Automatic Service Discovery and Composition for Heterogeneous Service Partners

Zille Huma

A thesis submitted to the
Faculty of Computer Science, Electrical Engineering, and Mathematics
of the University of Paderborn
in partial fulfillment of the requirements
for the degree of Dr. rer. nat.

June 2015

Supervisors:

Prof. Dr. Gregor Engels, University of Paderborn, Germany
Prof. Dr. Christian Gerth, University of Applied Sciences Osnabrück, Germany

Doctoral Committee:

Prof. Dr. Gregor Engels, University of Paderborn, Germany
Prof. Dr. Christian Gerth, University of Applied Sciences Osnabrück, Germany
Prof. Dr. Leena Suhl, University of Paderborn, Germany
Prof. Dr. Heike Wehrheim, University of Paderborn, Germany
Dr. Theodor Lettmann, University of Paderborn, Germany

Date of public examination:

July 9th, 2015

Acknowledgments

The research work for this dissertation was carried out in the Database and Information Systems group at University of Paderborn headed by Prof. Dr. Engels. It was partially supported by International Graduate School of Dynamic Intelligent Systems and the German Research Foundation (DFG) within the Collaborative Research Center 901 "On-The-Fly Computing".

There are quite a few people, whose support and input made the completion of this dissertation possible. I would like to thank each of them.

First and foremost, I owe my deepest gratitude to my supervisor, Prof. Dr. Gregor Engels for his invaluable support and patience. I am thankful to him for showing me how to do research, for his technical insights into the topic and for the personal guidance and motivation when I lost the path during the course of this work. I would also like to express my deepest gratitude to my co-supervisor Prof. Dr. Christian Gerth, who believed in my abilities, constantly guided and encouraged me to continue scientific work and extensively participated in our scientific writing activities. Christian! I owe a major part of my success in terms of the published scientific papers to you. Additionally, I would like to thank Prof. Dr. Leena Suhl, Prof. Dr. Heike Wehrheim and Dr. Theodor Lettmann for being the members of my doctoral committee.

I would also like to thank all my colleagues from the Database and Information Systems research group for the highly professional and extremely friendly environment that contributes to the highly-valuable scientific work produced in this group.

Further, I am deeply thankful to Prof. Dr. M. Jaffar-ur-Rehman (late) from M.A. Jinnah University, Pakistan, who first discovered my abilities of a potential researcher and guided me to take my first steps on this path. Dr. Jaffar! none of this was possible without you and I miss your presence on every achievement on professional and personal fronts.

Apart from these guiding forces in professional environment, the completion of this dissertation was not possible without the support of my family. Here, I owe a heartfelt thanks to my parents for their unconditional love, for their belief in my abilities, for their big dreams for me and for their extensive effort to provide me with the best possible resources and opportunities to excel in different directions in my academic life. Ammi and Abbu! you are the reason of my existence and I owe everything that I achieved in life only to you.

Last but not the least, the most heartfelt thanks goes to my husband Tauseef and my daughter Eshaal. Eshaal! thank you for all your patience,

understanding and support during my work. You are the sunshine of my life that lit my most stressed days with its beautiful smile. More than anything else, you are the center of my universe. Tauseef! thank you for being that strongest pillar of support for me to lean on when I needed it the most. Thank you for your unconditional support, for bearing with all my mood swings during the course of this work, for making all those sacrifices just to let me achieve my dream and for putting things back in perspective for me whenever they went haywire. You are the wind beneath my wings!

Finally, I thank the International Graduate School of Dynamic Intelligent Systems for partially providing the financial support for this dissertation. Additionally, I am thankful for the financial and scientific support of the Collaborative Research Center 901.

Abstract

For distributed software systems, Service-oriented computing (SOC) emerged as a promising trend to overcome some major challenges that hurdle the development of cross-organizational large-scale enterprise systems. SOC is based on the idea of providing and consuming distributed software components as *services* by the service partners. In this context, service providers independently develop and publish their services on the service markets, which in turn are automatically discovered and consumed by the service requesters to fulfill their needs. With the widespread acceptance of SOC and service markets, the number of available software services steadily increased in recent years. To fully leverage the opportunities provided by this plethora of services for the development of highly flexible and aligned enterprise systems, there is a requirement for automatic and accurate service discovery and composition approaches. These approaches mainly rely on automatic service description matching, which is a complex task due to the challenges like lack of comprehensive service descriptions and the underlying multifaceted heterogeneity of the service partners.

In contrast to the current standards like WSDL that allow structural service descriptions, there is a requirement for comprehensive service descriptions, which comprise structural as well as behavioral aspects of the services. Otherwise an accurate service discovery and composition is not possible. An automatic matching of such comprehensive service requests and offers is complicated due to the multifaceted heterogeneity of the service partners including the use of different specification languages, different data models, or different levels of granularity in the specification itself.

In this thesis, we present a framework for automatic service discovery and composition mechanism, which is based on rich service description language (RSDL) with visual notations to specify the service descriptions in terms of their structural and behavioral aspects. Based on such comprehensive service descriptions, our framework introduces an elaborated multi-phase matching mechanism for the service requests and offers. Through different phases, it also overcomes the underlying heterogeneity of the service partners using different techniques, e.g., data model heterogeneity is resolved through service description normalization on the basis of local-global data model matching. Similarly, granularity level heterogeneity and linguistic heterogeneity are also resolved during service description matching. As a proof of concept, our work is entailed with an implementation prototype, which implements a significant part of the proposed framework.

Zusammenfassung

Service-oriented Computing (SOC) ist ein vielversprechender Ansatz, um einige der großen Herausforderungen bei der Entwicklung von organisationsübergreifenden, hochflexiblen Software-Systemen für Grounternehmen zu überwinden. SOC basiert auf der Idee der Bereitstellung und Nutzung verteilter Software-Komponenten, die in Form von Services von Service-Anbietern bereitgestellt werden. Service-Anbieter entwickeln ihre Services unabhängig und veröffentlichen diese auf Service-Märkten, die von Service-Nutzern verwendet werden können, um geeignete Services zu finden. Die Anzahl solcher Service-Märkte sowie der bereitgestellten Services ist in den letzten Jahren stetig gestiegen. Um von dieser Vielzahl an Services bei der Entwicklung von hochflexiblen Software-Systemen für Großunternehmen profitieren zu können, müssen geeignete Services automatisiert gefunden und zu Service-Kompositionen zusammengesetzt werden können. Ansätze hierfür setzen meist auf ein automatisiertes Service-Matching, welches den gesuchten Service mit den angebotenen Services abgleicht. Ein solches Matching ist allerdings sehr komplex, da Service-Beschreibungen oft nicht ausführlich genug sind und die Beschreibungen des gesuchten Service sowie der angebotenen Services sehr heterogen sind.

Im Gegensatz zu gängigen Standards zur Beschreibung von Services wie der Web Service Description Language (WSDL), die zur Beschreibung von strukturellen Aspekte eines Dienstes genutzt werden können, besteht ebenso die Notwendigkeit auch das Verhalten eines Dienstes zu beschreiben. Andernfalls ist ein Matching zwischen dem gesuchten Service und den angebotenen Services nur auf strukturellen Informationen möglich, was oft zu ungenauen Ergebnissen führt. Zusätzlich erschwert wird ein Matching aufgrund der Nutzung von unterschiedlichen Sprachen zur Beschreibung von Services, durch abweichende Datenmodelle oder dem unterschiedlichen Detailgrad der Service-Beschreibungen.

In dieser Arbeit wird ein Ansatz fr das automatisierte Finden von Services sowie deren Komposition vorgestellt, der auf der Rich Service Description Language (RSDL) basiert. Die RSDL besitzt visuelle Notationselemente, um die Struktur und das Verhalten sowohl von angebotenen als auch gesuchten Services umfassend zu beschreiben. Basierend auf solchen Beschreibungen wird ein mehrphasiger Matching-Ansatz vorgestellt, der die bereits erwähnten Probleme schrittweise angeht und löst. Zum Umgang mit unterschiedlichen Datenmodellen wird ein Local-Global Matching des Datenmodells durchgeführt, um die Service-Beschreibungen zu normalisieren. Auf ähnliche Art wird mit unterschiedlichem Detailgrad oder sprachlichen Un-

terschieden der Service-Beschreibungen umgegangen. Als Proof-of-Concept wurde im Rahmen dieser Arbeit ein Prototyp entwickelt, der einen Großteil des vorgestellten Ansatzes implementiert.

Contents

1 Introduction

In this chapter, we give a detailed motivation for the work presented in this thesis. This is followed by a layout of our salient research goals and corresponding research contributions. In the last section, we provide a structure for the rest of the thesis.

1.1 Motivation

With the infiltration of internet in our daily lives, business organizations are constantly striving to make an efficient use of the emerging web technologies and achieve an edge in the competitive market. In this direction, an important challenge is an ongoing improvement of their existing IT systems through its modification and collaboration with other existing systems.

In recent years, service-oriented computing (SOC) [55] has emerged as a promising trend that enables the vision of cross-organizational collaborations by allowing large-scale, heterogeneous and flexible systems at enterprise level. SOC is based on the idea of publishing, discovering and consuming independently developed and distributed software components as *services*.

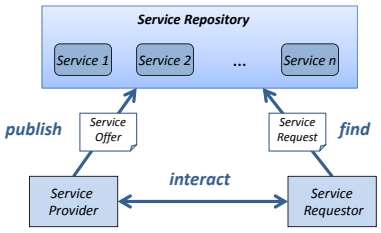

Figure 1.1: An Overview of Service Publishing and Discovery on Service Market

The general scenario of SOC concerning service publishing and discovery is shown in Figure 1.1. In this scenario main actors are service requesters and service providers that interact with each other on a service market. A service provider independently develops its service as a software

unit offering certain functionality. The developed service is then published in the repository of one or more service markets through its *service offer*, i.e., a description of the functionality offered by the service. On the other hand, a service requester discovers and composes the services available in the service repository to satisfy its *service request*, i.e., a description of the functionality required by the requester. A typical realization of this scenario can be seen in a service-oriented architecture (SOA) [111], where services are discovered by the SOA developers and used as basic building blocks for enterprise-level heterogeneous systems.

With the recent rise of cloud computing and its attached delivery models, such as, software as a service (SaaS), a steady increase in the number of available services can be observed. This growing plethora of available services along with the powerful service and platform technologies provide enormous opportunities for the development of highly-distributed and highly-scalable SOAs.

Currently, there are many milestones that need to be achieved for such a realization of the SOC. The achievement of these milestones is made difficult mainly due to the dynamic nature of the service market. In today's world, business markets are highly dynamic where businesses are always thriving for agility to meet the changing market conditions. Consequently, the service markets in the SOC paradigm are also changing constantly where new service providers may emerge all the time. The requirements of the service requesters are also constantly changing in terms of functionality and quality of the invoked services. This ever-changing nature of the service markets makes the achievement of following milestones highly difficult: maintaining high availability of services, assuring end-to-end security, and lack of well-advertised service registries, etc.

However, the most important milestone that needs to be achieved by SOC in this context is to allow automatic service discovery and composition while ensuring accurate results. This is enabled through the development of automatic service description matching mechanisms. Such mechanisms enable an automatic matching of requested and offered functionality specified in the service descriptions and deduce whether the requirements specified by the requester are fulfilled by one or more offered services. However, the development of such an automatic service description matching mechanism faces 2 important challenges: 1) lack of comprehensive service descriptions for the service partners, 2) the resolution of the underlying multifaceted heterogeneity of the service partners. These challenges are discussed in detail as follows.

Firstly, the service requests and the offers do not comprehensively specify

the requested and offered functionality, respectively. Consequently, an automatic matching based on such uncomprehensive service descriptions lead to inaccurate results hence making automatic service discovery difficult. For instance, the prevailing service description standard WSDL [162] allows to specify a service offer in terms of the structural aspects of the published service, i.e., its offered operation signatures. On the other hand, a service requester with no technical expertise often relies on keywords to search for the required services from the service market. However, our focus is a service requester with technical expertise, which also mainly relies on WSDL to describe its service request in terms of the required operation signatures. Such WSDL-based service descriptions are limited in the sense that they only specify the structural aspects of the required and offered functionality, which do not sufficiently describe the features required or offered by the service partners. For instance, such structural service descriptions do not represent the actual behavior of the requested/offered service. Consequently, a service description matching mechanism based on such descriptions can lead to inaccurate results. This is further clarified with the example shown in Fig. 1.2.

A requester with technical expertise wants to search and collaborate with an e-commerce service. For this purpose, he specifies his requirements in terms of required operation signatures as a WSDL-based service request. Additionally, he also has certain requirements concerning the sequence of operation invocation but this required behavior cannot be specified in WSDL. From the e-commerce services available on the service market, two services, i.e., *OnlineShop1* and *OnlineShop2* are selected by the service description matching mechanism. These matching results are based on a structural matching of the service request and offers where the required operation signatures *exactly* match the offered operation signatures for the selected service offers. But on a closer look, it is evident that only one of the two services, i.e., *OnlineShop1* actually fulfills the requester's requirements completely because it also offers the same operation invocation sequence as required by the requester.

Hence, a service description matching mechanism based on such structural service descriptions lead to inaccurate service discovery results. In order to develop service matching mechanisms that can ensure accurate results while automating the service discovery process, there is a requirement for more comprehensive service descriptions that also comprise further information, e.g., behavioral aspects in addition to the structural aspects of the required/offered services.

The idea of comprehensively describing the software components in terms

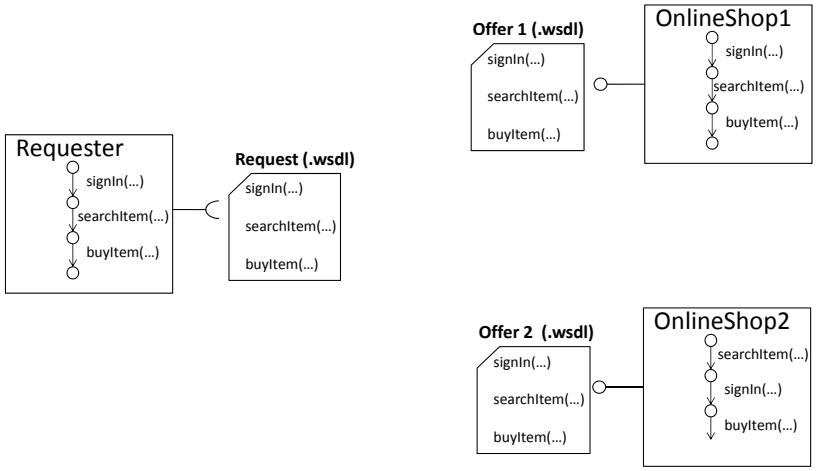

Figure 1.2: *Requester*, *OnlineShop1*, and *OnlineShop2* with same Operations but different Behavior

of their behavioral as well as structural aspects is not new and is already extensively addressed in the component-based software engineering (CBSE) domain [44, 32]. A variety of textual (e.g., Z-specification [146]) as well as visual languages (e.g., UML [115]) provide artifacts and notations to specify different types of behavior of a software component [63, 133].

