CMM is as follows.

	Credit.Period $\geq$ 15	Credit.Period $\geq$ 30
Credit.Period $\geq$ 15	1	0
Credit.Period $\geq$ 30	1	1

Similarly, the constraint (`Specialty.Type` = `Motor-vehicles`) is the weakest constraint of the subset {`Specialty.Type` = `Motor-vehicles`, `Specialty.Type` $\subseteq$ {`Motor-vehicles`, `Food-Products`}}, as the corresponding CMM is as follows.

	Specialty.Type = Motor-vehicles	Specialty.Type $\subseteq$ {Motor-vehicles, Food-Products}
Specialty.Type = Motor-vehicles	1	0
Specialty.Type $\subseteq$ {Motor-vehicles, Food-Products}	1	1

When a constraint has a column of ones in the corresponding CMM, this means it can be substituted by all the other elements of the subset, hence it is considered the weakest constraint. But when there is no column of ones in a given CMM, this means the constraints of the subset are contradicting constraints, and the describing-constraints cannot be aggregated.

Theorem 14 provides the conditions must hold in order to aggregate a sequence of describing-constraints with respect to a given goal. That the union of the describing constraints can be divided into a group of independent subsets, and every subset can be reduced to its weakest constraint.

Theorem 14 (Goal-based Describing-Constraints Aggregation) *Given a sequence of services* $\langle Srv_1, Srv_2, .., Srv_n \rangle$ *and a goal* G *such that their goal achievement processes are described as* $(G_i \vdash \langle GCtxt_i^{Pre}, GCtxt_i^{Post}, \beta_i, GCtxt_i^{Desc} \rangle)$*, where* $1 \leq i \leq n$*. The describing-constraints of these services can be aggregated with respect to* G *(denoted as* $\biguplus_{i=1}^{n} GCtxt_i^{Desc}$*) if* $(\bigcup_{i=1}^{n} GCtxt_i^{Desc})$ *is divided into a group of independent subsets, and every subset has a weakest constraint.*

Proof: *As every component service guarantees the satisfaction of its describing-constraints, the describing-constraints of the aggregate pattern must be satisfied by the describing-constraints of component services. This is achieved when the aggregated describing-constraints is formed from the weakest constraints of the component services' describing-constraints.*

In what follows, we will use the notation $(\biguplus_{i=1}^{n} GCtxt_i^{Desc})$ to refer to the aggregated describing-constraints set. Theorem 15 indicates the criteria required for the high-level functional context aggregate matching. That the user's pre-constraints satisfy the pre-constraints of the first component service, the persisting constraints at the last transition point of the services sequence satisfy the user's post-constraints, and the aggregated describing constraints of the service sequence satisfy the user's describing constraints.

Theorem 15 (High-Level Functional Context Aggregate Matching) *Given a sequence of services* $\langle Srv_1, Srv_2, .., Srv_n \rangle$ *such that their goal achievement processes are described as* $(G_i \vdash \langle\ GCtxt_i^{Pre},\ GCtxt_i^{Post},\ \beta_i,\ GCtxt_i^{Desc} \rangle)$*, where* $1 \leq i \leq n$*, and a user request that is described by the goal achievement process* $(G_u \vdash \langle GCtxt_u^{Pre}, GCtxt_u^{Post},\ \beta_u,\ GCtxt_u^{Desc} \rangle)$*. The aggregate high-level functional context of the service sequence matches the high-level functional context of the user's request if* $(GCtxt_u^{Pre} \trianglerighteq_{G_u} GCtxt_1^{Pre}) \wedge (f_n \trianglerighteq_{G_u} GCtxt_u^{Post}) \wedge ((\biguplus_{i=1}^{n} GCtxt_i^{Desc}) \trianglerighteq_{G_u} GCtxt_u^{Desc})$*, where* $f_0 = GCtxt_u^{Pre}$*,* $f_i = ((f_{i-1} \diamond_G GCtxt_i^{Pre}) + GCtxt_i^{Post})$*, and* $(1 \leq i \leq n)$*.*

Proof: *In order to have a valid service sequence, the persisting constraints at every transition point must satisfy the pre-constraints of the following service, and the describing-constraints of the service sequence must be valid to be aggregated. As indicated in Figure 5.11,* $GCtxt_u^{Pre}$ *represents* f_0*. Hence,* $(GCtxt_u^{Pre} \trianglerighteq_{G_u} GCtxt_1^{Pre})$ *must hold in order to be able to invoke the aggregate service ...(1). The persisting constraints will be determined similarly as in Theorem 12, by replacing the operations by services. Assuming (1) holds, then* f_n *and* $(\biguplus_{i=1}^{n} GCtxt_i^{Desc})$ *are guaranteed to be satisfied. Therefore, in order to match the functional context of the user* $(f_n \trianglerighteq_{G_u} GCtxt_u^{Post}) \wedge ((\biguplus_{i=1}^{n} GCtxt_i^{Desc}) \trianglerighteq_{G_u} GCtxt_u^{Desc})$ *must hold in order to consider* G_u *achieved according to Definition 26.*

5.2.2 GAP Aggregate Matching

The GAP aggregate matching process starts by scanning the service registry for services with plug-in GAPs. The result of this scanning process is an array for stacks corresponding to the required behavior model, as indicated in Figure 5.9. The aggregation process starts by checking the chunks in the stack corresponding to the first target state S_0, this stack is known by *Stack-0*. For every chunk in stack-0, a new stack will be created containing all the chunks that starts with the value of its *End* field. The stack-0 is known as the *ancestral stack*, while the newly created stacks are known as the *descendent stacks* (see Figure 5.13).

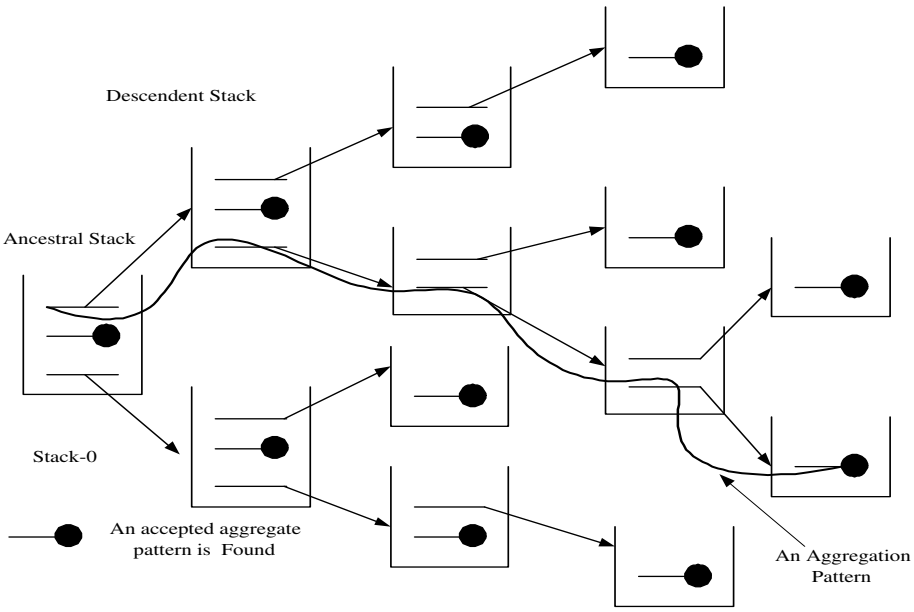

Figure 5.13: GAP Aggregation Approach

The descendent stacks will be examined to see if one of their chunks has covered the remaining target subsequence by checking if its *End* field contains the index of the last target state. If this case occurs, this means an accepted aggregate pattern is found, otherwise the scenario will be repeated by considering the descendent stack as an ancestral stack and generating the new descendent stacks. The pseudo code of the aggregation process is depicted in Algorithm 8. This algorithm is a recursive algorithm that takes the GAP request and the start and end indices to be examined. The algorithm uses *GetChunkStack* function

that returns the stack corresponding to *SeqStart* from the array of stacks representing the chunks list. The algorithm indicates that once an accepted aggregate pattern is found, the corresponding high-level functional context must match the request high-level functional context according to Theorem 15, otherwise the aggregate pattern will be considered as an invalid pattern. The recursive call of the algorithm uses (Chunk.End) as the SeqStart instead of (Chunk.End+1). This is because there exists a common state between any two consecutive chunks that must be taken into consideration. That, the state index represented by Chunk.End appears as the first state in the successor chunk, therefore the call with Chunk.End guarantees no state transitions will be missed. For example, Service1 and Service2 are part of a solution for the request indicated in Example 4, Figure 5.14 illustrates that the corresponding state sequences have a common state ($\langle\{b\},\{\}\rangle$) which is the state of the transition point between service1 and service2. Example 4 illustrates how Algorithm 8 is applied.

Example 4 (GAP Aggregation) *Figure 5.14 depicts the GAP request and the retrieved plug-in GAPs and their corresponding state sequences. Every stack corresponds to a state index. The corresponding chunk list in the array of stacks format will be as follows:*

Stack 0 *contains* $Chunk_1 = \langle$ *Service1,0,1*$\rangle$, $Chunk_4 = \langle$ *Service4,0,2*$\rangle$.

Stack 1 *contains* $Chunk_2 = \langle$ *Service2,1,3*$\rangle$, $Chunk_6 = \langle$ *Service6,1,4*$\rangle$.

Stack 2 *contains* $Chunk_5 = \langle$ *Service5,2,4*$\rangle$.

Stack 3 *contains* $Chunk_3 = \langle$ *Service3,3,4*$\rangle$.

Stack 4 *is empty.*

Algorithm 8 GAP Aggregator

Input: SeqStart, SeqEnd, Request

Output: True when solution list is found, False otherwise.

```
1: Begin
2: if (SeqStart= 0) then
3:    Create a new Aggregate Pattern
4: end if
5: ChunkStack = GetChunkStack(SeqStart)
6: if (ChunkStack.Size = 0) then
7:    Return (False)
8: else
9:    for each Chunk in the ChunkStack do
10:      Add Chunk.Service-ID to the Aggregate Pattern
11:      if (Chunk.End = SeqEnd) then
12:         if (HLFC of the aggregate pattern matches HLFC of the request) then
13:            Add the Aggregate Pattern to the solutions list
14:            Return(True)
15:         end if
16:      else
17:         Apply Algorithm 8 over (Chunk.End, SeqEnd,Request)
18:      end if
19:   end for
20: end if
21: End
```

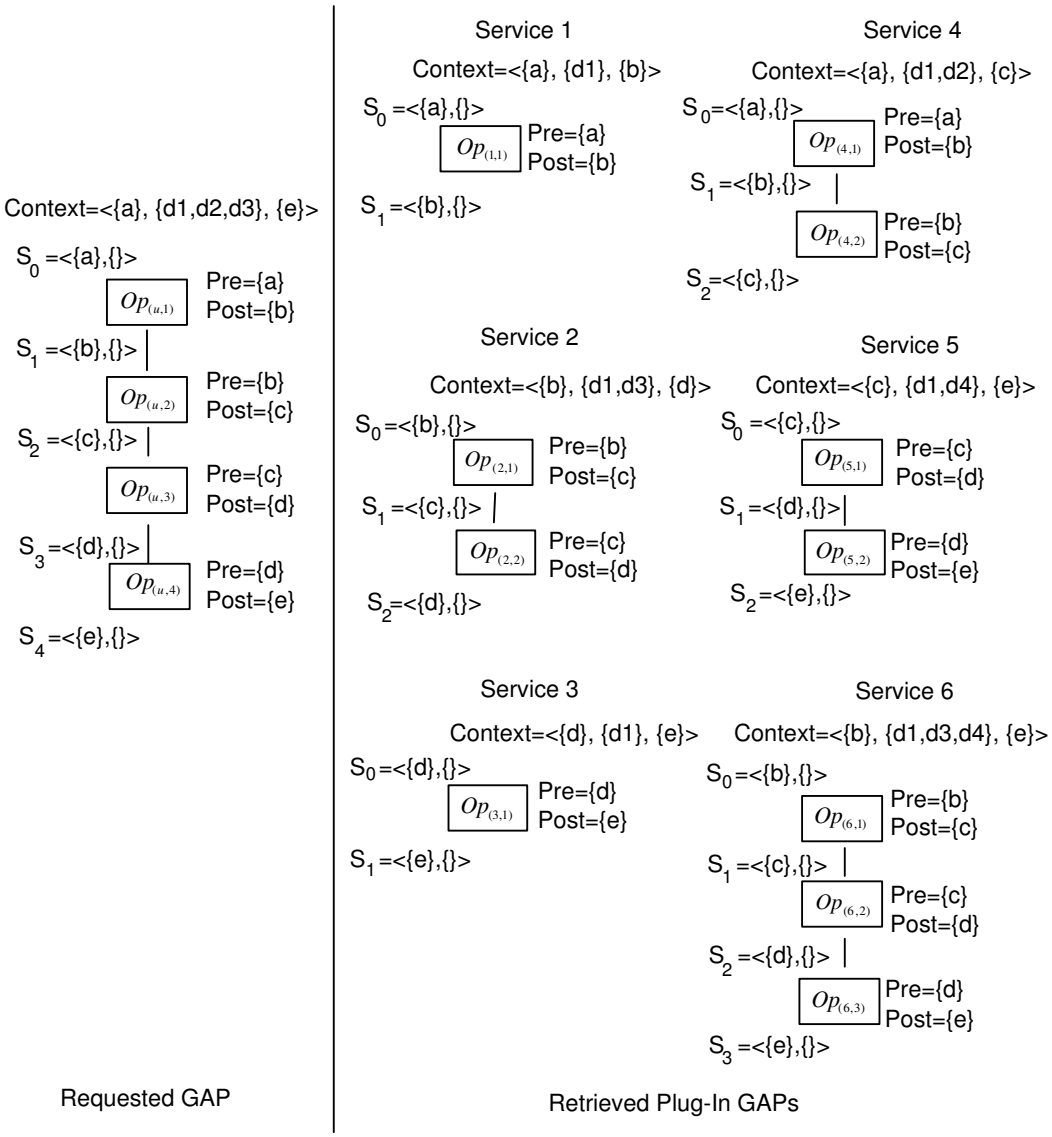

Figure 5.14: A Request GAP and its corresponding Plug-in GAPs

Algorithm 8 is initially invoked with (0,4,request), which constructs a stack from the stack 0 contents: $Chunk_1$, $Chunk_4$. Starting by the first chunk $Chunk_1$, the algorithm finds the pattern (Service1) does not cover all the required sequence, hence it makes a recursive call with (1,4, request). This call will construct a stack from stack 1 contents: $Chunk_2$, $Chunk_6$.

Starting by $Chunk_2$ the algorithms finds that the pattern (Service1-Service2) does not cover all the required sequence, hence it makes a recursive call (3,4,request). This call will construct a stack from stack 3 contents: $Chunk_3$, the algorithm finds that the pattern (Service1-Service2-Service3) fulfils the request so that it returns the sequence as a solution. Then it backtracks to the predecessor ancestral $Chunk_2$; as there is no more descendants for $Chunk_2$. The algorithm processes the next available chunk in the stack: $Chunk_6$; the corresponding aggregation pattern at this chunk is (Service1). The algorithm finds the pattern (Service1-Service6) fulfils the sequence, so it returns it as a solution, then backtracks to the ancestral $Chunk_1$, as no more descendants for this chunk, the algorithm moves to $Chunk_4$ as it is the next available chunk in the stack, the algorithms finds the pattern (Service4) does not cover all the required sequence, hence it makes a recursive call with (2,4,request). This call will construct a stack from stack 2 contents: $Chunk_5$, starting by $Chunk_5$, the algorithm will find that (Service4-Service5) fulfills the request so it reports it as a solution, then it terminates reporting the following three solutions, as depicted in Figure 5.15:

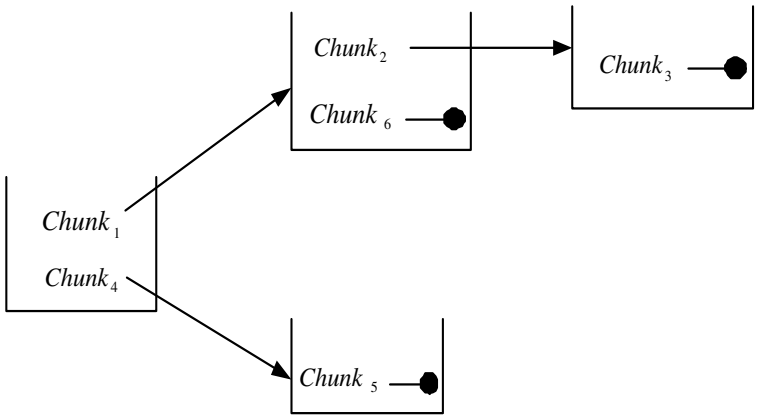

Figure 5.15: Aggregation Process Trace

1. *Service1-Service6.*
2. *Service1-Service2-Service3.*
3. *Service4-Service5.*

Assuming $d1, d2, d3, d4$ *are independent, the aggregate set of describing-constraints will be* $\{d1, d2, d3, d4\}$*, which satisfies the required set of describing-constraints* $\{d1, d2, d3\}$*.*

The complexity of the aggregation algorithm is analyzed by Theorem 16. The complexity of the aggregation process is high, however this was expected as Berardi et al. [2003] proved that the lower bound for composition of services expressed as finite state machines is decidable in EXPTIME. However, we believe our algorithms could be more optimized, but this is out of the scope of this thesis.

Theorem 16 (GAP Aggregation Complexity) *Let m is the number of states in the required behavior model, n is the number of retrieved chunks for every chunk stack. The worst case complexity of GAP aggregation to find a solution is* $O(n^m)$*.*

Proof: *The worst case scenario is that every state in the required behavior is matched only to one chunk in the corresponding. Hence the algorithm must make m recursive calls to find a solution. As every chunk stack has n possible chunks, hence every recursive call costs* $O(n)$*. Therefore the worst case total cost of the algorithm will be* $O(n^m)$*.*

After obtaining the aggregate solutions, they will be filtered according to the aggregate NTC, NFC, and LLFC, using the same rules indicated in Section 5.1. The aggregate NTC will contain the aggregate price, which is the summation of all component services' prices. The aggregate LLFC will contain the common device types supported by all component services, while the required aggregate bandwidth will be the maximum value of the component services' bandwidth values. The aggregate NFC will contain the aggregate reliability (which is computed as the minimum value of the component services' reliability values), the aggregate availability (which is computed as the minimum value of the component services' availability values), and the aggregate reputation (which is computed as the minimum value of the component services' reputation values).

5.2.3 GAP Forest Aggregation

In forest aggregate matching, the solution could be formed from different abstraction levels of the component services. In other words, it is not necessary that when a given request is represented in a given abstraction level, the aggregation pattern should be formed from GAPs represented in the same abstraction level. Therefore, in order to find the aggregate pattern for a given request GAP represented in a given abstraction level, all abstraction levels of services' GAPs will be examined, as every GAP could be a potential plug-in of the request. Hence, when examining the matching cases for a given request GAP-forest, every abstraction level of the GAP forest will have it own chunk list, that will submitted to the aggregation algorithm in order to find the corresponding list of solutions, as indicated in Figure 5.16. It indicates that for every GAP in the forest, a separate chunk-list will be generated by checking all the GAPs in the service registry.