For instance languages, such as, First-order logic [143], Object Constraint Language (OCL) [164], Visual Contracts (VC) [96], etc. particularly deal with functional behavior specifying the relationship between provided input, achieved output and the states of a component before and after its invocation. On the other hand, languages, such as, state machines [20], petri nets [58], business process model and notation (BPMN) [116] aim at describing temporal behavior specifying the possible sequence of the states achieved by a component resulting from the invocation of different transitions.

Similarly, the idea of comprehensive service descriptions has also been investigated in the SOC domain in recent years. Approaches like [65] propose to describe the operations in WSDL-based service descriptions in terms of their functional behavior. We use the term *behavioral semantics* for such a description of an operation in terms of its pre and post conditions as it describes its *meaning* through its behavior in contrast to its structural counterparts. Similarly, proposals like WS-BPEL [77, 112] and WS-CDL [161] allow a description of the temporal behavior of a service in terms of its invocation sequences, which is termed as its *service protocol*. In this direction, the semantic web services (SWS) paradigm [121, 51, 39, 98, 136]

has come up with languages and techniques to specify comprehensive service descriptions for service partners. However, so far these approaches also face certain limitations, e.g., some [98, 136] are not comprehensive enough in terms of behavioral description. On the other hand, approaches like WSML [51, 39] and OWL-S [121] are still limited to academia and are not widely accepted in practice because of diversion from the existing standards and/or the extra effort required by certain service partners to adapt to the complex textual notations [135].

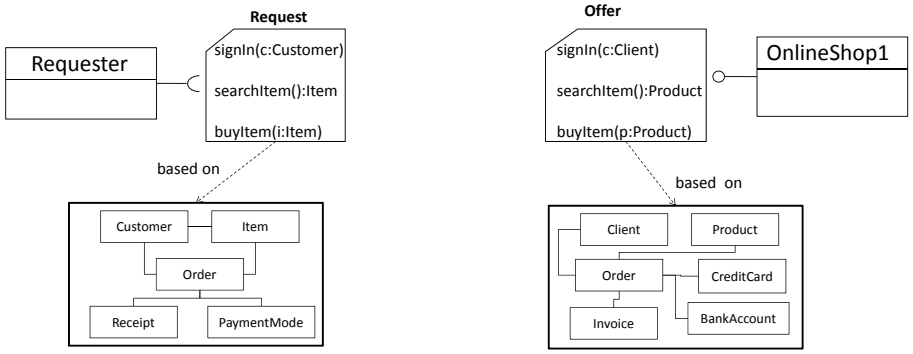

Figure 1.3: *Requester*, *OnlineShop1* and *OnlineShop2* with heterogeneous data models

Second important challenge faced for the development of an automatic service description matching mechanism is the inherent multi-faceted heterogeneity of the service partners in the SOC paradigm. One major reason of success of SOC is its claim to have flexible integration of the resources beyond the boundaries of the involved organizations and as a result allow these organizations to interact seamlessly. This claim of flexible integration and seamless interaction is only realizable if SOC has the ability to resolve the underlying heterogeneity of the service partners.

The first aspect of this multi-faceted heterogeneity is the data model heterogeneity of the service partners. SOC claims that it allows the service partners to function in their independent domains. Hence their service descriptions along with their respective data models *conform to* their independent understanding and knowledge of the domain, which may possibly be formalized as an ontology. An ontology is a mean to explicitly and formally define the meanings of the terms and concepts in a particular domain [148]. For instance, GoodRelations [68] is an ontology that formalizes the knowledge in the e-commerce domain. A conformance to such a formal ontology allows the service partners to explicitly specify the meaning of the

terms and concepts used in their service descriptions hence specifying their *ontological semantics*.

As a result of such independent domain knowledge, there may be situations where the service descriptions are *semantically similar* but *structurally different* or vice versa. To elaborate this point, an example is shown in Fig. 1.3. In this case, the requester's request and the providers' offers are based on their different knowledge of the e-commerce domain. As a result, their respective data models are heterogeneous despite representing the same domain. For instance, a class named as *Customer* in the requester's data model is modeled as *Client* in the provider's data model. Similarly, the classes *Item* and *Product* have different names but model same concept. *PaymentMode* in requester's data model corresponds to two classes *CreditCard* and *BankAccount* in the provider's data model. Hence, the association between *Order* and *PaymentMode* in the requester's data model is modeled as two associations namely *Order* to *CreditCard* and *Order* to *BankAccount* in provider's data model. Based on such heterogeneous data models, the resulting service descriptions are also heterogeneous where similar functionality can be specified differently or vice versa. An automatic service matching mechanism must be able to automatically overcome such a data model heterogeneity while matching the service descriptions.

The second aspect of the multi-faceted heterogeneity is the granularity-level heterogeneity of the service partners. Based on their independent domain knowledge, service partners can specify their service request/offer at different granularity levels, e.g., functionality specified in terms of a single operation in a service request may be specified by multiple operations in a service offer and vice versa. This results in complex correspondences, such as 1 : n, n : 1 and n : m between requested and offered operations in the service requests and offers. An example is provided in Fig. 1.4 for further clarification. The requester specifies two operations *findItem(...)* and *viewDetails(...)* in his service description. Additionally, we assume that service partners also describe their required and offered behavior and according to this description, *findItem(...)* finds an item based on the certain criteria and later on *viewDetails(...)* is used to view the details of the found item. On the other hand, *OnlineShop1* offers a single operation, i.e., *searchItem(...)* whose behavioral description comprise of searching an item and viewing its respective details. Based on the similarity between the requested and offered behavior, it can be deduced that there is an n : 1 correspondence between the requested and offered operations. Similarly, there is a 1 : n correspondence between the *buyItem(...)* in the service request and *validateCredentials(...)* and *makePayment(...)* in the service offer based on

their behavioral descriptions. An automatic service description matching mechanism must be able to detect such granularity level heterogeneity between the service descriptions and overcome it in order to enable an n : m matching between the requested and the offered operations.

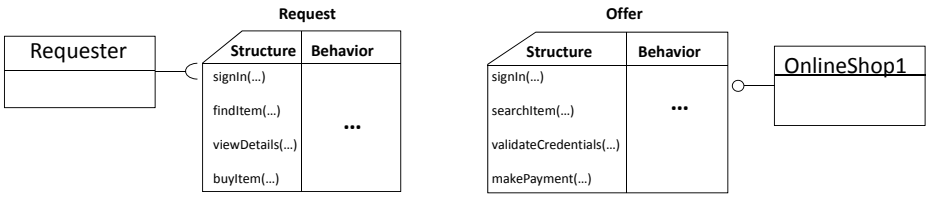

Figure 1.4: *Requester* and *OnlineShop1* with their service descriptions at different granularity level

Another possible consequence of such a difference of granularity level between service requests and offers is a *service composition*. In a realistic scenario, it is not always possible that a single service offer completely fulfills the service request and *multiple* service offers have to be composed to completely fulfill the service request. For instance in the considered e-commerce example, it can be the case that the offered service for an online shop only allows the search and selection of the items. To carry out the payment activity, a payment service has to be invoked by the requester. Hence, the online shop and payment services have to be composed in order to completely fulfill the request. The automatic service matching mechanism must also be able to determine such 1 : n matching between a service request and available offers leading to service composition.

Another aspect of this underlying heterogeneity is the linguistic heterogeneity among the service partners. Based on their independence allowed in an SOC environment, the service partners also have the flexibility to choose service description languages. This choice of language can be driven by different factors, such as, the nature of their domain, their preferences, their technical expertise and their requirements, etc. For instance, it is possible that a service provider, who is a skilled programmer prefers to use textual languages, such as OWL-S, WSML, etc. to specify different aspects of his service offer whereas a service requester who is an expert of system modeling prefers to work with visual notations and hence selects UML to specify his service request. Similarly, a service partner with expertise in formal languages may choose languages like Z-specification and CSP [71] for same purpose. Similarly, the service partners can also choose domain-specific languages that satisfy the particular needs of their domains, e.g., MARTE [23]

and UML-RT are languages that provide specialized notations to specify embedded and real-time systems.

In order to match such linguistically heterogeneous service descriptions of the service partners, the service matching mechanism must be able to translate them to a common representation and hence overcome their linguistic heterogeneity.

The practical application of such an automatic service discovery and composition mechanism meeting the above-mentioned challenges can be seen in the Collaborative Research Center (CRC) 901 "On-The-Fly (OTF) Computing" [1]. This also serves as an example scenario for this thesis. In Sec. 1.1.1, we give a detailed insight into different aspects of OTF Computing and introduce a real-world example from the e-tourism domain in this context.

1.1.1 An Example Scenario

To further elaborate the important concerns for automatic service discovery and composition and their practical application, we introduce the CRC 901 *OTF Computing* in this section. Additionally, we introduce the e-tourism scenario of our industrial partner *Hotel Reservation Service (HRS)* in the context of OTF Computing as a running example for this thesis.

A promising research endeavor aiming to overcome the important challenges faced by SOA is OTF Computing. The vision of OTF Computing is to develop methods and techniques:

- for the automatic and dynamic composition and execution of IT-services that are published by the service providers on the service markets,
- for ensuring the quality of these services,
- for the organization of the service markets.

Similar to the typical SOC scenario in Fig. 1.1, main actors in OTF Computing are also the *service requesters* and the *service providers*. Another main actor in OTF Computing is the *OTF provider* that enables a seamless interaction between the service requesters and the providers. To enable this seamless interaction, the *OTF provider* has to fulfill a variety of tasks. Firstly, it manages the service repository including a maintenance of

[1] http://sfb901.uni-paderborn.de/

the service offers according to their particular domains and categories. Secondly, it facilitates the service providers to publish their services through their service offers. Thirdly, the *OTF provider* enables an automatic service discovery and composition through automatic matching of service descriptions. For this purpose, the *OTF provider* uses tools and techniques that deal with different kinds of underlying heterogeneity mentioned earlier. Additionally, the *OTF provider* also analyzes the service discovery and composition results on the basis of the non-functional properties. However for the course of this thesis, this analysis of results on the basis of their non-functional properties is not relevant and hence not discussed further.

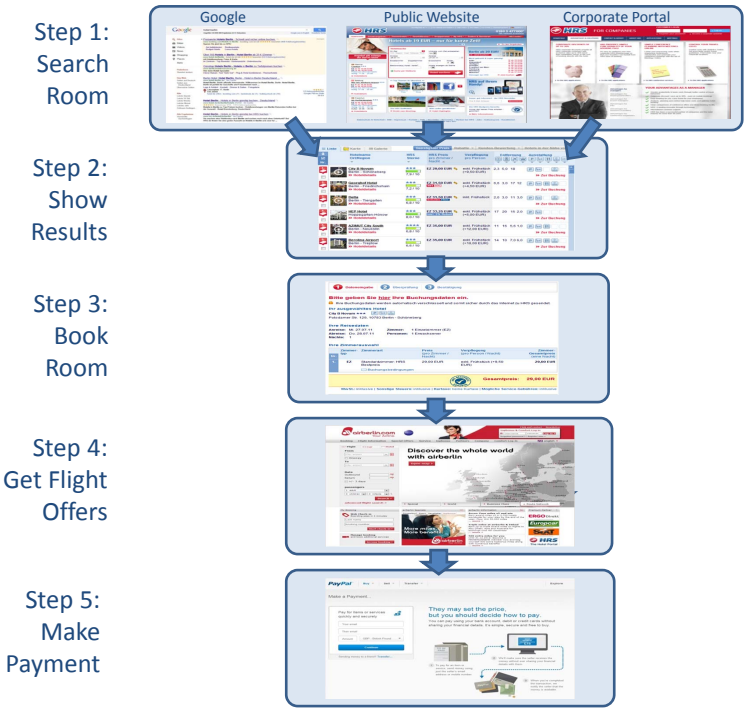

Figure 1.5: Typical Booking Scenario at HRS

A real-life application for OTF Computing comes from our industrial partner *Hotel Reservation Service (HRS)*. HRS is a worldwide company that provides accommodation booking facilities to its B2B and B2C users by interacting with the services of its 230,000 hotel partners world wide. Additionally, HRS also wants to extend its features by allowing the end-user to search and book the possible transport options to his destination.

Such a typical trip planning scenario at HRS is depicted in Fig. 1.5.

This scenario is comprised of the following steps: In the first step, the end-user searches for available hotel rooms through three different options, i.e., Google search engine, HRS website[2] and corporate portal for corporate clients of HRS. In the second step, the search results are returned by the partners' hotel services based on the specified search criteria and the end-user is directed to these results on the HRS website. In the next step, the selected room is booked for the end-user on the basis of his credentials and a confirmation is sent. After hotel booking, the end-user has the facility to receive flight offers from the partners' airline services to his destination. For the selected flight offer, a booking is made and an online payment is carried out for the booked flight.

This booking scenario is envisioned as an end-to-end use case by a technical personnel at HRS called *HRS application developer*. With his technical expertise in model-based system design, the HRS application developer is responsible for an overall design of the HRS application and has a complete view of the system from a higher abstraction level. To realize his designed booking scenario for HRS, HRS application developer aims to search and compose the suitable services available on the service market and integrate the resulting service composition with the HRS application.

In the context of SOC in general and OTF computing in particular, the HRS application developer acts as the service requester that accesses the service market with his request to search and compose the suitable services published by the service providers. These service providers are also technical personnel who are skilled to model, program and publish these software services. Fig. 1.6 shows how the service discovery and composition for HRS is performed in an OTF Computing setting. In Step 1, the requester accesses the service market with its service request comprising of the required operation signatures. For the e-tourism domain, the service repository on the service market contains a variety of services published through their service offers, e.g., services for hotels and hotel chains, online payment services, airline services, train services, etc. In a conventional SOC setting without OTF Computing, the requester usually have the option to select some of the published services on the basis of keyword-based search or by manually matching the requested and offered operation signatures. Later, he has to communicate personally with the selected service providers through informal behavioral descriptions of the required functionality to determine whether these services offer the same behavior as required by the requester or not. However, this manual matching approach to discover available services based

[2] http://www.hrs.com

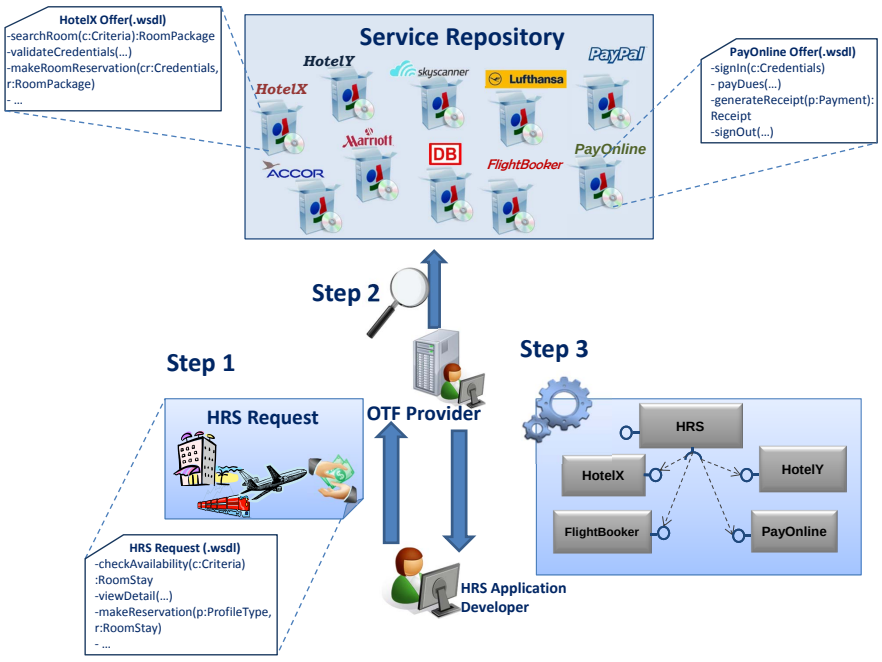

Figure 1.6: Service Discovery and Composition for HRS in OTF Computing

on structural service descriptions is a time-consuming and error-prone process.

OTF Computing aims at overcoming this problem where the OTF provider as the mediator automates the service discovery and composition process and enables seamless interaction between service partners. In Step 2, an automatic service description matching mechanism matches the service request to the available service offers. For this purpose, the OTF provider may use some existing *lexical* matching techniques for matching the wsdl-based structural service descriptions [147, 150]. However, such an automation can lead to inaccurate results. For instance, based on a lexical matching of the operation names and their input/output parameters, the requested operation *makeReservation(...)* is matched to the offered operation *makeRoomReservation(...)* in *HotelX* service offer. However, the behavior of the two operations can be completely different. For instance, the requested operation *makeReservation(...)* validates the client credentials and reserves a room for him and sends him a reservation notification at the end. On the other hand, *HotelX* specifies its service offer at a different granularity level and *makeRoomReservation(...)* does not validate the client credentials but is defined to book the room only. Hence, the matching results based

on an automatic matching of the structural aspects of the request and the offer are inaccurate, as their requested and offered behavior is totally different. Considering this situation, it is also important to take into account the behavioral description of the offered/requested operations in addition to matching their structural features. Conforming to the earlier discussion about the challenges faced for the automation of service discovery and composition process, OTF Computing has to come up with techniques for comprehensive service descriptions in case of the HRS scenario.

Additionally, the OTF provider also has to deal with the underlying heterogeneity of the HRS and the published services, which makes the automatic matching more complex. For instance, as evident from the types of input/output parameters, the underlying data model of the service partners are heterogeneous, e.g., `RoomPackage` in HotelX offer corresponds to `RoomStay` in HRS request. Such data model heterogeneity needs to be overcome to correctly match the respective service descriptions.

Additionally, the service partners have defined their service descriptions at different granularity levels leading to complex correspondences between the requested and offered operations. For instance, in the example mentioned earlier, a 1 : n operation correspondence occurs between the requested operation *makeReservation(...)* of HRS and two offered operations of *HotelX validateCredentials(...)*→ *makeRoomReservation(...)*.

Similarly, it is also highly likely that different service partners in this setting use different service description languages according to their requirements and skill sets. Hence, the challenge of linguistic heterogeneity has to be overcome by the OTF provider while matching the service descriptions.

As a result of the service discovery, the OTF provider determines multiple services that have to be composed to completely fulfill the HRS request, i.e., *HotelX* and *HotelY* provide the hotel booking facilities. Similarly, *PayOnline* offers the required payment functionality. Additionally, *FlightBooker* fulfills the transport-related requirements specified in the request. Consequently in Step 3, the OTF provider automatically creates a service composition based on the selected individual services and returns it to the requester.

Based on our example scenario introduced in this section, we will state the problem and the requirements for a potential solution in the next section.

1.1.2 Problem Statement and Solution Requirements

To fully exploit the potential of the SOC paradigm, OTF Computing aims at devising methods and techniques that can enable the service partners

to collaborate seamlessly while functioning in their independent domains with their respective domain knowledge. To that extent, an automatic service discovery and composition mechanism is required that ensures accurate results. The main goal of this thesis is to devise such a mechanism.

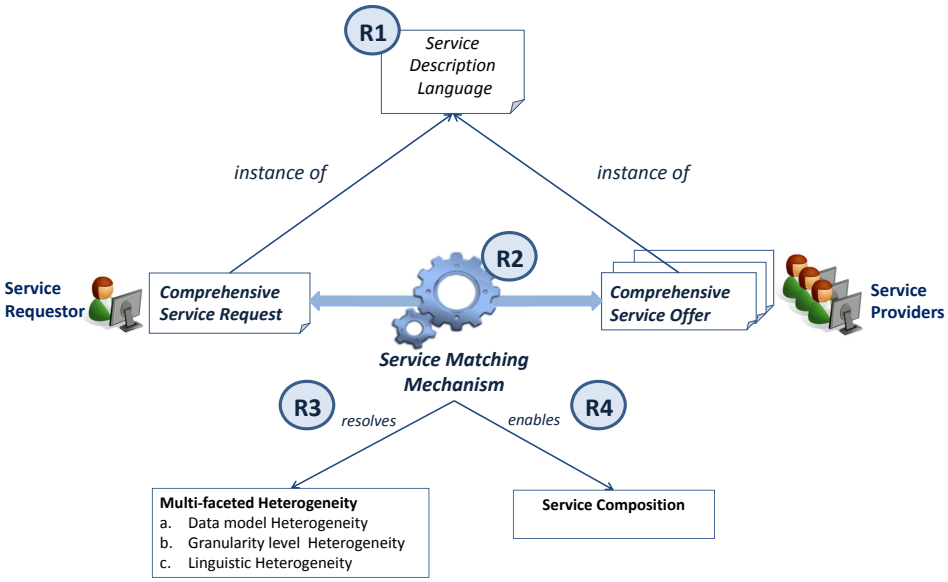

Figure 1.7: The Potential Solution and its Requirements

For the development of such a mechanism, there are certain requirements that need to be met, which are diagrammatically represented in Fig. 1.7:

R1: A *service description language*, which allows comprehensive description of a requested/offered service in terms of its structural as well as behavioral aspects.

R2: An *automatic service description matching mechanism* that matches such comprehensive service requests and offers.

R3: A *resolution of the underlying multi-faceted heterogeneity* of the service partners, i.e., data model heterogeneity, granularity level heterogeneity and linguistic heterogeneity, etc. while matching their service descriptions.

R4: Enabling *service composition*, i.e., composing multiple offers if a single service offer does not completely satisfy the request at hand.

In the next section, we give an overview of our proposed mechanism for automatic service discovery and composition.

1.2 Proposed Solution Overview

In order to automate service discovery and composition process in OTF Computing, we have proposed a framework that overcomes the challenges faced by OTF Provider and fulfills the requirements specified in Sec. 1.1.2. Fig. 1.8 gives an overview of the proposed solution.

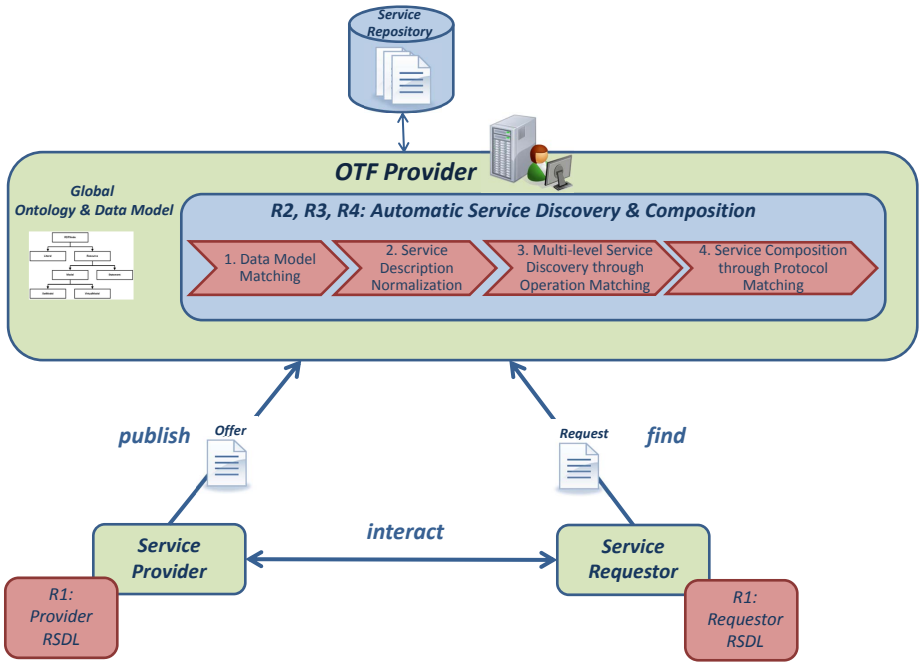

Figure 1.8: An Overview of the Automatic Service Discovery and Composition Mechanism of the Proposed Approach

First aspect of this solution is the *Rich Service Description Language (RSDL)*, which allows the service partners to comprehensively describe their service requests and offers in terms of its structural and behavioral aspects (satisfying R1). A service requester accesses the service market with his RSDL-based service request to search for suitable service offers. As a result, the automatic service discovery and composition mechanism of OTF provider is initiated (satisfying R2, R3 and R4). This mechanism comprises of multiple steps.

In the first step, requester's data model is automatically mapped to a global data model conforming to a global ontology. This global ontology and the conforming data model define a common representation of an informa-

tion domain. The local-global data model matching serves as the basis for the resolution of the data model heterogeneity while matching the request and available service offers.

In the second step, the request is *normalized* to a common global representation on the basis of local-global data model mappings achieved in the first step. These two steps are also performed for a service offer when a service provider accesses the OTF Provider to publish his service on the service market hence resolving the data model heterogeneity of the service partners.

In the third step, a multi-level service discovery is performed to discover the potential service offers that completely or partially satisfy the requirements specified in the service request. The main focus of this service discovery mechanism is the matching of operations in the service request and the available offers on the basis of their comprehensive descriptions, i.e., their structural elements as well as their behavioral semantics in terms of preconditions and postconditions.

In the last step, it is determined whether a single service offer can completely satisfy the service request or a service composition is required. This is based on a protocol matching between the request and the selected service offers.

In the following, we give a brief description of the individual components that realize the above-mentioned aspects of our approach and how they fulfill the requirements specified earlier:

- **Rich Service Description Language (R1):** One of the major research concern in SOC is a comprehensive service description language to enable automatic matching of service descriptions and achieve accurate service discovery results. An important factor in this regard is that these languages should be widely acceptable in practice. As mentioned earlier, most of the languages proposed so far have certain shortcomings in this direction. Particularly focusing on a specification of functional properties, we propose a *Rich Service Description Language (RSDL)* that provides a detailed set of UML-based notations to service partners to comprehensively specify their service descriptions.

- **Service Description Normalization through Local-Global Data Model Matching (R3(a)):** In order to overcome the data model heterogeneity of the service partners, we propose a local-global data model matching mechanism, which serves as the basis for the normalization of the service description to a common representation. In this direction, our approach extensively relies on a global ontology and

the conforming global data model, which formally describes a domain in a standardized way for all the stakeholders of that domain. The OTF provider is supported through guidelines to define and maintain such a global ontology and the conforming data model. When a service partner accesses the OTF provider to publish his service offers or to initiate a service discovery, his data model is automatically matched to the global data model on the basis of the ontological semantics defined through the global ontology.

Our approach uses the deduced data model mappings to automatically translate the heterogeneous service descriptions of the service partners to a standardized representation typed over the global data model. Hence, this overcomes their data model heterogeneity and enables their automatic matching in the later stages of the approach. This process of translation from a local to a common global representation is called service description normalization.

- **Multi-level Service Discovery based on Operation Matching (R2, R3(b)):** After the normalization of a service request on the basis of data model mappings, the automatic matching of the request and available offers is initiated through a multi-level service discovery mechanism. On the first level, a subset of available service offer is selected on the basis of the category matching between the request and the offers. However, the focus of this service discovery mechanism is operation matching between the request and the selected offers. For this purpose, the requested and offered operations are matched on the basis of their RSDL-based comprehensive descriptions. These comprise of their structural elements and their behavioral semantics specified through their preconditions and postconditions. As mentioned earlier, the granularity level heterogeneity among the service partners can lead to 1 : n, n : 1 and n : m operation correspondences and an operation matching approach must be able to detect such complex correspondences. In our approach, we carried out an in-depth study to identify different scenarios of these complex correspondences. Consequently, our approach precisely define the structure of such complex correspondences and proposes a set of operation matching strategies to determine these correspondences between a request and the available offers. On the basis of the operation correspondences determined in this step, a subset of available service offers is selected as potential candidates to satisfy the service request. A further analysis of this subset is carried out in the next step.

- **Service Composition based on Protocol Matching (R2, R3(c), R4):** Another important aspect of a behavioral description, which needs to be taken into account while service matching is the required and allowed operation invocation sequences specified as service protocols. For this purpose, our approach comprise of a protocol matching mechanism, which matches the requested and offered protocols on the basis of complex operation correspondences determined earlier. This mechanism also allows a composition of service protocols and hence determine possible service compositions if a single service offer fails to satisfy the service request.

  However, the proposed protocol matching mechanism has to overcome the linguistic heterogeneity as RSDL offers different notations to service requester and provider for the specification of their service protocols according to their respective needs. Therefore, the proposed mechanism enables an automatic translation of the linguistically heterogeneous service protocols to a common semantic domain, i.e., Labeled Transition System (LTS) before matching. Later, a *specialized LTS composition operator* is used to match and compose the LTSs on the basis of determined operation correspondences and determine possible service compositions.