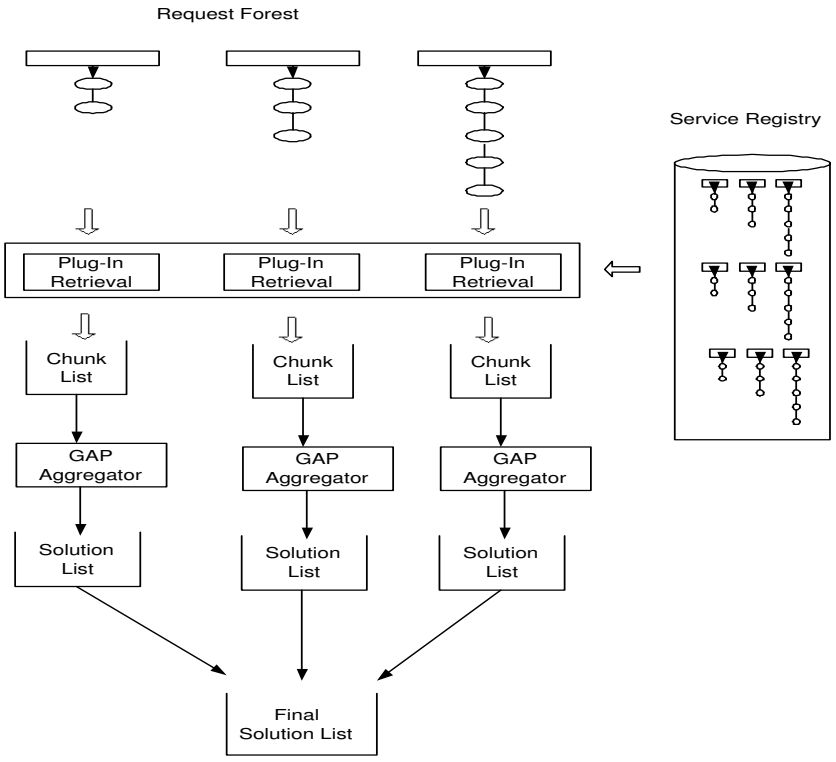

Figure 5.16: GAP Forest Aggregation

Algorithm 8 will be applied on every chunk-list resulting a list of solutions. The generated lists of solutions will form one final solution list. Algorithm 9 depicts the pseudo code of GAP-forest Aggregation. After obtaining the aggregate solutions, they will be filtered according to the aggregate NTC, NFC, and LLFC.

Algorithm 9 GAP-Forest Aggregation

Input: Request

Output: List All possible solutions

1: **Begin**
2: **for each** GAP in the Request GAP-Forest **do**
3: Create the corresponding Chunk-list
4: Apply Algorithm 8 on the created list.
5: **end for**
6: Collect all the generated solutions in one list
7: **End**

5.3 Service Matching Techniques Evaluation

As the devised matching techniques are semantic high-level functional matching techniques, this section evaluates the devised service matching techniques against the corresponding service matching approaches of the classification indicated in Table 2.1 (page 28). We use the F-measure as the comparison criterion between the matching approaches as 100% F-measure implies the matching results are totally correct. F-measure is calculated as $\frac{2 \times Precision \times Recall}{Recall + Precision}$, where Precision is calculated as the ratio $\frac{NumOfRetrievedCorrectAnswers}{NumOfRetrievedAnswers}$, and recall is calculated as the ratio $\frac{NumOfRetrievedCorrectAnswers}{NumOfCorrectAnswers}$.

As the devised service direct matching approach is a semantic high-level functional-based service matching approach, first we will show that it provides better F-measure values when compared to the unstructured-based matching approaches (syntactic and semantic) as well as syntactic high-level functional approaches. Second, we will show that when adopting m-to-n state matching approach, better F-measure values are obtained when compared to the

one-to-one state matching approach (syntactic and semantic). Finally, we will show that the devised aggregate matching technique provides better F-measure values when compared to the devised service direct matching technique, which indicates that the aggregate service matching technique can find solutions for requests that the direct service matching technique could not.

We adopt a basic implementation for realizing the service matching approaches of the proposed classification during the evaluation process. For example, a simple keyword-based matching technique is adopted to represent the syntactic unstructured service matching approach. Unfortunately, there are neither benchmarks nor standard data sets for semantic web services matching. Even the semantic data set for web services provided by WS-Challenge [2005] contains only abstract WSCD descriptions for WSDL interfaces, which does not contain any high-level functional specifications that could be adopted in our experiments [Tatemura et al., 2005]. Furthermore, currently there are no ontologies that can capture the substitution semantics of application domains. Hence we opted to use a simulation approach, where random data set and queries are generated to validate the devised matching techniques.

Work-Load Generation: The proposed techniques require the existence of a domain ontology that adopts the meta-ontology structure. The elements of the conducted experiments are: a domain ontology (that includes concepts, operations and concepts substitutability graph); a data set of services' G^+ models; and a query set and its correct answers. We have generated a random number of concepts and a random number of operations to represent the domain elements. In order to make sure there are no contradicting constraints during operations generation, the following restrictions are required when generating the domain operations:

- Every operation has one distinct concept as input parameter and same input concept will be the operation output.
- Every operation has one comparative pre-constraint over an attribute of the input concept such that the value of this attribute equals its lower limit.

- Every operation has one comparative post-constraint over the same attribute used in the pre-constraints such that the value of this attribute less than its upper limit.

Distinct random sets of operations are generated using the generated operations. This is a mandatory requirement in order to avoid appearance of multiple answers for a given query during the experiments. From these distinct sets of operations we will generate a sequence of operations. The union of the pre-constraints of the operations will be the pre-constraints of the functional context, similarly the union of the post-constraints of the operations will be the post-constraints of the functional context. For every generated sequence of operations and its corresponding functional context we generate a random number of new equality constraints based on a newly generated number of concepts to have the describing-constraints of the functional context. Finally, for every generated sequence of operations and its corresponding functional context we generate a new operation to represent the required goal. This represents a data set of G^+ models, in which every G^+ model has only one GAP.

Experiments Plot: A distinct data set of G^+ models will be generated from the generated ontology as indicated above. The query set is constructed by randomly selecting 10% of the data set, hence the correct answers of the queries are known a priori, every query will have only one answer as the generated data set are generated from a distinct sets of operations. This query set will be mutated such that every experiment applies a different mutation process. Ten mutated query sets are produced such that the first mutated query set has the first 10% of the query set being mutated, the second mutated query set has the first 20% of the query set being mutated, and so on until the tenth mutated query set has 100% mutated queries. Later, the original query set and the mutated query sets are applied to the matchmaker. The concepts substitutability graph will be generated during the mutation processes (details will be given later). The F-measure metric is computed for every query set. The above procedure is repeated 1000 times and the final average result is computed.

The matching algorithms are implemented using a basic sequential search and the satisfiability checker is implemented using enumeration for the attribute domain values (which is restricted to a small range since the attributes range does not affect the values of the retrieval metrics). Figure 5.17 summarizes how the simulation experiments will be performed.

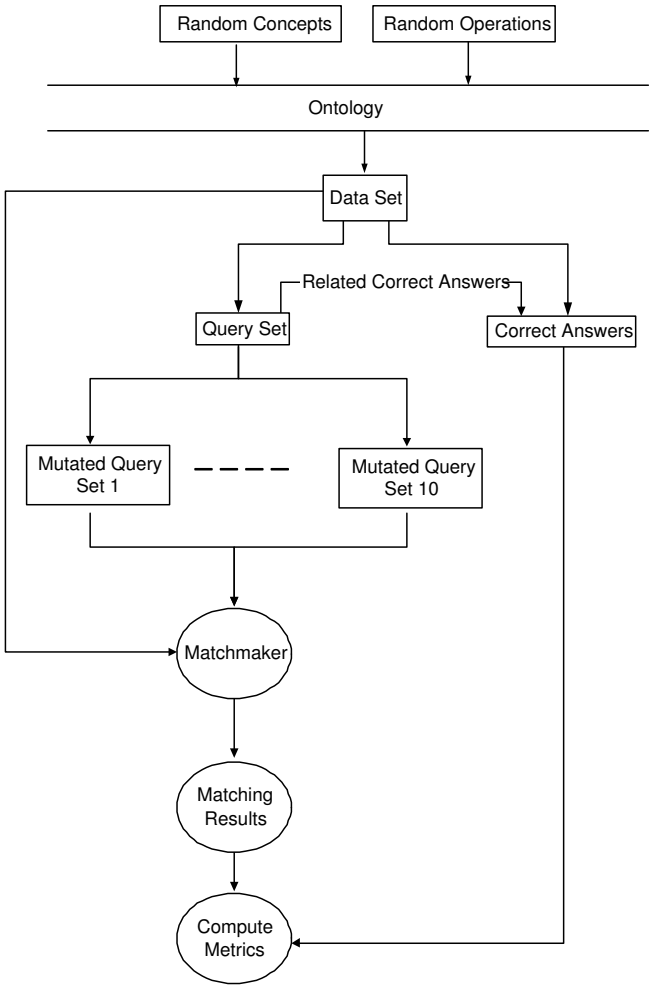

Figure 5.17: Simulation Experiments Plot

5.3.1 Service Direct Matching

In this experiment we compare between the devised direct service matching technique indicated in Theorem 13 and the imprecise service matching approaches that represent the unstructured matching approaches (syntactic and semantic) and a more restricted version of the technique indicated in Theorem 13 to represent the syntactic high-level functional-based matching approach. This restricted technique will use only direct satisfiability during constraint matching and adopts a one-to-one state matching approach when matching the operation sequences. First we introduce the imprecise matching approaches.

Imprecise Service Matching Approaches

The unstructured syntactic matching approach takes into consideration neither the structure nor the semantics of services during the matching process. Keyword-based matching is an example of this approach. To resemble keyword-based matching adopting our functional context model, we define the concept of *Context Flat Scope* (CFS) that is defined as indicated in Definition 39.

Definition 39 (Context Flat Scope) *Given a high-level functional context* $Ctxt_i = \langle$ *Pre, Descr, Post* $\rangle$*, the context flat scope of* $Ctxt_i$ *(denoted as CFS(*$Ctxt_i$*)) is defined as* $\Xi(Ctxt_i^{Pre}) \cup \Xi(Ctxt_i^{Descr}) \cup \Xi(Ctxt_i^{Post})$.

The context flat scope is a set of all the scopes of the constraints that belong to pre-constraints set, describing-constraints set and post-constraints set. The elements of CFS will be considered keywords to be used in the matching process. Thus, a service will be represented by the CFS of its high-level functional context (denoted as CFS_s) and the request will be represented by the CFS of its high-level functional context (denoted as CFS_u). Service matching is determined simply by comparing the elements of CFS_s and CFS_u. Any set classical coefficient such as Jaccard's Coefficient (JC) [Ganesan et al., 2003] could be used to compute the proximity between the two sets. In our implementation, we calculated the proximity as $\frac{\|CFS_s \cap CFS_u\|}{\|CFS_u\|}$, which is a variation of the original Jaccard's Coefficient that computed as $\frac{\|CFS_s \cap CFS_u\|}{\|CFS_S \cup CFS_u\|}$. This is because the original Jaccard's Coefficient is symmetric while our variation is asymmetric, which resembles the asymmetric nature of goal achievement operator. A service is considered a match for a goal when the proximity ratio equals to 1. This matching approach is known to be *Matching via Syntactic CFS*. Such matching approach leads to the appearance of both false negatives and false positives as it ignore the semantics of services, users and application domains. We will show that just by taking structure information into consideration a considerable improvement of the F-measure value will be obtained. To realize this approach, we define the concept of *Context Structured Scope* (CSS) that is defined as indicated in Definition 39.