1.3 Publication Overview

Most of the contributions presented in this thesis have been published as peer-reviewed papers at various international conferences. Fig. 1.9 gives an overview of these publications. In this figure, the publications are categorized according to the particular component of our framework that they focus. Below, we briefly describe each of these publications.

In [75], we presented our rich service description language (RSDL), which allows a comprehensive description of service requests and offers in terms of their structural and behavioral aspects. This paper was presented at 24th International Conference on Advanced Information Systems Engineering (CAiSE'12).

Our automatic service discovery mechanism for RSDL-based service descriptions, which acknowledges the granularity level heterogeneity of the service partners and allows complex operation correspondences was published in the proceedings of ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems (MODELS'12) [74].

In [76], we published our automatic service composition mechanism,

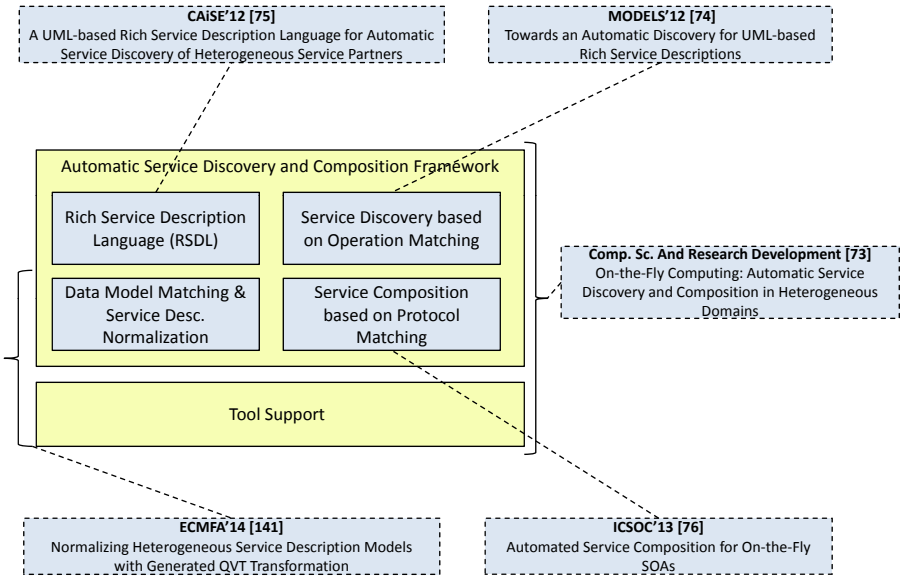

Figure 1.9: An Overview of the Publications

which is based on the service discovery mechanism and determines possible service composition by resolving the linguistic heterogeneity of the service protocols and composing them. This publication was presented at 11th International Conference on Service Oriented Computing (ICSOC'13).

For the resolution of data model heterogeneity, our service description normalization mechanism based on data model matching was published in the proceedings of the 10th European Conference on Modeling Foundations and Applications (ECMFA'14) [141]. In this paper, we also presented our tool support for data model matching and its detailed evaluation.

Our paper [73] published in Journal on Computer Science - Research and Development (CSRD) in 2014 gave an overview of the CRC 901 *On-the-Fly Computing* and in this context, presented our complete framework with detailed insight into its different components. Additionally, it also described in detail our prototypic implementation of the framework in terms of its different aspects.

In the next section, we give an overview of the thesis structure.

1.4 Thesis Structure

In this section, we provide an overview of the structure of this thesis, which is given in Fig. 1.10.

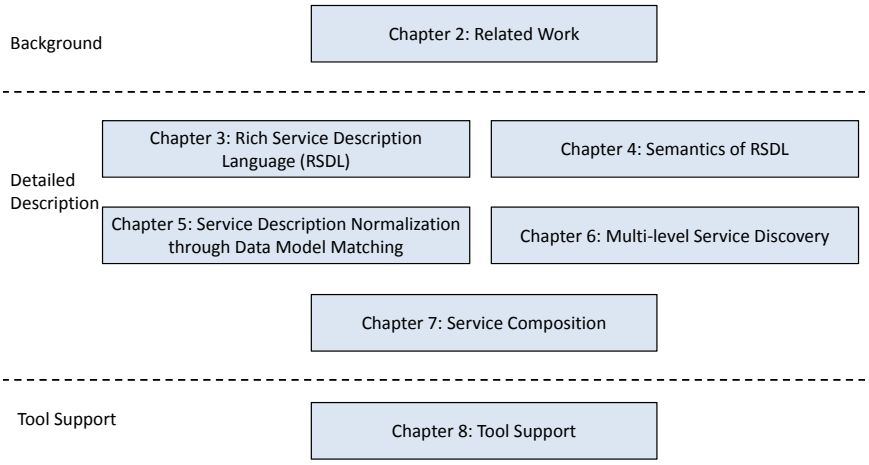

Figure 1.10: An Overview of the Thesis Structure

In Chapter 2, we build a background for our work by analyzing the existing work in the related areas. In this chapter, we give an overview of the SOC domain and the major milestones of automatic service discovery and composition. In this direction, the main concerns of comprehensive service descriptions and automatic service matching are investigated in detail in terms of an overview and evaluation of the existing works. In the following chapters, we give a detailed description of different parts of our proposed approach. Our proposed approach comprise of two main parts: A comprehensive service description language and an automatic service discovery and composition mechanism. Chapter 3 and 4 focus the first part of our approach and introduce the Rich Service Description Language (RSDL). In Chapter 3, we give a detailed insight into the structure of RSDL and its offered notations to specify comprehensive service requests and offers. In Chapter 4, the semantics for different RSDL artifacts are defined.

The second part of the proposed approach, i.e., automatic service discovery and composition mechanism is discussed in detail from Chapter 5 to Chapter 7. These chapters describe different phases and activities of our mechanism in detail. Chapter 5 covers the details of service description normalization based on data model matching. This normalization is meant to translate the service descriptions to a common representation while

overcoming their data model heterogeneity. Hence, as the basis for the service description normalization process, we propose an automatic data model matching mechanism, which is elaborately explained in Chapter 5. In Chapter 6 and 7, we discuss our service discovery and composition mechanism, where Chapter 6 mainly focuses the operation matching approach. In this context, the set of our proposed algorithms for complex operation matching are explained in detail. Chapter 7 covers the aspect of linguistic heterogeneity resolution in our approach in detail. It also describes the proposed technique for service composition based on service protocol matching and composition.

In Chapter 8, we discuss the tool support for our approach and introduce the implemented research prototype called the Service Discovery and Composition Workbench. The thesis is concluded by Chapter 9 where the results are summarized and potential future research directions are mentioned.

2
Related Work

In this chapter, we build the necessary background for our work presented in this thesis by carrying out a detailed examination of the existing related work. In this direction, we begin with a general introduction of service-oriented computing (SOC), its emergence and the recent growth. Later, we describe an important milestone of SOC, i.e., automatic service discovery and composition. This is followed by an in-depth account of two major areas that the researchers focus in this context: definition of service descriptions and automatic matching of these service descriptions. For these two areas, we identify the salient features that later serve as the criteria to classify the existing work. Then, we give an overview of the prominent work in these areas and evaluate them on the basis of the criteria introduced earlier. Finally, we present a summary and a discussion of this chapter.

2.1 Service-oriented Computing (SOC)

Recently, the Internet has spawned many changes in the way businesses are conducted and has completely reshaped the business world. It has infiltrated the daily lives to an extent that customers are focusing their daily life activities around web-based applications. Similarly, the advent of a whole new series of web-based technologies allowed organizations to shift their focus and exploit the opportunities provided by electronic market places. This also means constantly dealing with rapidly changing market conditions and coping with the factors such as ever-evolving customer requirements and new business processes. To achieve a competitive advantage through organizational agility, the organizations aim at constant modification and improvement of their enterprise IT system. This leads to an on-going addition of new capabilities in terms of software units and their integration with the existing enterprise system.

The recent growth of service-oriented computing (SOC) paradigm

promises to come up with potential solutions for these challenges. It achieves this goal by meeting some important requirements [94]: seamless integration of applications and resources across enterprises, homogeneous access to applications regardless of their implementation details, a logical architecture to manage computing resources, etc.

In this context, the concept of *services* as the basic building blocks has revolutionized the software development. With the recent technological breakthroughs in the area of smart devices, the growth of services has gone up to a dramatic level. The development and deployment of services can vary from an infrastructure facility to a software component. This claim is supported by the emergence of concepts like software-as-a-service (SaaS), infrastructure-as-a-service (IaaS) and platform-as-a-service (PaaS). The resulting benefits are not only reaped by the conventional stakeholders like E-commerce but other mainstream segments, such as, transportation, public sector, manufacturing, and banking, etc. are also following the suit. This claim is confirmed with the emergence of services for tasks, such as, loan originations and servicing, shipments track-and-trace services, healthcare, and water utility monitoring, etc.

For our work, our particular focus is on the software services, which are referred to simply as *services* during the course of this thesis. The idea of remotely consuming software components over networks has a long standing tradition in SE with initiatives like RPC [17], CORBA [18], etc. This idea evolved itself into web services [3] as specified in [160]: *"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically web service description language (WSDL) [162]). Other systems interact with the Web service in a manner prescribed by its description using SOAP [144] messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."* Since then, the conceptual and technological developments in this direction [111, 165] have further highlighted the benefits of services and SOC.

In a nutshell, software services are independently developed software computational units that can perform different functions varying in their degree of complexity. These services are published on the service markets on the basis of their service descriptions and can be discovered, composed, and consumed over networks resulting in large-scale distributed, interoperable and evolving IT systems. Typically, such services are developed independently without a preconception of the context they will be used in. Consequently, this notion culminates the idea of reusability and loose coupling pattern, which is a basic aim of distributed software development.

In this context, an important milestone is the automation of the service discovery and composition process. In the next section, we give a general overview of this important milestone of SOC and its different challenges.

2.2 Automatic Service Discovery and Composition

In order to take full advantage of the benefits offered by the services and the resulting SOC paradigm, an important research concern is to automate the service discovery process while ensuring accurate results. However, considering the reusability and loose coupling aspects of SOC, it is not practical to assume that a single service offer can fulfill *all* the requirements of the service requester. Normally, in realistic scenarios, more than one services have to be composed to completely fulfill a service request. This aspect is already a relevant topic of research for the SOC community [131, 9]. This means that the service discovery approaches should not only devise methods for automatic discovery of the potential service offers that completely/partially satisfy a particular service request but should also come up with mechanisms to compose these services in order to completely fulfill the requester's requirements.

To achieve this milestone of automatic service discovery and composition with accurate results, there are certain areas that need to be investigated in detail. For such a detailed investigation, it is important to outline certain relevant details of the particular scenario of OTF computing considered in this thesis. In the given scenario, the service providers develop their services and publish them on the service market through their service offers. In this case, a service provider is visualized as a person with strong technical know-how and programming skills to design and develop a SaaS and publish it on the service market. With the emergence and wide acceptance of model-driven engineering (MDE) paradigm [22], we assume that the service provider is also familiar with model-based design and development techniques. In his respective service offer, the service provider aims at describing the offered functionality in detail in terms of all the possible usecases that can be invoked on the offered service.

On the other hand, the service requester in the given scenario accesses the OTF provider to initiate an automatic service discovery process in order to fulfill its requirements specified in his service request. In this case, the service requester is not a normal web user with minimal technical skills. Rather, he is visualized as a system architect who is responsible for designing and building a system on the basis of the underlying services discovered from

the service market. For this purpose, we assume that the service requester is well-equipped with model-based design and architecture skills and has an overall view of the system at a higher abstraction level. Based on these skills, the service requester specifies his service request by describing the particular use case that needs to be realized in the designed system on the basis of available services.

On its invocation, automatic service discovery mechanism of the OTF provider determines any possible service offer that is able to fulfill these requirements and realize the required use case of the service requester. In case, a service offer does not completely satisfy these requirements, possible service compositions comprising multiple service offers are determined.

In context of this scenario, the first area that need detailed investigation is the comprehensive specification of the required and offered functionality in service requests and offers. Second area of investigation is the service description matching mechanism that enables automatic and accurate service discovery. In the next two sections, we discuss these aspects in detail and give an overview of the work done in this direction so far.

2.3 Service Description

As mentioned in Sec. 2.1, SOC has enabled a major shift from everyday services to software services. This shift is based on the core idea of publishing of services by the service providers on the service market and in turn, the discovery and consumption of these services by the service requester. Analogous to the concept of print and media advertisements for the everyday services, the publishing and the discovery of the software services is carried out on the basis of the offered and requested service descriptions called *service offers* and *service requests*, respectively.

The correlation between the conventional and software service descriptions is precisely described by O.Sullivan [120] as follows: *"The everyday services that surround us, and the way in which we engage with them, are the result of social and economic interaction that has taken place over a long period of time. An attempt to provide automated electronic services that ignores this history will deny consumers the opportunity to negotiate and refine over a large range of issues, the specific details of the actual services to be provided."* Keeping this argumentation in view, a requested/offered software service can be described as a combination of the requested/offered functionality together with the constraints, obligations, penalties and the rights of the involved service requester and the provider. Hence, a service

request/offer not only has to encompass the functional aspects of the service but also has to consider its non-functional aspects.

In a broader perspective, service descriptions can be compared to the software component specification, which is extensively investigated in the area of *software component models* since many years [154, 90, 36]. An extensive study and classification of the existing component models has been presented in [90, 36]. In order to describe the features of a component to the outer world, these component models come with the component description languages to define the interfaces of a component. We particularly analyze these models on the basis of their component description languages.

With the advent of SOC, the concept of *component* is interchanged with that of *service* and the researchers have put more focus on the specification of services and their properties in detail [120]. For our work, we are interested to explore how these existing approaches enable a precise specification of service descriptions comprising different aspects of the requested and offered service. Hence, we consider the approaches for component as well as service descriptions because of their similar purpose as the terms *service* and *component* are used interchangeably in the literature.

In the coming sections, we first introduce a classification scheme and later analyze the existing service description languages in detail on the basis of this classification.

2.3.1 Classification of a Service Description Language

With the current focus on the elaborate service descriptions to enable automatic service discovery and composition, the researchers have come up with a variety of languages for the specification of different aspects of a service. In general, the service description languages can be classified on the basis of different criteria. In the next section, we introduce the set of selected criteria that is relevant for our work and that is later used to classify the existing service description languages.

Comprehensiveness Level

A service partner specifies his service description to describe his requested/offered functionality to the outer world. The level of detail about the requested/offered functionality specified in the service description can vary according to the technical expertise and preferences of the service partner. However, this level of comprehensiveness of the service description directly effects the accuracy of an automatic service discovery process. With

comprehensive service descriptions, the matching mechanism has more information about the requested and offered functionality resulting in precise matching as compared to the matching of less comprehensive service descriptions.

Consequently, a comprehensive service offer will increase the chances that the published service will be correctly matched, discovered and invoked to fulfill the relevant service request. Additionally, for the service requester it is also important to precisely describe the requirements in order to discover correct services. Hence, the first criteria to classify service description languages is their allowed level of comprehensiveness in which they describe different aspects of the requested/offered service.

According to the existing studies [154, 120], a service can be broadly described on the basis of its functional and non-functional aspects, where the *functional* aspects of a service are focused on its operational details in terms of its inputs, behavior, outputs, calculations, binding protocols and similar technical details. On a more refined level, these functional aspects can be further classified in terms of the service's *structural* and *behavioral* details [133]. Similarly, the *non-functional* aspects of the service cater to the non-operational details that are necessary to develop a business contract between the requester and the provider, such as its pricing, availability, quality-of-service (QoS) and the related legal obligations, etc.

These varying degrees of comprehensiveness of the service description languages is visually described in Fig. 2.1.

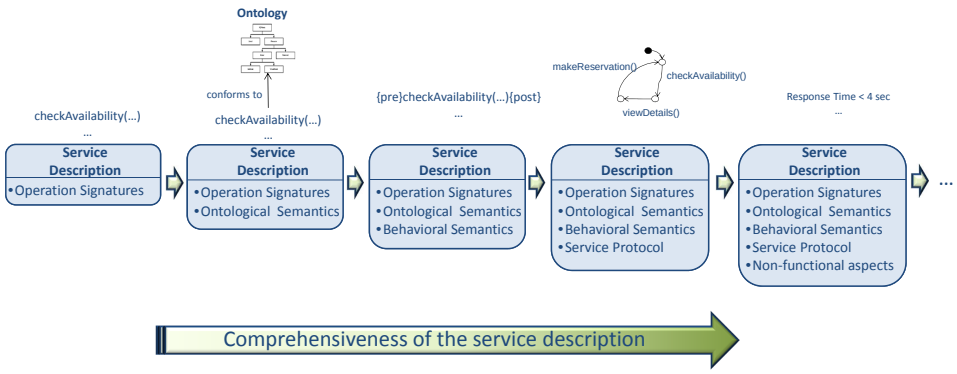

Figure 2.1: Varying Degrees of Service Description Comprehensiveness

The basic service description describes a service in terms of the structural elements of its requested/offered functionality. In recent years, WSDL [162] has emerged as a standard for such structural service descriptions describing

a service in terms of its requested/offered operation signatures.

To make these service descriptions comprehensive and meaningful, one possibility is to define the *ontological semantics* for the service description in order to clearly specify the meanings of the terms and the concepts that it uses on the basis of an underlying *ontology*.

According to Studer et al. [148] *" An ontology is a formal, explicit specification of a shared conceptualization. A 'conceptualization' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. For example, in medical domains, the concepts are diseases and symptoms, the relations between them are causal and a constraint is that a disease cannot cause itself. 'Formal' refers to the fact that the ontology should be machine readable, which excludes natural language. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group."* Development and usage of ontologies for semantic interpretation in the software development field has developed into an extensive trend in recent years. In practice, there is no fixed interpretation and representation for ontology, and different artifacts and terms that represent an information model are used interchangeably as ontology [52], e.g., taxonomy, schema, information model, etc. Therefore, the notations for ontology specification range from Database [104, 130], via XML Schema [118] to formal ontology languages like RDFS [159] and OWL [158]. Similarly, the types of ontologies used in practice range from general-purpose ontologies, like NAICS [155], WordNet [130], etc. to domain specific ontologies like GoodRelations [68], OnTour [41], etc.

The ontological semantics of the service description elements are defined by annotating them with the corresponding semantic concepts in an underlying ontology. In case of structural service descriptions, such ontological semantics can be defined for the operation names, input and output parameters. In recent years, the shift of service description languages towards such ontological semantics [121, 51, 39, 98, 136] have validated the inadequacy of structural service descriptions.

Another perspective to add more meaning to the structural service descriptions is by describing the behavior of the required/offered service. The behavior of a service can be described from two different perspectives: functional and temporal. The functional behavior describes its functionality in a static manner at particular *snapshots* during execution in terms of the system states that hold before and after the invocation [63, 133]. For an operation, such a functional behavior is termed as its *behavioral semantics*.

The temporal behavior gives a continuous view on a service by specifying different states that it goes through as a result of the invoked transitions in a particular sequence [133]. This continuous view can be *internal* specifying the internal execution details, the data flow and the states changes of the service or it can be *external* that represents the required/allowed invocations, the resulting transitions and data flow and the state changes visible to the outer world. In the particular context of service descriptions that are meant to describe the required/offered functionality to the outer world, the specification of external dynamic behavior is more relevant. This external dynamic behavior is described in terms of the required/allowed operation invocation sequences known as *service protocol*.

Such comprehensive functional service descriptions can be further enhanced by the addition of non-functional aspects, e.g., response time, cost, reliability, reputation etc. of the requested/offered service. For our work, we are particularly focused on *comprehensive functional service descriptions* and hence do not go into the detail of the specification of non-functional aspects in the service descriptions.

Ease-of-Use

An important criteria that contributes to the selection of a service description language is its ease-of-use. This means that a service description language is preferred by the service partners if they find it easy to adapt to in terms of time and resources. In the context of the OTF computing scenario discussed earlier, different factors can be determined that contribute to the ease-of-use of a service description language.

From the perspective of a service provider, an important factor to select a service description language is the conformance to existing standards. A service offer in a standard language can be published in the commercially available service registries, which are based on prevailing standards. Additionally, majority of the service requesters and the service matching mechanisms are likely to be familiar to the standard description language thereby increasing the chances of the service offer to be discovered and consumed by a larger audience.

This means that a language conforming to standards can by easily accepted and adapted in practice by industry as compared to some research initiative with new and unfamiliar notations limited to the academia. A suitable example for this case is the service description language SAWSDL [136], which is an acclaimed initiative for semantic service descriptions and is based on the prevailing service description standard WSDL [162].

From the perspective of the service requester, another important factor that contributes to the ease-of-use of a specification language is its mode of specification, i.e., visual vs. textual. In the context of given scenario where the service requester is the system architect with model-based design and architecture skills, it is preferable to have visual notations for service descriptions, which are believed as a comparatively effective, efficient and unambiguous mean of designing and engineering a component-based system [6, 40]. Visual languages form an integral part of the research in the SE domain and the most promising initiative in this direction is UML [115], which is an industry standard for software modeling and specification. Carrying this trend forward to SOC domain, a visual service description language is likely to be easier to use and promptly adapted by the service partners as compared to a language with complex textual notations.

Request/Offer Distinction

In the context of CBSE, a component specification comprises of its *provided* and *required interfaces* [99], which specify the provided and the required functionality of that component, respectively. A communication between two components is only possible if their respective required and provided interfaces match. This discrimination between different interfaces has been carried forward to the SOC domain where the requested and offered functionality is specified as service request and offer, respectively.

The essence of SOC is to allow the service partners to have their individual contexts with different levels of technical expertise. This is also evident from the OTF computing scenario at hand. In such a case, the service requester and the provider have their individual needs as far as the specification of service request and offer is concerned. For instance, according to the profile of the service requester in the given scenario, he prefers visual notations to specify his service request. Additionally, it is important for the service requester to precisely describe the particular usecase that it wants to realize on the basis of discovered services.

On the other hand, with the vast number of service offers available on the service markets and the thriving competition among the service providers, it is particularly important for a service provider to add enough information in his service offer in order to increase its chances of being discovered and consumed by a suitable service requester. Therefore, comprehensive notations for service descriptions are particularly important for service provider. In this direction, his aim is to describe all the possible usecases that can be realized through his offered service.

Considering such individual needs and concerns, it is important that a service description language considers and caters to these requirements of the service partners and allows to specify the service requests and offers accordingly.

2.3.2 Overview of Service Description Languages

There is a broad spectrum of existing related work for service specification languages. As mentioned earlier, a plethora of research work in this direction already exists in the context of CBSE and component models. In the following, we first give a brief overview of description languages developed in the area of CBSE. Later, we will discuss in detail the description languages that are developed particularly in the context of SOC for service descriptions.

In the area of component models, we refer to the comprehensive classification presented by [36]. In this direction, we only consider the general-purpose component models that consider an *operation* as the basic unit of the component description.

The first set of component models under discussion are those based on a particular technological platform and hence rely on the underlying programming language to specify the structural interface description for the components. EJB [47], Java Beans [149], OSGI [119] are examples of such component models that heavily rely on Java programming language and the components are described through interfaces defined in Java. Similarly, the component models, such as, RUBUS [62] specify C-language interfaces to describe the functionality of the components.

In order to allow communication between heterogeneous components by defining language-independent interfaces, the researchers presented the *Interface Description Language (IDL)* approach, where mappings are defined between IDL and different programming languages in order to translate the IDL-based request sent to a component to its particular programming language and vice versa. In this context, the Object Management Group (OMG) IDL is proposed as a part of the Component Object Request Broker Architecture (CORBA) component model [113] and allows the specification of language-independent structural interface descriptions. As an extension, CompoNETS [10] additionally allows a specification of the component behavior through protocols specified as Petri Nets.

Component models like Fractal [45] and Software Appliances (SOFA) [28] also present their respective IDLs for the specification of structural interface descriptions, which are mapped to the underlying implementation languages. Additionally, these models come up with their particular no-

tations for the specification of behavior protocols. A variation of IDL is proposed in the Microsoft Component Object Model (MSCOM) [21], which is one of the most commonly used software component model. It works on the principle that the interfaces are specified separately from the components that implement and use them. Hence, its interface specification language aims at reducing the distance between the IDL and programming languages by introducing the object-oriented interfaces, which are defined independent of the components. Later, these interfaces are directly implemented by the components.

With the emergence of the Unified Modeling Language (UML) [115] as a de facto industrial standard for multi-purpose modeling, component models have also attempted to gain from its benefits. UML provides an extensive set of visual notations that holistically encompass different aspects of component-based systems. Component models, such as, Palladio [12], KobrA [5] and Pin [70] are some prominent ventures that have used UML notations for component interface description. KorbA [5] is the conceptual component model that defines a UML profile for different aspects of component modeling. It uses the UML concept of *interface* to specify the structural interface descriptions. Palladio [12], which particularly aims at specification and evaluation of performance aspects of the components also define such UML-based structural descriptions and use Service Effect Specifications (SEFFs) to specify the protocol of the components [132]. Pin [70], a component model for prediction-enabled component technologies defines the Component Composition Language (CCL) for interface descriptions, which specifies operation signatures and UML statemachine-based protocols.

In the area of SOC as well, service description languages are an important research concern. Below, we discuss some salient initiatives for service description languages particularly in the SOC domain.

Carrying the idea of IDL for component descriptions forward to the SOC domain, the W3C consortium proposed the Web Service Description Language (WSDL) [162] for service descriptions, which has established into a standard. WSDL is an XML-based IDL to describe the interface of a service as a collection of its operations with their structural details.

The concern for having comprehensive service descriptions is not new in SOC. An important initiative in this direction was the semantic web services (SWS) trend comprising the comprehensive service description approaches that mainly focus the use of ontologies for the definition of semantics. Semantic annotations for WSDL (SAWSDL) [136] is a SWS language produced by the W3C working group, which is a revised and restricted version of WSDL-S [98]. SAWSDL extends the WSDL standard through the spec-

ification of the ontological semantics of different WSDL elements, such as, interface, port, operation, etc. Each of these elements can be annotated with the concepts in the underlying ontology through the `modelReference` element. Although in WSDL-S, elements like `precondition` and `effect` were proposed for the specification of behavioral semantics, these elements are not carried forward to SAWSDL due to lack of agreement among the community.

Yet Another Semantic Annotation for WSDL (YASA) [30] was proposed to cover this gap, which allows a more comprehensive service description including preconditions, postconditions, effects, and other aspects for different elements of a WSDL description. A web service ontology is used to model all possible types of aspects that can be specified for different elements of WSDL. For every element in a WSDL description, a YASA description specifies its different aspects through a combination of `ServiceConcept` and `ModelReference` element, which refer to the particular aspect through service ontology and its annotated concept in the domain ontology, respectively.

OWL-S [121], earlier DAML-S, is one of the most promising SWS incentives by Semantic Web Service Initiative (SWSI). It is based on the Web Ontology Language (OWL), which is a standard for ontology definition. The service is described in OWL-S from two different perspectives: a high-level perspective and a detailed perspective. The high-level perspective in `ServiceProfile` element describes a service as an atomic functional unit in terms of its inputs, outputs, preconditions and effects. For a detailed description, OWL-S describes the requested/offered functionality through its `ServiceModel` element, where the individual operations are specified as *atomic processes* with their structural elements, i.e., `inputs`, `outputs` and the behavioral semantics, i.e., `preconditions`, and `effects` (IOPEs). The ontological semantics are specified through an underlying OWL ontology and the behavioral semantics can be represented as logical expressions. In this direction, OWL-S does not provide any concrete language and leaves this choice for the user who can use rule languages, such as KIF [156], SWRL [72], etc. *Composite processes* in OWL-S is aimed for process specification, which can be the requested/offered service protocols of the individual services as well as a service composition with interactions among the participating services. Special constructs are defined to specify the control and data flow in these composite processes. Although OWL-S diverges from the existing standard WSDL, its atomic process notion directly correspond to the WSDL operation, which allows a direct mapping between the elements of OWL-S to WSDL through its *ServiceGrounding* element.

Universal service description language (USDL) [8] is another OWL-based

initiative that aims at adding semantics to the WSDL structural service descriptions. For this purpose, an OWL-based representation of the WordNet Ontology [130] is constructed and the ontological semantics for WSDL elements, such as, ports, operation, and messages are defined on the basis of this WordNet ontology through the `Concept` element of USDL. Additionally, USDL defines the elements `Condition` and `affects` for the behavioral semantics of WSDL elements based on the concepts in the WordNet Ontology. For instance, an atomic condition is a binary predicate on concepts and several atomic conditions can be combined to have complex conditions. Similarly, affects can be of kind `creates`, `updates`, `deletes`, and `finds` on one or more concepts.