Definition 40 (Context Structured Scope) *Given a high-level functional context* $Ctxt_i$ = ⟨ *Pre, Descr, Post* ⟩, *the context structured scope of* $Ctxt_i$ *(denoted as CSS(*$Ctxt_i$*)) is defined as* ⟨ $\Xi(Ctxt_i^{Pre})$, $\Xi(Ctxt_i^{Descr})$, $\Xi(Ctxt_i^{Post})$ ⟩.

The context structure scope is a tuple of the scopes of the pre-constraints set, describing-constraints set and post-constraints set. Thus, a service will be represented by the CSS of its high-level functional context (denoted as CSS_s) and the request will be represented by the CSS of its high-level functional context (denoted as CSS_u). Service matching is determined simply by comparing the elements of CSS_s and CSS_g. The proximity between the tuples is computed as $\frac{1}{3}\times$ ($\frac{\|\Xi(Ctxt_s^{Pre})\cap\Xi(Ctxt_u^{Pre})\|}{\|\Xi(Ctxt_u^{Pre})\|}$ + $\frac{\|\Xi(Ctxt_s^{Desc})\cap\Xi(Ctxt_u^{Desc})\|}{\|\Xi(Ctxt_u^{Desc})\|}$ + $\frac{\|\Xi(Ctxt_s^{Post})\cap\Xi(Ctxt_u^{Post})\|}{\|\Xi(Ctxt_u^{Post})\|}$). A service is considered a match for a goal when the proximity ratio equals to 1. This matching approach is known to be *Matching via Syntactic CSS*. Such matching approach leads to the appearance of both false negatives and false positives as well, because it ignores the semantics of services, users and application domains. We will show by adopting the semantics of the application domain as well as the structure information considerable improvement of the F-measure value will be obtained. This could be realized by checking the reachability between the scopes of the CSS elements instead of syntactically comparing them. Hence, the proximity between the tuples is computed as $\frac{1}{3}\times$ ($\frac{\|\Xi(Ctxt_s^{Pre})\twoheadrightarrow_g\Xi(Ctxt_u^{Pre})\|}{\|\Xi(Ctxt_u^{Pre})\|}$ + $\frac{\|\Xi(Ctxt_s^{Desc})\twoheadrightarrow_g\Xi(Ctxt_u^{Desc})\|}{\|\Xi(Ctxt_u^{Desc})\|}$ + $\frac{\|\Xi(Ctxt_s^{Post})\twoheadrightarrow_g\Xi(Ctxt_u^{Post})\|}{\|\Xi(Ctxt_u^{Post})\|}$), where $\|\ \Xi(Ctxt_s^{Ppre})\twoheadrightarrow_g \Xi(Ctxt_u^{Pre})\ \|$ is the number of scopes in $\Xi_g(Pre)$ that are reachable (directly or indirectly) to scopes in $\Xi_s(Pre)$, similarly the other ratios are computed. A service is considered a match for a goal when the proximity ratio equals to 1. This matching approach is known to be *Matching via Semantic CSS*.

So the comparison will be between the following approaches:

1. Matching via Transformation: This adopts models' structure information, the semantics of services, users and application domain during the matching process, such as the one indicated in Theorem 13.

2. Matching via Implication: This adopts models' structure information, the semantics of services and users during the matching process, and ignores the semantics of application

domain. This is a restrictive version of Theorem 13, where constraints are matched via direct satisfiability, and one-to-one state sequence approach is adopted.

3. Matching via Semantic CSS: This adopts models' structure information, and the semantics of application domain during the matching process.

4. Matching via Syntactic CSS: This adopts only models' structure information during the matching process.

5. Matching via Syntactic CFS: This adopts neither models' structure information nor any semantics during the matching process.

The aim of the comparison is to show that the more information and more semantics about services, users and application domains are used during the matching process, the more precise matching results could be obtained. Unlike other matching approaches, matching via transformation approach uses all available information and semantics during the matching process, thus it is capable of retrieving only the correct results.

Mutation Process: Query mutation is performed as follows: the high-level functional context of a query element will be replaced by another different but substitutable high-level functional context. The new substitutable high-level functional context is generated by replacing every constraint in its sets by another constraint that has a distinct new scope, the same comparative operator, and the same value. An entry in the corresponding substitutability graph is generated between the old scope and the new scope with respect to the operation attached to the functional context of the query element, having values equality as the conversion function, and has no substitution constraints. For example, constraint $x = 10$, will be replaced by constraint $y = 10$ such that an entry in the substitutability graph for x and y will be created using x=y as the conversion function, and an OMM that maps = to =. By doing so, constraints matching via direct satisfiability, matching via syntactic CSS, matching via syntactic CFS will fail to identify the newly generated constraints, whereas constraints matching via transformation and matching via semantic CSS succeed to recognize the substitutability between the old and new scopes. Results of the experiment are depicted in Figure 5.18.

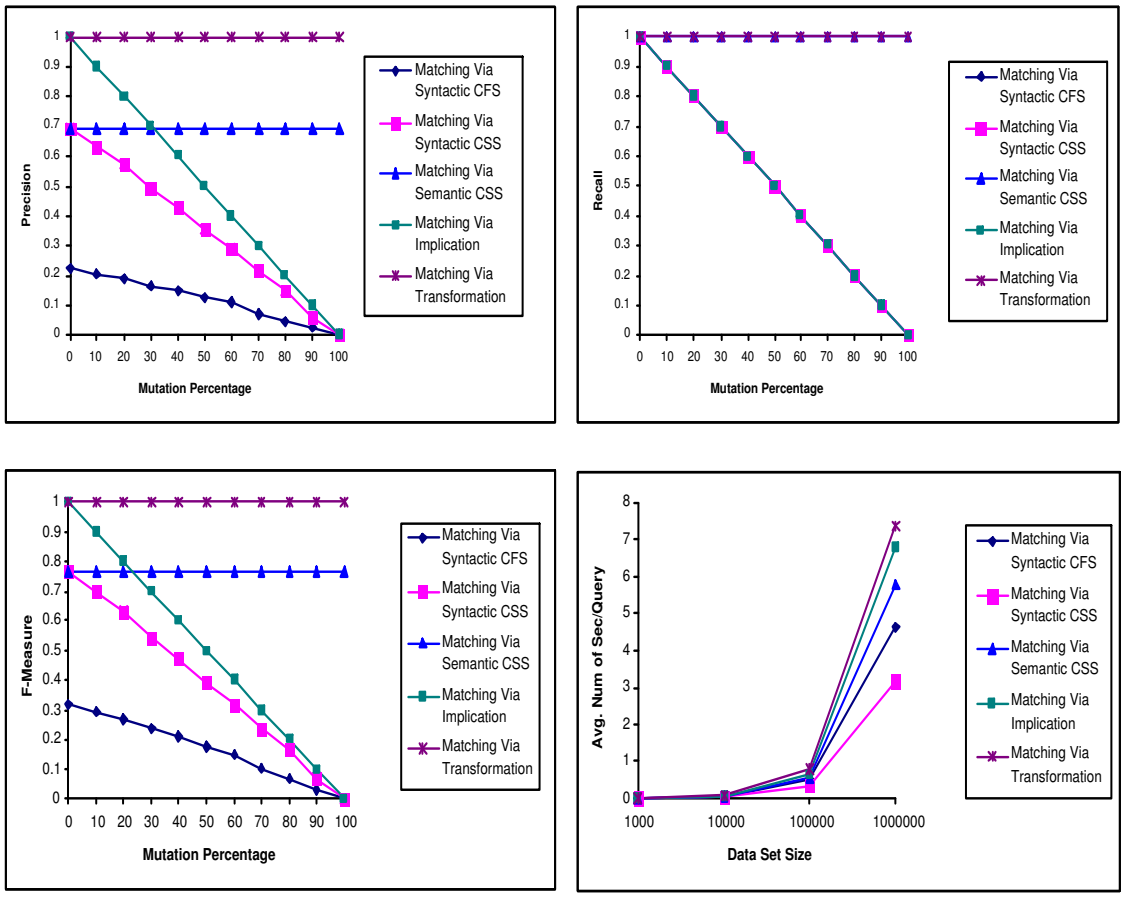

Figure 5.18: Simulated Comparison between Service Direct Matching Approaches

Results indicate that as the mutation percentage increases syntactical approaches fails to identify the new scopes, which increases the number of false negatives, which is negatively reflected on the precision and recall metrics. Before applying any mutations, the imprecise matching approaches failed to provide 100% precision due to the existence of false positives. When the false negatives start to appear due to mutations, the precision of imprecise approaches start to decline more, except the case of matching via semantic CSS. Matching via Semantic CSS succeeds in eliminating the false negatives as it adopts application domain semantic but failed to eliminate the false positives as no user or service semantics are adopted. On the contrary, matching via implication succeeded to eliminate the false positives but failed to eliminate the false negatives. Also all the semantic approaches have 100% recall,

as adopting semantics during matching eliminates the effects of the mutations, hence they succeed to eliminate the false negatives, but not all of them succeed to eliminate the false positives (reflected in the precision metric), as the only approach that succeeds to eliminate both of false negatives and false positives is matching via transformation.

The average query times curves indicated in the figure are computed in order to give the reader an intuition about the performance of the approaches. The more information and semantics that are adopted during the matching process, the more time required to obtain the results which is expected as more processing is required. As one can notice, the time complexity of the syntactic imprecise approach using CSS is less than the time complexity of the syntactic imprecise approach using CFS. This is because matching using syntactic CSS require less number of comparisons than matching using CFS. Matching via syntactic CSS requires ($||\Xi(Ctxt_s^{Pre})|| \times ||\Xi(Ctxt_u^{Pre})|| + ||\Xi(Ctxt_s^{Desc})|| \times ||\Xi(Ctxt_u^{Desc})|| + ||\Xi(Ctxt_s^{Post})|| \times ||\Xi(Ctxt_u^{Post})||$) comparisons, while matching via syntactic CFS requires ($||\Xi(Ctxt_s^{Pre})|| + ||\Xi(Ctxt_s^{Desc})|| + ||\Xi(Ctxt_s^{Post})||) \times (||\Xi(Ctxt_u^{Pre})|| + ||\Xi(Ctxt_u^{Desc})|| + ||\Xi(Ctxt_u^{Post})||$) comparisons.