Web Service Modeling Language (WSML) [39] is a service description language, developed at DERI institute Innsbruck as a part of the Web Service Modeling Framework (WSMX). It realizes the core conceptual model for service description named as Web Service Modeling Ontology (WSMO) [51]. WSML/WSMO differentiates between the specification of service offer and request and defines them as `web service` and `goal`, respectively. Like OWL-S, WSML/WSMO conceptualizes the service descriptions in terms of a high-level description (class `capability`) and detailed description (class `interface`). The `capability` of the web service or goal describes it as an atomic functional unit in terms of its preconditions, postconditions, assumptions and effects specified as logical expressions in WSML rule language, which is based on F-Logic [80]. The ontological semantics can be defined on the basis on an underlying WSMO `ontology`. Additionally, WSMO particularly focuses different types of heterogeneity that can exist between service descriptions and aim to support their matching through the use of different `mediators` for service descriptions. Unlike OWL-S, WSML/WSMO does not explicitly differentiate between the structural and behavioral parts of the detailed description and merge them in service protocols specified as `interface`. WSML allows the specification of service protocols as Abstract State Machines (ASMs) and the transition rules that invoke state changes in these ASMs comprise of input/output parameters, preconditions and effects. This information contained in the transition rules can be compared to the structural and behavioral description of operations but according to WSMO grounding to WSDL [89], these transition rules and WSDL operations are conceptualized on different granularity level and hence may not have a direct one-to-one correspondence.

Apart from these initiatives that aim to holistically cover the topic of service descriptions, there are others that particularly focus a certain aspect of the requested/offered service. For instance, Visual Contract (VC)

[96] is a UML-based language to specify the behavioral semantics for a software component in general and a service in particular. VC-based behavioral semantics of the requested/offered services are specified through a pair of UML object diagrams depicting the preconditions and postconditions of a service operation [49, 65]. Similarly, Business Process Execution Language for Web Services (WS-BPEL) (or BPEL in short) [77] is the industry standard defined by OASIS for the description of processes for web services. BPEL is an XML-based language, which can be used to specify individual requested/offered service protocols as well as service compositions as processes. It defines a set of activities (e.g., `invoke`, `reply`, etc.), whose invocation can be ordered using the control elements (e.g., `sequence`, `flow`, etc.). BPEL provides a detailed grounding with WSDL, where the activities in a BPEL process can be directly linked to the operations in the corresponding WSDL document. Web Service Conversation Language (WSCL) [69] is another such W3C proposal for the description of service protocols as well as compositions. These focused languages have to be used in combination with other service description languages to reap their benefits, e.g., METEOR-S [97] uses a combination of SAWSDL and BPEL to specify different aspects of the service descriptions.

Next we evaluate these existing languages according to the criteria discussed earlier.

2.3.3 Evaluation

In this section, we evaluate the existing service description languages according to our classification for the languages described earlier in Sec. 2.3.1. In this evaluation, our emphasis is on the languages that are defined in the SOC domain and are particularly focused on the description of services.

In case of the component models discussed in Sec. 2.3.2, we evaluate only those component description languages that allow more than just structural descriptions. This is because the history of structural component descriptions can be traced back to the advent of CBSE with a wide spectrum of proposed approaches and we argue that their lower degree of comprehensiveness make them less interesting for our evaluation results.

Fig. 2.2 shows the results of our evaluation. In the context of the comprehensiveness of the description languages, none of the languages allow a holistic specification of the functional aspects of the services. In this direction, OWL-S [121] and WSML/WSMO [39, 51] are relatively effective but have certain shortcomings. Although, OWL-S allows to define the behavioral semantics as logical expressions, it does not specify any particular

Criteria / Language	Comprehensiveness				Ease-of-use		Request / Offer Distinction
	Operation Signatures	Ontological Semantics	Behavioral Semantics	Service Protocol	Conformance to Standards	Visual Notations	
CompoNETS [10]	+	-	-	+	o	+	+
Fractal [45]	+	-	-	+	-	-	+
Sofa 2.0 [28]	+	-	-	+	-	-	+
Palladio [12]	+	-	-	+	o	+	+
Pin [70]	+	-	-	+	o	+	+
WSDL [162]	+	-	-	-	+	-	-
SAWSDL [136]	+	+	-	-	+	-	-
YASA [30]	+	+	+	-	+	-	-
USDL [8]	+	+	+	-	+	-	-
OWL-S [121]	+	+	o	+	+	-	-
WSML/WSMO [39,51]	o	+	+	+	-	-	+
VC [96]	-	-	+	-	+	+	-
WS-BPEL [77]	-	-	-	+	+	-	-

+ supported - not supported o partially supported

Figure 2.2: Evaluation of existing Service Description Languages

language to specify these logical expressions and suggests to select any existing rule language. Such combination of OWL-S with other languages can introduce further complexities in the process of service description. On the other hand, WSML/WSMO does not have any explicit specification for the requested/offered operations of a service as it describes a service functionality through its protocol as ASM with transition rules to invoke state change. These transition rules do not directly correspond to operations and are conceptualized on different granularity level. Hence, a WSML service description does not correspond to the conventional structure of a functional service description described in Sec. 2.3.1

As far as the ease-of-use for these description languages is concerned, instead of conforming to standard OMG IDL like CompoNETS [10], most of the component models have come up with their specific IDLs. Additionally, apart from Palladio [12] and Pin [70], none enables the use of visual notations for component descriptions. In the SOC domain, approaches like

SAWSDL [136] and YASA [30] conform to the existing service description standard WSDL. Similarly, OWL-S [121], USDL [8] and VC [96] also rely on other de facto standards like XML, OWL, and UML. In this context, WSML [39] totally diverts from the existing standards and comes up with a group of newly-defined languages for the specification of different aspects. A significant number of service description languages, such as, SAWSDL, OWL-S, etc. have textual notations leading to long and complex service descriptions. So despite their conformance to standards like WSDL and XML, studies [135] claim that such complex textual notations make their wide acceptance difficult. This also holds true in the given scenario of service discovery, where particularly the service requester is inclined towards visual notations due to his modeling expertise and hence it is difficult for him to adopt to such textual service description languages for the specification of his service request.

In the context of distinction between service request and offer, all the component models conceptually differentiate between the *required* and *provided interface* of the components. However, in the SOC domain, none of the approaches except WSML/WSMO differentiate between these two types of service descriptions. WSML/WSMO make this conceptual differentiation through the terms *goal* and *web service* for service request and offer. However, these two types of service descriptions in WSML are considered to be exactly similar and are specified using exactly similar notations. This means that these approaches only make this distinction on a conceptual level and do no investigate and cater to possibly different needs or technical expertise of the service partners.

For our service discovery and composition approach, we propose a service description language to overcome these issues discussed here.

2.4 Service Description Matching

In addition to the component specification, seamless adaptation and integration of the components based on their specification matching has also been extensively investigated in the area of CBSE [27, 101, 11]. This has been carried forward to the SOC domain as well, where the process of automatic service discovery and composition majorly relies on an underlying service description matching mechanism. Such a matching mechanism matches the service request at hand to the service offers available on the service market. On the basis of the matching results, a possible service composition may be defined that satisfies the requester's requirements specified in the service

request marking the success of the service composition process. However to achieve this success, it is important that the matching mechanism accurately matches the service descriptions at hand. For this purpose, the matching mechanism has to take certain parameters into account, which are discussed in the next section in detail. Later, we also give an overview of the service description matching mechanisms proposed by the existing approaches. Additionally, we also evaluate these mechanisms on the basis of the defined parameters.

2.4.1 Dimensions of Service Description Matching

The distributed nature of the SOC domain adds multi-faceted complexity to the problem of service description matching. In this context, Becker et al. [11] has earlier proposed a detailed classification of different types of component description mismatches and has come up with an adaptation model that can serve as a basis for a component description matching approach.

Based on this initial understanding, we investigate the service description matching problem and conclude that there are certain dimensions, in which a matching mechanism can function. Each of these dimensions comprise of certain aspects that are arbitrarily addressed by a matching mechanism and it can be evaluated on the basis of its addressed aspects. These dimensions visualized in Fig. 2.3 are discussed in next section in detail, which are later used to classify and evaluate the existing service description matching mechanisms.

Matched Elements

Firstly, a service description matching mechanism has to be classified on the basis of the elements of the service descriptions that it considers while matching.

As discussed in Sec. 2.3.1, service partners can specify their service requests and offers in varying levels of detail according to their preferences and technical capabilities. This decision on their part makes an impact on the outcome of the matching mechanism. For a matching mechanism, it is important to have sufficient information about the requested and offered functionality in order to correctly match them. Consequently, the accuracy of a matching mechanism can be evaluated on the basis of the elements that it matches in the service descriptions.

As an example, consider the case where a matching mechanism enables service discovery and composition on the basis of structural service descrip-

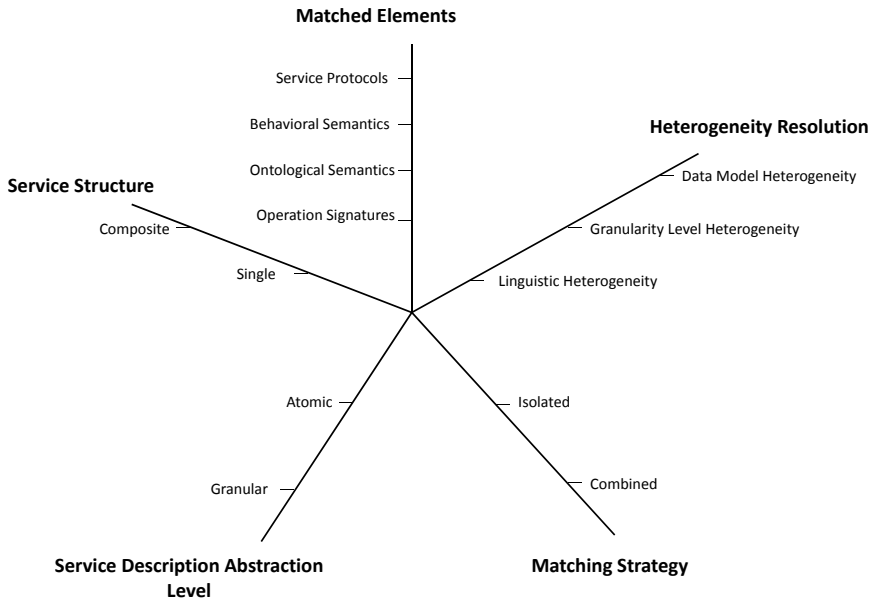

Figure 2.3: Different Dimensions of a Service Description Matching Approach and their Key Aspects

tion matching only. But the requested and offered operation signatures can have different ontological semantics with similar structure and vice versa as discussed in Chap. 1. Hence, in this case, the considered matching mechanism can lead to inaccurate results and can be improved by also considering the ontological semantics of these operation signatures. Similarly, the accuracy of the matching results can be further enhanced by considering different types of behavioral descriptions in the matched service request and offer.

Additionally, the language selected for the specification of each element is also important as it directly effects the wide acceptance and usage of the matching mechanism. An evaluation of the prevailing service description languages presented in Sec. 2.3.3 can serve as a guideline for the selection of a suitable language in this case.

Heterogeneity Resolution

The second important dimension that can be used to classify a matching mechanism is its heterogeneity resolution mechanism. In Chap. 1, we already mentioned that the essence of SOC is that it allows the service partners to function in their independent domains and such independence leads to multi-

faceted heterogeneity among them. A matching mechanism, which is able to resolve this multi-faceted heterogeneity while matching can cater to many diverse domains and service partners in a bigger spectrum and hence can be more effective in practical scenarios.

The first type of heterogeneity that can arise in this case is the *data model heterogeneity*. The service partners tend to have their independent domain knowledge, which leads to their independent local data models. These independent data models can be semantically similar and structurally different or vice versa. As a result, the requested and offered service descriptions that conform to these heterogeneous data models will also be heterogeneous with structural and semantic similarities and differences. An important and recent development in SOC is the explicit specification of the semantics called ontological semantics [121, 51, 136]. In this context, the common approach is to enable the annotation of the elements of the service descriptions with the concepts in an underlying ontology resulting in an explicit and formal semantic specification. This annotation may be done by the service partners manually or as a more sophisticated solution, their can be an automatic annotation mechanism for this purpose. Later, reasoners are developed that realize the matching of service descriptions based on these formal semantics. Another direction is added to the problem of data model heterogeneity if the service partners choose to formally specify their domain knowledge in terms of their independent local ontologies. With the widespread development and use of domain ontologies on a commercial level, it is highly likely that the service partners conform to different local ontologies and consequently their service descriptions conform to their respective local ontology. In such a case, a potential solution has to consider these independent ontological semantics, which need to be consolidated in order to resolve the data model heterogeneity.

Similarly, this independence of the service partners' domains lead to their *granularity level heterogeneity* as well. Based on their conceptualization of the requested and offered functionality, the service partners can specify their service descriptions on different granularity levels. For instance, the requested functionality specified as a single operation in the service request may correspond to the offered functionality specified through multiple operations in the service offer or vice versa. Due to this difference of granularity level, it is not always possible to have a 1 : 1 matching between the operations of the service request and offer hence leading to complex, 1 : n, n : 1, and n : m correspondences between the operations in a request and an offer.

Another important aspect in this regard is the possible *linguistic het-*

erogeneity of the service partners. According to the scenario explained in Sec. 2.2, service partners can select different service description languages based on their individual technical skills and preferences. In order to enable their matching, it is necessary to translate these linguistically heterogeneous service description to a common representation.

In order to ensure its wider acceptance and accurate matching, a service description matching mechanism has to devise techniques to resolve these different types of heterogeneity between the service partners and their service descriptions.

Matching Strategy

Another salient dimension that provides a basis to differentiate the service description matching approaches is their strategy to match different elements of the service descriptions. Comprehensive service descriptions enable accurate matching results because of the fact that the service matching approaches would have elaborate information to match about the requested and offered service. In this context, there can be different strategies to achieve an overall matching result on the basis of the matching of comprehensive service description.

Firstly, an *isolated* strategy can be used where corresponding aspects of the service request and the offer are matched independently and a match degree between them is determined on the basis of the aggregation of these independent match results of different aspects. In this case, the contents or the matching process of one aspect do not affect the matching of another aspect.

However, we argue that it does not make much sense to match different aspects of such comprehensive service descriptions in isolation because these are not independent. Rather, they are interconnected and have dependencies among themselves. Such inter-dependencies and mutual information among different aspects can be exploited to define a *combined* strategy for matching and to further enhance the accuracy of the matching results. For instance, as the operations' input/output parameters can also be a part of their preconditions and postconditions, the operation signature matching results can be reused for the matching of operations' behavioral semantics. In this direction, the consideration of ontological semantic matching result can further enhance the accuracy of results. Similarly, as the service protocols define the required and the allowed invocation sequences for the requested and the offered operation respectively, the operation matching results can be reused to match the service protocols. On the other hand, the order-

ing constraints in the service protocols can act as pivotal information for complex n : m matching between requested and the offered operations.