5.3.2 Behavior Models Direct Matching

This section provides an evaluation for the devised m-to-n state matching approach. It compares it with the semantic one-to-one state matching approach indicated in Theorem 7, and the syntactic one-to-one operation sequences matching. The syntactic one-to-one operation sequences matching is determined as indicated in Proposition 13, where the operator $\| OpSeq \|$ determines the number of operations of $OpSeq$.

Proposition 13 (Syntactic Operation Sequence Matching) *Given two operation sequences $OpSeq_i$ and $OpSeq_j$, $OpSeq_j$ matches $OpSeq_i$ if $\|OpSeq_i\| = \|OpSeq_j\|$ and $\forall \langle Op_x, Op_{x+1} \rangle \in OpSeq_j$, $\exists \langle Op_y, Op_{y+1} \rangle \in OpSeq_i$ such that $(Op_y = Op_x) \wedge (Op_{y+1} = Op_{x+1})$.*

Proof: *This proposition matches the operation sequences when the are identical.*

Matching adopting Proposition 13, checks first the length of the source and target operation sequences before checking operations' signatures, that if the two sequences have different lengths, this implies they will not be matched.

Mutation Process: The mutation process is performed by merging all the operations of a given sequence into one operation such that the new mutated operation sequence will have only one operation. The new mutated operation is constructed such that:

- Its input is a collection of the sequence operations' inputs.
- Its output is a collection of the sequence operations' outputs.
- Its pre-constraint is a conjunction of the sequence operations' pre-constraints.
- Its post-constraint is a conjunction of the sequence operations' post-constraints.
- Replacing its input and output concepts by new substitutable concepts, where these concepts as well as the concepts substitutability graph will be created as indicated in last experiment. Hence, its pre and post-constraints will be replaced by new substitutable constraints.

For example, given a GAP with the operation sequence Op_1, Op_2, Op_3 such that:

- $Op_1(C_1) : C_1$, $Op_2(C_2) : C_2$, $Op_3(C_3) : C_3$.
- Op_1^{Pre} is $\{C_1.t = lowerLimit(t)\}$, Op_2^{Pre} is $\{C_2.v = lowerLimit(v)\}$, Op_3^{Pre} is $\{C_3.r = lowerLimit(r)\}$.
- Op_1^{Post} is $\{C_1.t < UpperLimit(t)\}$, Op_2^{Post} is $\{C_2.v < UpperLimit(v)\}$, Op_3^{Post} is $\{C_3.r < UpperLimit(r)\}$.

The new constructed operation Op_4 will be defined as follows:

- $Op_4(C_1, C_2, C_3) : (C_1, C_2, C_3)$.
- Op_4^{Pre} is $\{C_1.t = lowerLimit(t), C_2.v = lowerLimit(v), C_3.r = lowerLimit(r)\}$.
- Op_4^{Post} is $\{C_1.t < UpperLimit(t), C_2.v < UpperLimit(v), C_3.r < UpperLimit(r)\}$.

Having the attributes of C_1, C_2, C_3 to be substitutable to the attributes of C_4, C_5, C_6, respectively. Then the operation will be mutated to:

- $Op_4(C_4, C_5, C_6) : (C_4, C_5, C_6)$.
- Op_4^{Pre} is $\{C_4.t = lowerLimit(t), C_5.v = lowerLimit(v), C_6.r = lowerLimit(r)\}$.
- Op_4^{Post} is $\{C_4.t < UpperLimit(t), C_5.v < UpperLimit(v), C_6.r < UpperLimit(r)\}$.

As the original query set and the mutated query sets have the same answers, one-to-one matching approaches fail to answer the mutated queries. We have indicated this by showing how the retrieval metrics decreases as the percentage of mutated queries increases.

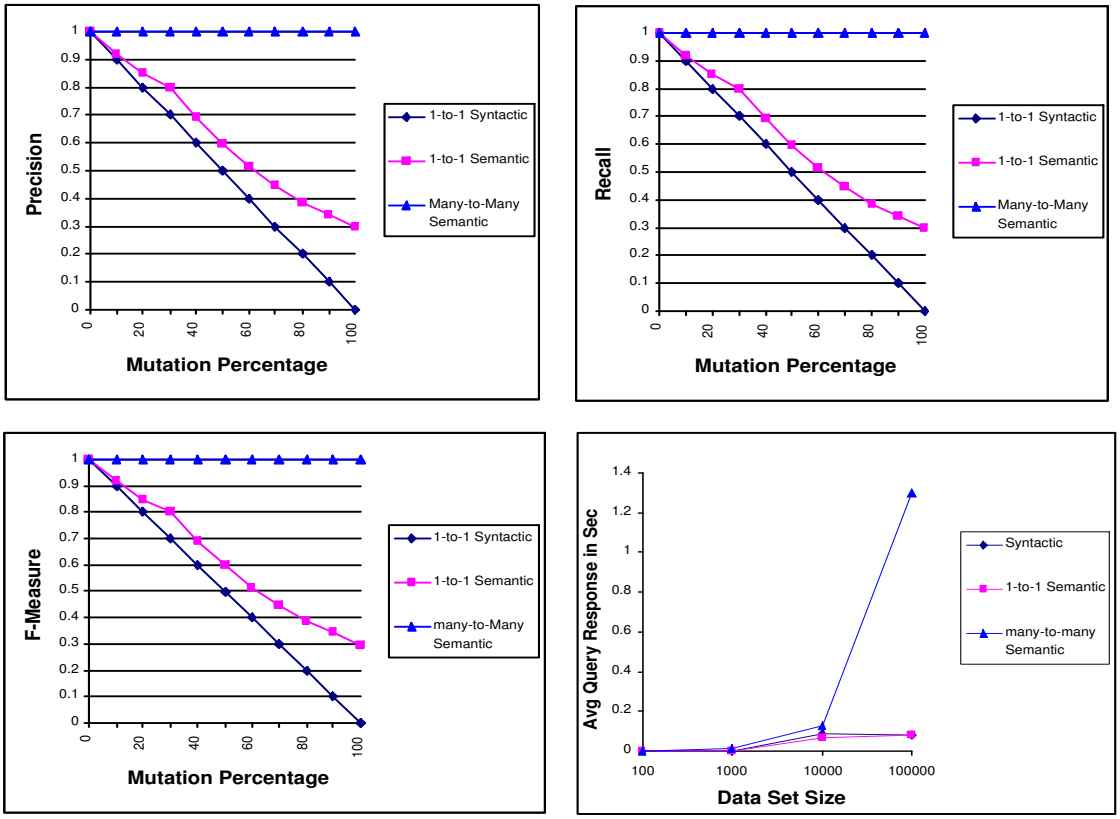

Figure 5.19: Comparison between Operations Sequence Matching Approaches

Figure 5.19 indicates that adopting the m-to-n state matching approach succeeds in obtaining the correct results in spite of the mutations, while other approaches fail to do so. Adopting one-to-one semantic matching could not obtain better matching results than the syntactic approach, except in the cases that have only one operation in the sequence, as it

succeeds to match the new corresponding states, while syntactical approach could not match the new signatures (this is seen as the slight increase on the retrieval precision and recall for the one-to-one semantic approach). Therefore, we argue only semantic state matching cannot prevent the appearance of false negatives unless m-to-n matching approach is adopted. In this experiment, one-to-one matching approaches have similar response times. This is because sequences are filtered first according to their length before checking the matching of individual operations/states. In spite of the higher response time of the m-to-n approach, we argue that it is yet still acceptable.

5.3.3 Service Aggregate Matching

This section provides an evaluation for the G^+ aggregate matching approach against the G^+ direct matching approach using the F-measure metric. Both approaches are full fledged semantic approaches (that is they use the semantics of services, users and application domains).

Mutation Process: The randomly selected elements of the query set to be mutated will be used to form a sequence of G^+ models. The corresponding aggregate context will be generated and matched according to Theorem 15. A new operation will be used to represent the goal of this aggregated G^+ model. This aggregated G^+ model is the new query to be submitted. The answer of this query is guaranteed to exist, as the answers for the component G^+ models already exist. Experiment results are shown in Figure 5.20.

Due to this mutation procedure, the direct matching approach will fail to find a solution for the newly generated complex G^+ model while the aggregate approach will be able to find a solution, which is the sequence of simple G^+ models used to form the complex model. As the original query set and the mutated query sets have the same answers, the direct approach fails to answer the mutated queries. We have indicated this by showing how the retrieval metrics decrease as the percentage of mutated queries increases. However, the average response time noticeably increased due to aggregate matching.