Service Description Abstraction Level

Another important dimension in which the service description matching approaches can be differentiated is their conceptualization of the abstraction level for the service descriptions, which directly effects the accuracy of matching results. Conceptually, a considerably large number of service discovery approaches deal with service descriptions on a higher abstraction level, where the service is considered as a blackbox entity with an *atomic* service request/offer. Such atomic request/offer comprises a single *operation* in terms of its inputs and outputs. The service description matching mechanism in this case is reduced to the matching of such atomic descriptions on the basis of their inputs and outputs, which is additionally enhanced by a matching of pre- and postconditions in recent years. Service description languages, such as, OWL-S and WSML, etc. have defined notations to specify such atomic service descriptions. Service matching approaches for such atomic service description do not consider further elements of service description, such as, the multiple requested/offered operations and the service protocols during service discovery as these are used later to invoke the discovered services.

On the other hand, there are matching approaches that consider such a blackbox description of the service to be conceptually insufficient in current times. They argue that services are at the core of computing activities and are designed to perform complex tasks. In such cases, it is not suitable to conceptualize these services as blackboxes and *granular* service descriptions comprising of details like multiple operations and the service protocol are required to describe the requested/offered functionality. In this direction, another trend is to *partially* support granular service descriptions, e.g., considering a granular service request, which is used to discover and compose the atomic service offers.

In this context, we argue that in this SOC era, it is important to conceptualize services as complex entities with multiple interconnected functions. Hence, granular service descriptions are more suitable for practical scenarios as they can ensure accurate service discovery results.

Service Structure

Another important dimension of a service matching approach is the structure of the resulting service, i.e., *single* or *composite*.

In the context of SOC, which is based on the independent domains and different granularity levels of the service partners, it is not practical to assume that a single offer fulfills all the requirements in the service request hence leading to a 1 : 1 match between them. It is important that a service description matching approach should not restrict itself to a precise match between the service request and *single service* offer. Rather, in case of a partial matching between the request and an offer, it should also enable a composition of multiple service offer to completely satisfy the service request. This leads to a 1 : n match and the resulting service is structured as a *composite service* based on the discovered service offers.

2.4.2 Overview of the Service Description Matching Approaches

Existing related work in the area of service description matching exist on a broad range and comes under the topics, such as, component/service matching, component/service interoperability, component/service discovery, and component/service composition, etc. Considering this plethora of existing work over the span of last two decades, we have restricted ourselves to the study of a selection of the existing approaches that are relatively more relevant to the problem at hand. Firstly, we do not consider the matching approaches dealing with unstructured service descriptions i.e., service description that do not comply to any existing service description language and comprise of unstructured information in form of plain text, keywords, etc. Secondly, in coordination to the problem at hand, we examine the matching approaches that focus on the functional aspects and do not consider the approaches that particularly focus the matching of non-functional aspects. Thirdly, in order to focus on the current state-of-the-art, we focus on the comparatively recent approaches presented after 2008 and only discuss few of the earlier prominent approaches relevant to our problem. In the following, we give a brief overview of these works.

First of these is presented by Naeem et al. [108], which is a service composition approach for atomic service descriptions based on a visual contracts (VC) matching approach. It is an extension of the 1 : 1 VC matching presented in [65]. It allows a gradual matching between the request and multiple offers, each specified as a single VC leading to a 1 : n VC match.

Graph theory is used as the underlying formalism for matching.

METEOR-S [97] is a framework to support the semantic annotation of services developed at LSDIS laboratory at University of Georgia. It comprises of METEOR-S web services discovery infrastructure (MWSDI) and METEOR-S web services composition framework (MWSCF) for automatic service discovery and composition. MWSDI enables the specification and matching of SAWSDL-based service descriptions, where the definition and matching of ontological semantics is supported through a semi-automatic mechanism. For service composition, MWSCF allows to define a BPEL-based requester protocol and each activity is bound to a service through the service matching carried out by MWSDI.

Web services modeling framework (WSMX) [61] is a service execution environment based on WSMO [51] developed at DERI institute Innsbruck. Relying on the mediator concept introduced in WSMO, WSMX focuses the heterogeneity resolution while matching of the service *goals* and *capabilities*. In this context, the service descriptions and underlying ontologies are based on WSML and an ontology to ontology mediator (OO-mediator) is defined to resolve the semantic mismatches between the local ontologies. For this purpose, a mapping language is introduced to *semi-automatically* define the mappings between the service partners' independent ontologies at design time. OO-mediator uses these mappings to reason and match the data elements during service description matching through web service to goal mediator (WG-mediator). Like METEOR-S, WSMX also considers the service offers (i.e., WSMO capabilities) as an atomic functionality unit during service discovery phase and hence a requester protocol is used to define service composition.

Corfini et al. [25] proposed a service composition mechanism for OWL-S based service requests and offers where the particular focus is to allow matching while considering different local ontologies for the available service offers. In this direction, the mechanism automatically defines mappings between different local ontologies of the available service offers at design time. For this purpose, the notion of *semantic fields* [109] is used and is implemented through the `SemFiT` tool comprising of different match strategies using the information contained in the respective ontologies. These mappings are used to match heterogeneous data elements at the service discovery time.

Bellur & Vadodaria [14] propose a service description matching algorithm for atomic OWL-S based service descriptions comprising their respective inputs, outputs, preconditions and effects. The matching is carried out in two stages where in the first stage the input and output parameters in the request and a certain offer are matched on the basis of the ontological

semantics defined through a common ontology using concepts like semantic distances and logic-based subsumption reasoning [123]. In the second stage, the preconditions and effects in the service request and offer specified as expressions are matched on the basis of the results of the first stage. The matching results between service request and offer are ranked according to their degree of match, i.e., the extent to which the requirements specified in the request are satisfied by the offer. The proposed matching approach supports a $1:1$ match between service request and offer, which is extended to a service composition approach in their later work [13].

Similarly, Bener et al. [15] also proposes a matching algorithm for atomic OWL-S service descriptions where preconditions and effects are specified as SWRL rules. Similar to [14], the matching results between the inputs and outputs based on their ontological semantics are further used to match the preconditions and effects in the service request and offer. This approach also considers the case where the service partners may conform to different local ontologies and hence comes up with a mechanism using WordNet [130] to match the elements in such a case. Matching results of the algorithm are in the form of a ranked list of service offers satisfying some/all requirement specified in the service request.

Kona et al. [88] propose a service composition mechanism for atomic USDL-based service descriptions. To match the data elements, it is assumed that the input/output parameters, preconditions and postconditions are matched on the basis of ontological semantics defined through a common ontology. The approach comes up with an elaborate graph-based mechanism to determine possible compositions of different types, such as, sequential, non-sequential, and non-sequential conditional composition represented as graphs. This graph is further translated to an OWL-S service description of a composite service.

Brogi et al. [26] particularly focuses the fact that considering a black box representation of service offers does not suffice for precise service discovery and the individual operations and their allowed invocations sequences for the offered services also have to be taken into account. For this purpose, service aggregation matchmaking (SAM) algorithm is proposed that matches and composes the offered service protocols specified as OWL-S process models to satisfy an atomic structural service request. With the assumption of a common ontology shared among the service partners, this algorithm constructs a dependency graph from the OWL-S processes to determine any possible service composition. In case a determined composition does not *completely* fulfill the requester's requirements, the algorithm suggests possible improvements to the service request in order to consume the determined

service composition.

Spanoudakis & Zisman [145] propose a service discovery mechanism to build a service-based system based on its UML design. In this approach, the UML-based structural and behavioral elements of the system design form the service requests and the offered services are described in terms of their structural and behavioral features specified through WSDL and BPEL. This approach emphasizes the use of different languages by service partners according to their technical profiles and comes up with a linguistic heterogeneity resolution mechanism that translates the service description elements to data graphs and state machines before their matching. The proposed mechanism however does not support automatic service composition and suggests that this is done by the requester manually on the basis of the discovered services.

Chabeb et al. [31] proposed a matching mechanism for YASA-based service description. The matching approach matches the individual elements, such as, interfaces, operations, inputs, outputs, preconditions and postconditions on the basis of their ontological semantics defined through a domain ontology. Based on the aggregation of these element-level matching results, a comprehensive mechanism is used to determine an overall matching degree between a service request and an offer.

Grigori et al. [60, 35, 59] particularly focuses the behavioral matching of the request and offer, where the service protocols are specified as BPEL processes. For this purpose, the requested BPEL process is translated to its respective graph and later on matched to the graphs of the offered BPEL processes. It uses different types of matching, such as, linguistic matching, granularity-level matching, etc. between graph elements to determine a graph similarity value. The mechanism determines 1 : n and n : 1 correspondences between the requested and offered BPEL activities on the basis of their input/output parameters. The service discovery process returns a list of the service offers ranked on their similarity values.

Cubo etc al. [37] semantically matches WSDL-based operations in the requested and offered service descriptions by assuming that ontological semantics are defined through XML-based technologies. The resulting 1 : 1 operation correspondences are used to match the service protocols as specialized labeled transition systems (LTSs), which are checked for their compatibility through detection of any deadlock resulting from a possible mismatch in their synchronous product.

Cuzzocrea and Fisichella [38] presents a service matching approach where the service request and offer are specified as OWL-S processes. The approach defines a graph representation for OWL-S processes to automatically trans-

late the requested and offered processes to their respective graphs. Later, a graph matching approach is applied which finds common substructures based on node similarities. In this direction, a 1 : 1 node similarity is calculated for atomic processes (operations) and these matching results are used to calculate the maximum common subgraph between the two graphs.

Fernández et al. [57] proposes a service matching approach that particularly deals with the data model and linguistic heterogeneity issues for the service partners. The approach proposed a common representation of a service description known as *general common model (GCM)* after a conceptual comparison of the existing languages. To match the possibly linguistically heterogeneous atomic service descriptions of the service partners, they are translated to this common representation through the mappings defined between GCM and their respective specification languages. A service description matching mechanism matches the GCM-based service request and offer on the basis of its inputs, outputs, preconditions and postconditions considering their ontological semantics. In case the service partners do not share a common underlying ontology, their ontological semantics are consolidated on the basis of already-defined ontology alignments. For this purpose, a semi-automatic mechanism for defining the ontology alignments is defined.

Klusch et al. [81] proposes a matching mechanism for atomic structural OWL-S service description. This is a hybrid matching mechanism combining semantic matching and text-based similarity calculation. If the matcher is not able to determine a satisfactory result on the basis of semantic matching, it tries to determine an approximate match result through different text-based similarity calculation techniques. Such an approximate match may be selected by the requester as it *partially* satisfies its requirements. Different variants of this matcher are implemented with different combinations of the semantic and textual matching techniques. Later, a self-adaptive mechanism [82], which uses a learning mechanism to discover the most optimal combination of these matching techniques to aggregate matching results and hence enable successful service discovery. This approach of self-adaptive hybrid matching is further enhanced to also consider the matching of preconditions and effects in iSeM [83] while reusing the signature matching results. Some other variants [85, 86] reuse the hybrid matching mechanism for other service description languages.

Similarly, Ke & Huang [79] define a self-adaptive service discovery approach for OWL-S atomic service descriptions without behavioral semantics. It is based on the matching of their respective *ontology trees* constructed from their ontological semantics and relies on the computation of different metrics, such as, conception similarity, structure similarity, and attribute

similarity, etc. In case the matching results are not satisfactory, the request is automatically adapted on the basis of the similarity results to achieve a satisfactory matching result.

Liu et al. [95] argue that it is difficult for service partners to adapt to new languages for service description as well as to define ontologies for ontological semantics. Hence, they propose to exploit information retrieval techniques and use a web search engine results for calculating the semantic distances between the terms used in the WSDL-based service descriptions. In this approach, the matched service descriptions are presented as bipartite graphs with the terms as nodes and their semantic distance as the weight of the connection between the corresponding nodes. The overall similarity of the two service descriptions is calculated through a similarity metric defined on the basis of matched and unmatched terms and their similarity distances in the bipartite graphs.

Masuch et al. [100] proposes a hybrid matcher for atomic OWL-S descriptions, which applies textual and semantic matching techniques and aggregates the matching results to determine the overall similarity value between the service descriptions. The preconditions and effects are specified as SWRL rules, which are matched on the basis of their structure and ontological semantics.

Motahari-Nezhad et al. [107] argue that for an accurate service description matching, it is important that different aspects are not considered in isolation. Rather, the service description matching mechanism should be able to exploit and combine the information in different aspects to achieve accurate matching results. In this context, they propose an approach to match the WSDL descriptions while exploiting the ordering constraints and other information in the respective BPEL service protocols. As a result, 1 : n and n : 1 correspondences are determined between the operations and their parameters.

Plebani & Pernici [127] propose a matcher for WSDL and SAWSDL service descriptions where the operation matching is based on their signatures and ontological semantics defined through a common underlying domain ontology for the service partners. In case of WSDL descriptions, these ontological semantics are defined by the matcher automatically during matching whereas the service partners define them beforehand in case of SAWSDL descriptions. A similar approach is given by Tran et al. [153] for SAWSDL service descriptions. However, the matching results in this approach are comparatively detailed comprising of a similarity degree, ratio of the matching input/output parameters, the mappings between the matched parameters, etc.

2.4.3 Evaluation

In this section, we evaluate the service description matching approaches discussed in Sec. 2.4.2 according to the dimension of this problem defined in Sec. 2.4.1.

First of all, it is important to note that most of the approaches in this direction in the recent times realize the importance of comprehensive service descriptions for the accuracy of automatic service discovery and composition hence enabling the matching of different aspects of such comprehensive service descriptions.

In the context of multi-faceted heterogeneity inherent due to the nature of SOC paradigm, the existing matching approaches have not convincingly dealt with this aspect so far.

The first in this direction is the resolution of the data model heterogeneity. Unfortunately, a considerable number of approaches do not provide any solution for this important issue. Some [108, 145, 38] consider service descriptions based on the same data model without conforming to a formal ontology to define ontological semantics. Others [97, 14, 88, 31, 37, 81, 79, 100, 95, 127, 153] conform to formal ontology to define the ontological semantics for these data elements. However, these simplify the situation at hand by assuming that the service partners share a common underlying ontology and they define their service descriptions and their ontological semantics while conforming to this common ontology.