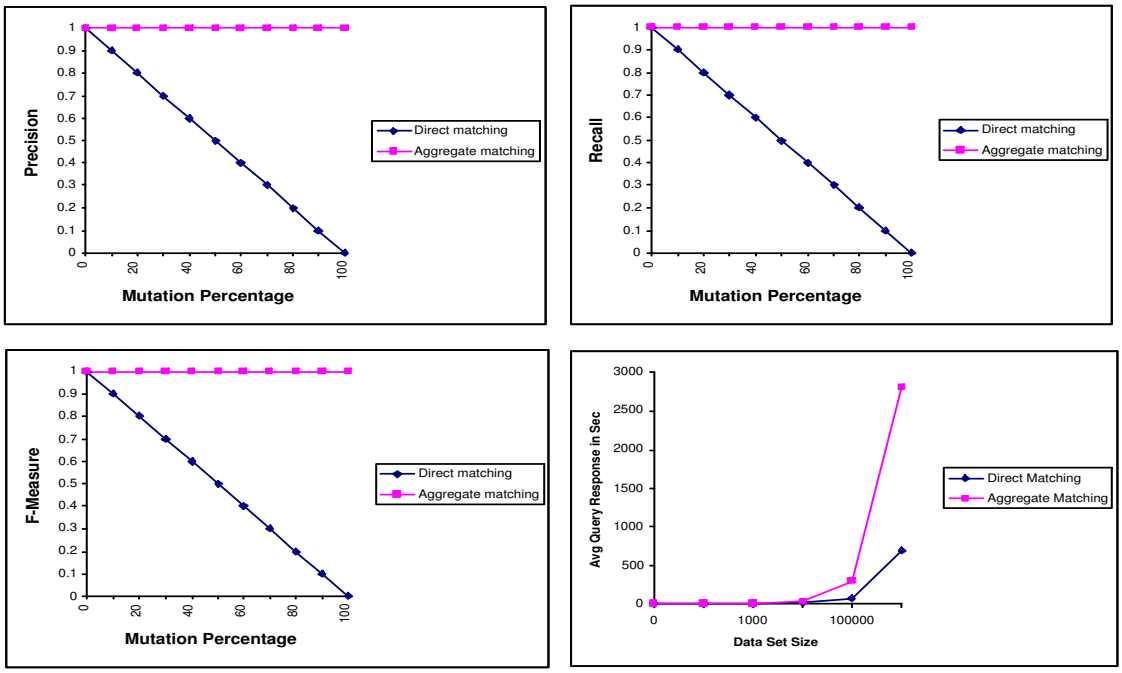

Figure 5.20: Direct versus Aggregate Semantic Matching

5.4 Conclusion

In this chapter, we proposed two new correctness-aware matching techniques for semantic web services adopting FSMS (direct and aggregate). First, we proposed the service direct matching technique, in which services are matched according to the substitutability of their G^+ models. Second, we proposed the service aggregate matching technique. We proposed the criteria for correct high-level functional context aggregation as well as the criteria for detecting plug-in GAPs. Finally, we evaluated the devised matching techniques using simulation experiments. We used the F-measure as the comparison aspect against the major existing service matching approaches. Simulation results showed that all the results retrieved by the devised matching techniques were correct answers, while existing matching approaches could not eliminate the appearance of false positives and false negatives.

Chapter 6

Conclusion

In this thesis, we investigated the existing obstacles to fully automate the service matching process, providing a number of integrated solutions to overcome such obstacles. We argued that to automate the service matching process, the matchmaker must be able to determine the correctness of the matching results with respect to the defined users' goals. Hence, capturing the semantics of services, users, and application domains in a machine-understandable format becomes a necessity to enable the matchmaker to process such semantics. We emphasized the importance of adopting formal matching schemes during the matching process, as this provides the matchmaker with the techniques required for determining the correctness of the matching results. Also we argued that the service matching process should be primarily based on the high-level functional specifications (roles, goals, contexts, and external behaviors), as this enables the mediation process to be performed on the ontological level of application domains, while other types of specifications should be used as secondary filters. We indicated that services' external behavior should be adopted instead of their internal behavior to maintain services' encapsulation. We proposed use of scenarios to partially capture the services' external behavior, however we argued that m-to-n state matching approaches should be adopted during behavior matching instead of one-to-one state matching approaches in order to minimize the appearance of false negatives. In this thesis, we indicated the types of semantics to be captured, the models to be used, the matching rules to be applied, the correctness criteria to be followed, the mediation techniques to be embraced and the matching

approaches to be adopted, in order to automate the service matching process. We summarize the proposed solutions as follows:

- We proposed the SWSMF framework to capture the technical and non-technical specifications of services' descriptions and users' requests in a machine-understandable format. Within SWSMF, we proposed the G^+ model that captures the goals of both users and services. The G^+ model links between the goals, the achievement contexts, and the corresponding realization scenarios.

- We proposed a meta-ontology approach for modelling application domains to overcome the semantic interoperability problem as well as facilitate the ontology mapping process. The proposed meta-ontology captures the concepts and the operations of application domains in its schematic layer, and captures in its semantic layer, in a context-based manner, the functional substitution semantics between application domain concepts; using the proposed Concepts Substitutability Graph (CSG).

- We proposed the Functional Substitutability Matching Scheme (FSMS) to match the high-level functional specifications. FSMS uses substitutability as the matching rule and user goal achievement as a correctness criterion. FSMS adopts two mediation techniques using the semantics captured in the CSG: it uses the proposed constraint substitutability approach to match constraints, and it uses the proposed state sequence mediation procedure (SMP) to match behavior models. The proposed constraint substitutability approach is able to match constraints with different scopes, while SMP is able to match state sequences with different length, as it adopts an m-to-n state matching approach. FSMS does not adopt any rigid taxonomies to determine concepts' functional equivalence but it determine such equivalence in a context-based manner with respect to users' goals.

- We proposed two correctness-aware high-level functional service matching approaches (direct and aggregate) that match services according to their G^+ goals, adopting FSMS. Hence, they guarantee the achievement of users' goals and do not require services' descriptions and users' requests to be described using the same concepts.

6.1 Future Work

We emphasis on the following directions as extension of the work proposed in this thesis.

6.1.1 Flexibility and Performance Enhancement

Adopting more advanced constraint model and optimizing the developed algorithms are important directions for future work. We believe there is good potential for enhancing the complexity of SMP algorithm and the aggregation algorithm using different heuristics. Also work in this thesis is based only on the mandatory constraints captured in the functional contexts, however more flexibility could be achieved by taking into consideration both the mandatory and optional constraints when matching functional context. This can be achieved by adopting imprecise computation techniques such as the one discussed in [Elhaweet et al., 2001]. Also to optimize service registry storage, the G^+ models could be described via XML files, then stored in a compressed format. Taking the advantage of having the exact queries over G^+ models to be matched without the need for decompression [Tolani and Haritsa, 2002; Arion et al., 2003; Cheng and Ng, 2004].

6.1.2 Service Selection

As this thesis proposed a matching scheme for the high-level functional specifications. Matching schemes for other types of specifications need to be developed. The matching scheme required for non-functional and non-technical specifications need to indicate what will be the matching rules as well as what will be the correctness criteria, taking into consideration the need for End-to-End measures when aggregate services are adopted. Hence, the proposed NTC, NFC, and LLFC need to be extended to capture more semantics such as information about service level agreements, security, and negotiations.

6.1.3 Service Discovery

Work in this thesis is based on a single service registry, which is considered a solution for the centralized service discovery problem. However, we are not expecting all services will

be registered in one registry. Therefore peer-to-peer technologies such as distributed hash indexing protocols [Stoica et al., 2001; Wang et al., 2003; Schmidt and Parashar, 2004] could be adopted to manage the service matching over multiple registries. Request rewriting and rerouting becomes mandatory in such environment.

6.1.4 Service Composition

In this thesis, we provided a solution for sequential service aggregation. However, we have not investigated the execution details in order to determine whether the aggregate service is executable or not. Hence, specifying what should be stored in LLFC to check if services are executable is an important research direction. Another research direction is to adopt a concurrent composition approach instead of the sequential one. However, concurrent execution for the services will trigger new levels of complexity to ensure the execution correctness.

Appendix A

SWSMF Realization

There are four perspectives that should be taken into consideration for the realization of the SWSMF framework, as each perspective plays a different role during the matching process. These perspectives are: an ontology engineer's perspective, a service provider's perspective, a user's perspective, and a matchmaker's perspective. In what follows we show how SWSMF is realized for each perspective by listing the tasks required within each perspective. Integrating the four perspectives together gives the whole picture of how our approach handles the matching process.

Ontology Engineer Perspective: An ontology engineer is the person responsible for building application domain ontologies. As SWSMF specifies a meta-schema for building domain ontologies, an ontology engineer should adopt this meta-schema in order to be able to build valid application domains' ontologies. Therefore, an ontology engineer has to accomplish the following tasks.

1. **Identify the application domain**. Every application domain has a unique code registered in the application domain repository. If the required application domain is not registered (does not exist in the repository), the ontology engineer has to register the application domain in order to obtain its corresponding code.

2. **Identify the concepts of the application domain**. The ontology engineer has to

define the entities of the application domain as well as their attributes. The type and range of every attribute should be defined.

3. **Identify the roles of the application domain**. The ontology engineer has to define the different roles (social actors) of the application domain, as well as their attributes. The type and range of every attribute should be defined.

4. **Identify the operations of the application domain**. The ontology engineer has to identify the transactions' types of the application domain. For every transaction type, the corresponding operation should be defined. For every operation, its input, output, pre-conditions, post-conditions, and attributes need to be defined. Inputs and outputs are chosen from the defined concepts.

5. **Build concept substitutability graph**. The ontology engineer has to define the substitution rules, conversion functions, and operator mapping matrices for every pair of the defined concepts with respect to every defined operation in order to be able to build the concepts substitutability graph.

6. **Register the devised ontology**. After accomplishing the previous tasks, the ontology engineer will have an application domain reference and it corresponding valid ontology, which should be published in the required repositories.

The correctness and the quality of the models used in building an application domain ontology is the responsibility of the ontology engineer.

Service Provider Perspective: Service providers are the ones responsible for creating and publishing a given service. Therefore, they have to accomplish the following tasks in order to publish a new service.

1. **Identify the application domains**. Using the application domain repository, they have to select the application domains (get the codes) for the new services.

2. **Identify the used ontologies**. Using the ontologies repository (that adopts the

proposed meta-ontology), they have to choose the ontologies to be adopted for every required application domain.

3. **Identify service goals**. For every chosen application domain and for every adopted ontology, they have to choose the operations that represent the service's goals.

4. **Identify the high-level functional contexts**. For every defined goal, they have to define the corresponding high-level functional context by listing the corresponding pre-constraints, describing-constraints, and post-constraints.

5. **Identify users roles**. For every defined goal, they have to define the beneficiary roles of the corresponding application domain, which represent the target market sectors for their services.

6. **Identify the expected GAPs**. For every defined goal, they have to define the expected interaction scenarios between the new service and the prospective users, in order to achieve the goal.

7. **Extract GAP forests**. For every defined goal, they have to build the corresponding G^+ model using the defined functional contexts and GAPs. Then, they have to extract the corresponding GAP-forest for every G^+ model.

8. **Identify the Supported Non-functional Specifications**. For every defined goal, they have to define the supported non-functional specifications in order to build the non-functional contexts.

9. **Identify Supported Non-technical Specifications**. For every defined goal, they have to define the supported non-technical specifications in order to build the non-technical contexts.

10. **Identify Low-Level functional Specifications**. For every defined goal, they have to define the supported low-level functional specifications in order to build the low-level functional contexts.

11. **Build SAVs and SPVs**. From the above, they have to build the corresponding SAVs. Also if they like to publish any documents for human users they have to included in the SPVs.

12. **Publish the new service**. After building the SAVs and SPVs of the new web services, they have to register the services in the desired service directory.

User Perspective: Users need to format their request according to SWSMF specifications. For this, they have to accomplish the following tasks.

1. **Identify the application domains**. Using the application domain repository, they have to select the required application domain (that is get the codes).

2. **Identify the used ontologies**. Using the ontologies repository (that adopts the proposed meta-ontology), they have to select an ontology for the selected application domain.

3. **Identify required goals**. For every chosen application domain and for every adopted ontology, they have to choose the operations that represent the required goals.

4. **Identify the high-level functional contexts**. For every defined goal, they have to define the required high-level functional context by listing the required pre-constraints, describing constraints, and post-constraints.

5. **Identify users roles**. For every defined goal, they have to define their role.

6. **Identify the expected GAPs**. For every defined goal, they have to define the expected interaction scenarios between them and the required service.

7. **Identify the Required Non-functional Specifications**. For every defined goal, they have to define the required non-functional specifications in order to build the non-functional context.

8. **Identify the Required Non-technical Specifications**. For every defined goal, they have to define the required non-technical specifications in order to build the non-technical context.

9. **Identify the Required Low-Level Functional Specifications**. For every defined goal, they have to define the required low-level functional specifications in order to build the low-level functional context.
10. **Build UAVs**. For the above, they have to build the corresponding UAV.
11. **Submit UAVs**. Generated UAVs will be submitted to the matchmaker to retrieve the results.

Human users might not be capable to build their own UAVs. Therefore, templates for various goals supported by the chosen ontologies could be given to the user to select from, and some to modify, as every ontology has a finite set of operations. Also, tools (such as ontology navigators) could be provided to build the required UAVs.

Matchmaker Perspective: The matchmaker will examine the submitted UAVs against the published SAVs. Therefore, a matchmaker needs the following in order to be able to work.

- **A global coding scheme for applications domains**. This is ensured when one repository is used to maintain application domain codes.
- **Access to a valid ontologies repository**. This repository contains all the defined ontologies for the registered application domains. All these ontologies must follow the proposed meta-ontology. Also, a list of mapped ontologies should be available in the repository.
- **A defined matching scheme**. A matchmaker should know which matching scheme is required for every type of specifications.
- **At least one SAV for every registered service**.
- **At least one UAV for every user request**. The matchmaker need to generate the corresponding GAP-forest from a user UAV.

The matching process will be performed in a multi-stage manner, in which every stage applies a different filter. The used filters and their corresponding order is as follows.

1. **Application domain filter**: Services with different application domain codes from the ones defined in the UAV will be ignored.

2. **Adopted ontology filter**: Services with different ontology codes from the one defined in the UAV and that do not exist in the ontology mapping list, will be ignored.

3. **Role filter**: Services with different supported roles from the role defined in the UAV (that is cannot functionally substitute the user's role) will be ignored.

4. G^+ **filter**: Adopting the required matching scheme, services with G^+ models that do not match the required G^+ model in the UAV will be ignored.

5. **Other Specifications filters**: Adopting the required matching schemes for non-technical, non-functional, and low-level functional specifications, the remaining set of services will filtered.

After applying all of these filters in the indicated order, the remaining results are guaranteed to correctly fulfill the user request. For human users, SPVs can be used in service selection process.

Bibliography

A. Abecker, R. Tellmann, and S. Grimm. Analysis of web service solutions and frameworks. Technical Report IST-2001-37134, SWWS Semantic Web Enabled Web Services, March, http://swws.semanticweb.org/public_doc/D1.2.pdf 2004.

R. Aggarwal, K. Verma, J. Miller, and W. Milnor. Constraint driven web service composition in METEOR-S. In *Proceedings of IEEE International Conference on Services Computing*, pages 23–30, NYC, USA, 2004.

M. Aiello, M. P. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, and P. Traverso. A request language for web-services based on planning and constraint satisfaction. In *Proceedings of the Third International Workshop on Technologies for E-Services*, pages 76–85, Hong Kong, China, 2002.

G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, A. I. Manolescu, and Pugliese. Xquec: pushing queries to compressed xml data. In *Proceedings of the Twenty Nineth International Conference on Very Large Data Bases (VLDB)*, pages 1065–1068, Berlin, Germany, 2003.

S. Balzer, T. Liebig, and M. Wagner. Pitfalls of OWL-S: A practical semantic web use case. In *Proceedings of the second International Conference on Service Oriented Computing (ICSOC)*, pages 289–298, NY, USA, November 2004.

B. Benatallah, M.-S. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB*, 14(1):84–96, 2005.

V. Benjamins, J. Contreras, O. Corcho, and A. Gmez-Prez. Six challenges for the semantic web. *AIS SIGSEMIS Bulletin*, 1(2):24–25, 2004.

D. Berardi. *Automatic Service Composition: Models, Techniques and Tools.* PhD thesis, Universit'a degli Studi di Roma La Sapienza, 2005.

D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proceedings of the first International-Conference on Service Oriented Computing (ICSOC)*, pages 43–58, Trento, Italy, 2003.

T. Berners-Lee, J. Hendler, and O. Lassila. *The Semantic Web.* Scientific American, May 2001.

A. Bernstein and M. Klein. Towards high-precision service retrieval. In *Proceedings of the first International Semantic Web Conference (ISWC)*, pages 84–101, Sardinia, Italy, 2002.

P. Bernstein. Middleware: A model for distributed systems services. *Communications of the ACM*, 39(2):86–98, 1996.

T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of the twelfth International World Wide Web Conference*, pages 403–410, Budapest, Hungary, 2003.

C. Bussler. Semantic web services: reflections on web service mediation and composition. In *Proceedings of the fourth International Conference on Web Information Systems Engineering (WISE)*, pages 253–260, Rome, Italy, 2003.

F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and dynamic service composition in eflow. In *Proceedings of the twelfth International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 13–31, Stockholm, Sweden, 2000.

J. Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. In *Proceedings of Workshop on Application of Description Logics*, Vienna, Austria, September 2001.

D. Chakraborty, F. Perich, S. Avancha, and A. Joshi. Dreggie: Semantic service discovery for m-commerce applications. In *Proceedings of the twentieth Symposiom on Reliable Distributed Systems: Workshop on Reliable and Secure Applications in Mobile Environment*, pages 28–31, 2001.

J. Cheng and W. Ng. Xqzip: Querying compressed xml using structural indexing. In *Proceedings of the Nineth International Conference of Extending Database Technology (EDBT)*, pages 219–236, Heraklion-Crete, Greece, 2004.

J. Colgrave, R. Akkiraju, and R. Goodwin. External matching in uddi. In *Proceedings of the Second IEEE International Conference on Web Services (ICWS)*, pages 226–233, San Deigo, USA, 2004.

A. Dardenne, A. vanLamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, (20):3–50, 1993.

K. Decker, M. Williamson, and K. Sycara. Matchmaking and brokering. In *Proceedings of the Second International Conference in Multi-Agent Systems(ICMAS)*, pages 1–19, Kyoto, Japan, 1996.

A. Dey, D. Salber, and G. Abowd. A context-based infrastructure for smart environments. In *Proceedings of the first International Workshop on Managing Interactions in Smart Environments*, pages 114–128, Dublin, Ireland, 1999.

A. Dogac, Y. Tambag, P. Pembecioglu, S. Pektas, G. Laleci, G. Kurt, S. Toprak, and Y. Kabak. An ebXML infrastructure implementation through uddi registries and rosettanet pips. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 512–523, NY, USA, 2002.

X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In *Proceedings of the thirtieth International Conference on Very Large Data Bases (VLDB)*, pages 132–143, Toronto, Canada, 2004.

M. Dumas, J. O'Sullivan, M. Heravizadeh, A. ter Hofstede, and D. Edmond. Towards a semantic framework for service description. In *Proceedings of the IFIP Conference on Database Semantic*, pages 277–291, Hong Kong, China, 2001.

I. Elgedawy. A conceptual framework for web services semantic discovery. In *Proceedings of On The Move (OTM) to meaningful internet systems*, pages 1004–1016, Catania, Italy, November 2003.

I. Elgedawy, Z. Tari, and J. Thom. A high level functional matching for semantic web services. In *Proceedings of the third International Conference on Service Oriented Computing (ICSOC)*, pages 115–129, Amsterdam, Netherlands, 2005.

I. Elgedawy, Z. Tari, and M. Winikoff. Exact functional context matching for web services. In *Proceedings of the second International Conference on Service Oriented Computing (ICSOC)*, pages 143–152, NY, USA, 2004a.

I. Elgedawy, Z. Tari, and M. Winikoff. Scenario matching using functional substitutability in web services. In *Proceedings of the fifth International Conference on Web Information Systems Engineering (WISE)*, pages 59–65, Brisbane, Australia, 2004b.

W. Elhaweet, I. Elgedawy, and I. Abd−Elsalam. Adaptive fixed priority end-to-end imprecise scheduling in distributed real time systems. In *Proceedings of International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, USA, 2001.

D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

D. Fensel, F. H. I. Horrocks, D. McGuinness, and P. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, May 2001.

C. Fidge. Contextual matching of software library components. In *Proceedings Of the nineti-eth Asia-Pacific Software Engineering Conference (APSEC)*, pages 297–307, Gold Coast, Queensland, Australia, 2002.

R. Findler, M. Latendresse, and M. Felleisen. Behavioral contracts and behavioral subtyping. In *Proceedings of the eightieth European software engineering conference held jointly with the nineth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 229–236, Vienna, Austria, 2001.

P. Ganesan, H. G. Molina, and J. Widom. Exploiting hierarchical domain structure to compute similarity. *ACM Transactions on Information Systems*, 21(1):64–93, January 2003.

S. Green. *A Synthesised Approach to Goal-Oriented Requirements Engineering.* PhD thesis, University of London, Imperial College of Science Technology and Medicine, Department of Computing, 2004.

T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1995.

N. Guarino. Semantic matching: Formal ontological distinctions for information organization, extraction, and integration. In *Proceedings of International Summer School on Information Extraction (SCIE)*, pages 139–170, Frascati, Italy, 1997.

V. Haarslev and R. Möller. RACER system description. In *Proceedings of the first International Joint Conference on Automated Reasoning*, pages 701–706, Siena, Italy, 2001.

M. Hattori, K.Cho, A. Ohsuga, and M. Isshiki. Context-aware agent platform in ubiquitous environments and its verification tests. In *Proceedings of the first IEEE International Conference on Pervasive Computing and Communications*, USA, March 2003.

B. Hofreiter, C. Huemer, and W. Klas. ebXML: Status, research issues, and obstacles. In *Proceedings of the twelfth IEEE International Workshop on Research Issues in Data Engineering (RIDE)*, pages 7–18, San Jose, USA, 2002.

I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proceedings of Principles of Knowledge Representation and Reasoning*, pages 636–645. Trento, Italy, 1998.

HP. Espeak: Architecture specification. http://www.hpl.hp.com/personal/Alan-Karp/espeak/version2.2/Architecture-2.2.pdf, Septemeber 1999.

Y. Huang and N. Venkatasubramanian. QoS-based resource discovery in intermittently available environments. In *Proceedings of the eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 50–60, Edinburgh, Scotland, UK, 2002.

M. Huhns. Agents as web services. *IEEE Internet Computing*, 6(4):93–94, 2002.

P. Jeavons and M. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence*, 79(2):327–339, 1995.

Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

V. Kashayp and A. Sheth. Semantic and schematic similarities between database objects: A context-based approach. *The VLDB journal*, 5(4):276–304, 1996.

U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, and D. Fensel. WSMO web service discovery. http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112, 2004.

G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley, 1998.

O. Lassila and R. R. Swick. Resource description framework(rdf) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 1999.

A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. In *Proceedings of the first InternationalConference on Service Oriented Computing (ICSOC)*, pages 335–350, Trento, Italy, 2003.

A. Lazovik, M. Aiello, and M. Papazoglou. Associating assertions with business processes and monitoring their execution. In *Proceedings of the Second International Conference on Service Oriented Computing*, pages 94–104, New York, USA, 2004.

A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*, 6(3), 2006.

C. Lee and S. Helal. Context attributes: An approach to enable context-awareness for service discovery. In *Proceedings of the third IEEE/IPSJ Symposium on Applications and the Internet*, Orlando, Florida, USA, January 2003.

P. Loucopoulos and E. Kavakli. Enterprise modelling and the teleological approach to requirements engineering. *International Journal of Cooperative Information Systems*, 2(1): 45–79, 1995.

H. Lu. Semantic web services discovery and ranking. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, pages 157–160, 2005.

J. Luo, B. Montrose, and M. Kang. An approach for semantic query processing with uddi. In *Proceedings of On The Move (OTM) to meaningful internet systems*, pages 89–98, Agia Napa, Cyprus, 2005.

J. Magee, J. Kramer, and D. Giannakopoulou. Analysing the behaviour of distributed software architectures: a case study. In *Proceedings of the sixth IEEE Workshop on Future Trends of Distributed Computing Systems*, pages 240–245, Tunis, Tunisia, 1997.

E. M. Maximilien and M. P. Singh. Toward autonomic web services trust and selection. In *Proceedings of the second International Conference on Service Oriented Computing (ICSOC)*, pages 212–221, 2004.

S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2): 46–53, 2001.

M. Mecella, B. Pernici, and P. Craca. Compatibility of e-services in a cooperative multi-platform environment. In *Proceedings of the second International VLDB Workshop on Technologies for e-Services (VLDB-TES)*, pages 44–57, Rome, Italy, 2001.

B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing web services on the semantic web. *Very Large Data Base Journal*, 12(4):333–351, 2003.

P. Mika, A. Gangemi, D. Oberle, and M. Sabou. Foundations for service ontologies: Aligning OWL-S to DOLCE. In *Proceedings of the thirteenth International World Wide Web Conference*, pages 563–572, NYC, USA, 2004.

B. Orriëns, J. Yang, and M. Papazoglou. Servicecom: A tool for service composition reuse and specialization. In *Proceedings of the fourth International Conference on Web Information Systems Engineering (WISE)*, pages 355–358, Rome, Italy, 2003.

B. Orriëns, J. Yang, and M. Papazoglou. Service component: a mechanism for web service composition reuse and specialization. *Journal of Integrated Design and Process Science*, 8 (2):13–28, 2004.

OWL-Services-Coalition. OWL-S: Semantic markup for web services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2003.

M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Importing the semantic web in uddi. In *Proceedings of the fourteenth International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 225–236, Toronto, Canada, 2002a.

M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the first International Semantic Web Conference (ISWC)*, pages 333–347, Sardinia, Italy, 2002b.

M. Papazoglou, M. Aiello, M. Pistore, and J. Yang. Planning for requests against web services. *IEEE Data Engineering Bulletin*, 25(4):41–46, 2002.

M. Papazoglou and D. Georgakopoulos. Service oriented computing. *Communications of the ACM (special issue)*, 46(10):24–28, 2003.

M. P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the fourth International Conference on Web Information Systems Engineering (WISE)*, Rome, Italy, 2003.

P. Patel-Schneider and D. Fensel. Layering the semantic web: Problems and directions. In *Proceedings of the first International Semantic Web Conference*, pages 16–29, Sardinia, Italia, 2002.

A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S web service annotation framework. In *Proceedings of the thirteenth international conference on World Wide Web*, pages 553–562, NYC, USA, 2004.

J. Pearson and P. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, citeseer.nj.nec.com/pearson97survey.html, Oxford University, Computing Laboratory, 1997.

B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–584, 2000.

M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso. Planning and monitoring web service composition. In *Proceedings of the second ICAPS International Workshop on Planning and Scheduling for Web and Grid Services*, pages 70–77, British Columbia, Canada, 2004.

S. Pokraev, J. Koolwaaij, and M. Wibbels. Extending uddi with context-aware features based on semantic service descriptions. In *Proceedings of the first International Conference on Web Services*, pages 184–190, Las Vegas, USA, 2003.

C. Potts, K. Takahashi, and A. Anton. Inquiry-based requirements analysis. *IEEE Software*, 11(2):21–32, 1994.

I. Pratistha and A. Zaslavsky. Fluid: supporting a transportable and adaptive web service. In *Proceedings of the nineteenth ACM symposium on Applied computing*, pages 1600–1606, Nicosia, Cyprus, 2004.

P. Rajasekaran, J. Miller, K. Verma, and A. Sheth. Enhancing web services description and discovery to facilitate composition. In *Proceedings of the first International Workshop on Semantic Web Services and Web Process Composition (SWSWPC)*, San Diego, CA, USA, 2004.

S. Ran. A model for web services discovery with QoS. *ACM SIGecom Exchanges*, 4(1):1–10, 2003.

P. Resnick. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 448–453, Montre'al, Que'bec, Canada, 1995.

J. Roddick, K. Hornsby, and D. Vries. A unifying semantic distance model for determining the similarity of attribute values. *Proceedings of the twenty-sixth Australasian computer science conference*, 16:111–118, 2003.

C. Rolland and C. B. Achour. Guiding the construction of textual use case specifications. *Data and Knowledge Engineering Journal*, 25(1-2):125–160, 1998.

C. Rolland, C. Souveyet, and C. B. Achour. Guiding goal modelling using scenarios. *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*, 24(12): 1055–1071, 1998.

D. Roman, U. Keller, and H. Lausen. Web service modeling ontology (WSMO). http://www.wsmo.org/TR/d2/v1.1/20050210/, Feburary 2005.

A. Sajjanhar, J. Hou, and Y. Zhang. Algorithm for web services matching. In *Proceedings of the sixth Asia-Pacific Web Conference*, pages 665–670, 2004.

G. Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.

C. Schmidt and M. Parashar. A peer-to-peer approach to web service discovery. *World Wide Web Journal*, 7(2):211–229, 2004.

H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proceedings of the twelfth International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 247–263, Stockholm, Sweden, 2000.

K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Proceedings of the first International Conference on Web Services (ICWS)*, pages 395–401, Las Vegas, USA, June 2003.

S. Staab, W. van-der Aalst, V. Benjamins, A. Sheth, A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. Web services: been there, done that? *IEEE Intelligent Systems*, 18(1):72–85, 2003.

I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, USA, 2001.

M. Sundaram and S. S. Shim. Infrastructure for b2b exchanges with rosettanet. *Proceedings of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems(WECWIS)*, 2001.

K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Stanford University, USA, 1999.

K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, 2004.

K. Sycara, S. Wido, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Journal of Autonomous Agents and Multi-Agent Systems*, 5:173–203, 2002.

J. Tatemura, A. Sawires, O. Po, D. Agrawal, and K. S. Candan. Wrex: A scalable middleware

architecture to enable xml caching for web services. In *Proceedings of ACM/IFIP/USENIX Sixth International Middleware Conference*, pages 124–143, Grenoble, France, 2005.

S. Thakkar, J. Ambite, and C. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of the second ICAPS International Workshop on Planning and Scheduling for Web and Grid Services*, British Columbia, Canada, 2004.

S. Thakkar, J. Ambite, C. Knoblock, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of AAAI Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, 2002.

P. Tolani and J. Haritsa. Xgrind: A query-friendly xml compressor. In *Proceedings of the Eighteenth International Conference on Data Engineering (ICDE)*, pages 225–234, San Jose, USA, 2002.

S. Uchitel. *Incremental Elaboration of Scenario-Based Specifications and Behaviour Models Using Implied Scenarios*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 2003.

UDDI-Technical-Committee. UDDI specifications version 3.02. http://uddi.org/pubs/uddi_v3.htm, October 2004.

Q. Wang, Y. Yuan, J. Zhou, and A. Zhou. Peer-serv: A framework of web services in peer-to-peer environment. In *Proceedings of Web-Age Information Management (WAIM)*, pages 298–305, Chengdu, China, 2003.

K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in system development: Current practice. *IEEE Software*, 15:34–45, 1998.

P. Weinstein and W. Birmingham. Comparing concepts in differentiated ontologies. In *Proceedings of the twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW)*, number 6, pages 1–22, Voyager Inn, Banff, Alberta, Canada, 1999.

WS-Challenge. Introducing the web service challenge. http://ws-challenge.org/, 2005.

L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30 (5):311–327, 2004.