In this direction, with the independence of service partners in SOC and the diversity of their technical skills, it is not practical to assume that the service partners will conform to common domain knowledge in advance. Hence, they use their independent ontologies for the definition of their service descriptions and their respective ontological semantics.

Approaches like [15, 25, 61, 57] provide the service partners the flexibility to conform to their independent local ontologies and the heterogeneity is resolved while consolidating these different ontological semantics. For this purpose, [25, 61, 57] propose the definition and use of ontology alignments among independent local ontologies. However, such a solution may perform well in an academic setting but its scalability is questionable in a practical scenario where alignment between each pair of local ontologies can be very costly due to considerably large number of requests and offers. In this context, [15] does not rely on ontology alignments rather it uses the general-purpose ontology WordNet [130] to define and match the ontological semantics in case of independent local ontologies for the matched service partners.

Criteria / System	Matched Elements				Heterogeneity Resolution			Match Strategy	Serv. Desc. Abstr. Level	Service Structure
	Operation Signatures	Ontological Semantics	Behavioral Semantics	Service Protocols	Data Model Heteogeneity	Granularity Level Heterogeneity	Linguistic Heterogeneity	Isolated (i) / Combined (c)	atomic (a) / granular(g)	single(s) / composite (c)
Naeem et al. [108]	-	-	VC	-	-	o	-	i	a	c
METEOR-S [97]	WSDL	SAWSDL	-	BPEL (req.)	-	-	-	c(o)	g(o)	c
WSMX [61]	WSML	WSML	WSML	WSML (req.)	+	-	-	c(o)	g(o)	c
Corfini et al. [25]	OWL-S	OWL-S	-	OWL-S (prov.)	+	-	-	c(o)	g(o)	c
Bellur & Vadodaria [14]	OWL-S	OWL-S	Expr. lang.	-	-	o	-	c(o)	a	c
Bener et al. [15]	OWL-S	OWL-S	SWRL	-	+	-	-	c(o)	a	s
Kona et al. [88]	USDL	USDL	USDL	-	-	o	-	i	a	c
Brogi et al. [26]	OWL-S	OWL-S	-	OWL-S (prov.)	-	-	-	c(o)	g(o)	c
Spanoudakis & Zisman [145]	UML + WSDL	-	-	UML + BPEL	-	-	+	c(o)	g	s
Chabeb et al. [31]	WSDL	YASA	-	-	-	-	-	i	g	s
Grigori et al. [60,35,59]	BPEL / WSCL	-	-	BPEL / WSCL	-	+	-	c(o)	g	s
Cubo et al. [37]	WSDL	XML	-	LTS	-	-	-	c(o)	g	s
Cuzzocrea & Fisichella [38]	OWL-S	-	-	OWL-S	-	-	-	c(o)	g	c
Fernandez et al. [57]	Diff.	Diff	Diff	-	+	-	+	i	a	s
Klusch et al. [81,83]	OWL-S	OWL-S	OWL-S	-	-	-	-	c(o)	a	s
Ke & Huang [79]	OWL-S	OWL-S	-	-	-	o	-	c(o)	a	c
Liu et al. [95]	WSDL	Search Engine	-	-	-	-	-	c(o)	g	s
Masuch et al. [100]	OWL-S	OWL-S	SWRL	-	-	-	-	i	a	s
Motahari-Nezhad et al. [107]	WSDL	-	-	BPEL	-	+	-	c(o)	g	s
Plebani & Pernici [127]	WSDL	SAWSDL	-	-	-	-	-	c(o)	g	s
Tran et al. [153]	WSDL	SAWSDL	-	-	-	-	-	c(o)	g	s

Diff.: WSDL / OWL-S / WSMO / SAWSDL + : supported o : partially supported - : not supported

Figure 2.4: Evaluation of some prominent Service Discovery and Composition Approaches

As far as granularity level heterogeneity is concerned, very few approaches consider and come up with mechanism to resolve this type of heterogeneity. Approaches like [108, 13, 88, 79] allow service composition by enabling a match between an *atomic* service request and multiple atomic service offers. We believe that in these approaches, the matching of an atomic service request to atomic service offers is analogous to a $1 : n$ opera-

tion correspondence between requested and offered operations. Some others [107, 59] particularly realize the need to consider the difference of granularity level between a service request and an offer and come up with mechanism to determine 1 : n and n : 1 operation correspondences. However, these approaches restrict the granularity level heterogeneity to the operation signatures only and resolve it through the matching and splitting/merging of input and output parameters. The complexer case of different granularity of behavioral semantics despite similar operation signatures and the resulting complex operation correspondences is ignored in these approaches.

The domain of service description matching also lack sufficient work in the area of linguistic heterogeneity resolution. Very few approaches [57, 145] realize the fact that the service partners may opt for different service description languages according to the particular scenario at hand, their specific requirements and technical skills. Hence, these approaches allow heterogeneous languages for the specification of service request and offers, which are translated to a common representation before their matching is carried out.

As far as the matching strategy of these approaches is concerned, approaches like [100, 31, 88, 57] adopt an *isolated* strategy where they match different elements of the service descriptions in isolation and then aggregate their matching result to determine an overall degree of matching between the request and offer. Similarly, there are other approaches like [14, 15, 83, 79, 37, 38, 145, 107, 60, 59], that do not match elements in isolation rather adopt a *combined* strategy where the contents or matching results of one aspect are reused while matching another aspect.

However, it is worth mentioning here that most of the existing approaches with combined match strategy are limited in nature and do not elaborately exploit the inter-dependencies of the comprehensive service descriptions. These restricted combined approaches are mentioned through *c(o)* in Fig. 2.2. These are limited in the sense that they lay particular focus on the matching of a certain aspect, such as operations or protocols and ignore or simplify the matching of other aspects. For instance, approaches like [14, 15, 83, 79] particularly deal with operation matching where the results of operation signature matching using ontological semantics contribute to the matching of their behavioral semantics. However, service protocols are not considered in these approaches. There are others [37, 38, 145] that further deal with service protocol matching in addition to operation matching. However in these approaches, simple 1 : 1 operation correspondences are considered while matching the protocols. For this purpose, either it is assumed that such 1 : 1 operation correspondences already exist or a rela-

tively simple operation matching mechanism that mostly rely on signature matching is defined to determine such correspondences. In this case, the complex n : m operation correspondences are not considered and hence not used while matching the protocols. A step further in this direction are the approaches [107, 60, 59] which handle the operation and protocol matching in an interconnected manner where complex 1 : n and n : 1 operation correspondences are also taken into account. The ordering constraints specified in the protocols contribute to determine such complex correspondences. However, operation matching is limited to matching of operation signatures and other aspects, i.e., ontological and behavioral semantics do not play any role.

Hence, none of the existing approaches allows a holistic matching mechanism where the operation and protocol matching are combined while considering their contents and matching results elaborately. In this direction, it is required that the operation matching should not be restricted to the operations' signatures but should also take their ontological and behavioral semantics as well as their ordering constraints in the service protocols leading to n : m operation correspondences. These complex operation correspondences should later contribute to the service protocol matching ensuring accurate service discovery results.

It is also worth noting that a considerably large number of the service matching approaches target atomic or partially granular service descriptions (mentioned as g(o) in Fig. 2.2). For example, approaches like [97, 61, 57, 100, 14, 26, 25] allow the matching of service descriptions where one or both service partners describe their required/offered functionality in terms of atomic service description. This leads to the fact that the service description matching in these cases is dealt with an operation matching mechanism only and protocol matching is not taken into account.

In addition, it is also important to mention that fairly large number of service matching approaches only realize matching between a single service request and offer. These approaches ignore the fact that a service requester can have diverse needs that may not be fulfilled by a single service offer and a composite service comprising of multiple service offers might be necessary to completely fulfill the service request.

Based on this evaluation of the existing service description matching approaches, we deduce that there is a requirement for an automatic service description matching approach that enables accurate results by:

1. **comprehensive matching** of the service request and available offers using different functional aspects, such as, operation signatures, their

ontological and behavioral semantics and service protocols. In this direction, the accuracy is improved through a **combined approach** where the contents/matching results of one aspect contribute for the matching of other aspects.

2. providing a **multi-faceted heterogeneity resolution** mechanism, which should resolve (i) the **data model heterogeneity** arising from different domain knowledge, which may possibly be specified as an ontology (i) the **granularity level heterogeneity** arising from the heterogeneous behavioral semantics leading to complex n : m operation correspondences (ii) the **linguistic heterogeneity** resulting from the heterogeneous languages of the service partners selected due to their individual requirements and skills.

3. enabling **service composition** in case a single service offer is not able to completely fulfill the requirements specified in the service request.

2.5 Summary and Discussion

In this Chapter, we provided the background that is necessary for the work presented in this thesis. This comprises an insight of service-oriented computing and its major concern of automatic service discovery and composition. In this direction, we further investigated in detail its two important sub-areas, i.e., specification of service descriptions and their matching. Additionally, we presented a detailed overview as well as an evaluation of the state of the art in both these areas.

On the basis of this evaluation, we derived the basic requirements for an automatic service discovery and composition approach ensuring accurate results. In the remainder of the thesis, we present our service discovery and composition approach that aims to fulfill these requirements and hence lead to accurate results.

3

Rich Service Description Language (RSDL)

In this chapter, we introduce our Rich Service Description Language (RSDL), which enables service partners in the OTF Computing scenario to describe their service requests and offers comprehensively in terms of their functional aspects. In this context, it should be kept in mind that the main focus of this thesis is not to have a complete language specification with its elaborate and well-defined semantics. Rather, the aim is to focus on comprehensive notations for service descriptions that can enable automatic service matching while fulfilling the criteria specified in Sec. 2.3.1.

Before going into the details of the proposed language, we lay out the requirements for a comprehensive service description language on the basis of the evaluation of existing approaches presented in Sec. 2.3.3. To fulfill these requirements, we present RSDL and its elaborated set of notations to specify service requests and offers in the later sections. A summary and discussion will be presented at the end of the chapter.

3.1 Requirements for a Comprehensive Service Description Language

In this section, we outline the requirements that a service description language has to fulfill in order to be adopted by the service partners in the OTF Computing scenario to allow automatic service description matching. These requirements are based on the criteria laid out in Sec. 2.3.1. As discussed in Sec. 2.3.3, the existing languages for service description do not meet this criteria completely.

A potential service description language has to fulfill the following requirements:

- It should introduce an elaborated set of notations allowing the service partners to holistically describe the functional aspects of their

request/offer.

- It should be easily adaptable by the service partners through its conformance to the existing standards and use of visual notations.

- It should provide specialized notations for the service requesters and providers to specify their service descriptions according to their individual needs and motives.

In the next section, we introduce our proposed language *Rich Service Description Language* or RSDL in short.

3.2 Rich Service Description Language

Rich Service Description Language (RSDL) focuses the specification of service descriptions while fulfilling the requirements specified in Sec. 3.1. It provides a detailed set of UML-based visual notations to comprehensively specify the functional aspects of a service description. It ensures general acceptability by conforming to UML, which is already a de facto standard in software engineering domain and provides a detailed set of visual notations to specify different aspects of a system. Additionally, RSDL acknowledges the difference between the motives of service requester and provider while specifying their respective service protocols and hence provides specialized notations according to the particular nature of the protocols.

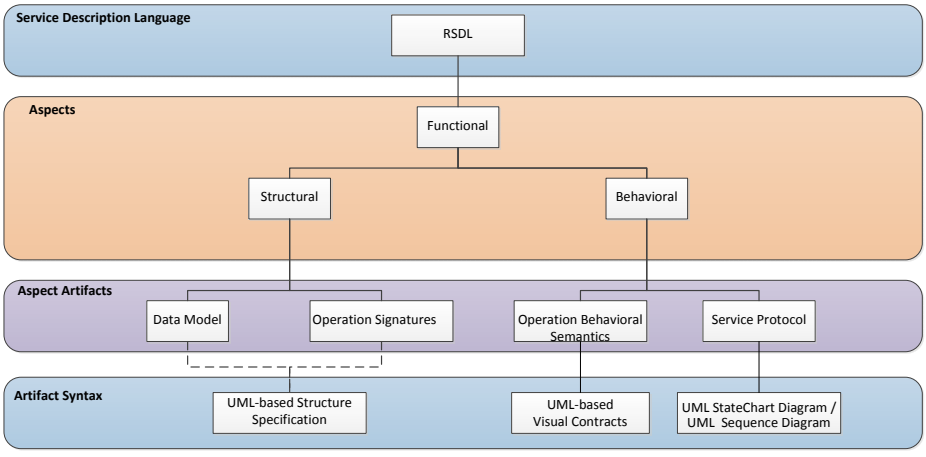

Figure 3.1: The hierarchical Structure of the RSDL

In the following, we first introduce the hierarchical structure of the RSDL encompassing its basic artifacts. Later, we will discuss in detail the syntax of RSDL used to realize its artifacts.

3.2.1 Structure of the RSDL

The elaborate structure of the RSDL is introduced in Fig. 3.1. Fulfilling the first requirement for a comprehensive service description language, RSDL allows a detailed specification of functional *aspects* of a service mainly in terms of its structure and behavior. For this purpose, it specifies different *artifacts* specified through the set of notations comprising the *syntax* of RSDL. These artifacts of RSDL also have well-defined *semantics*, which will be presented in Chap. 4.

To cater to the comprehensiveness notion defined in Sec. 2.3.1, RSDL also supports the definition of ontological semantics for the structural artifacts, which will also be discussed in the next section.

A detailed description of the notations selected for the specification of these artifacts is as follows.

3.2.2 Syntax of RSDL

The syntax for different artifacts of the RSDL is based on UML as the underlying language. For visual languages like UML, the syntax mainly comprises of two parts, i.e., abstract and concrete syntax. Following this tradition, we define a metamodel for RSDL to specify its abstract syntax. Additionally, we specify its concrete syntax in terms of RSDL-based service requests and offers for our running example of a tourism scenario.

Following the existing trend where the frameworks for visual editors development, e.g., Eclipse Modeling Framework[1], mainly use metamodels for this purpose, the RSDL metamodel also serves as a basis for the development of the service discovery and composition workbench in our approach.

Before going into the details of RSDL's syntax, we briefly discuss our selection of UML. One of the requirements for a suitable service description language is that it should be easily adaptable by the service partners through its conformance to the existing standards and based on visual notations.

UML is a standardized modeling language in the SE domain, which has been approved as an industry standard for specifying, designing and documenting software systems since 2000. It allows to build a better understanding of the large-scale software systems, where the complex design of

[1] http://www.eclipse.org/modeling/emf/

the system may not be easily conveyed through textual descriptions. UML-based visual modeling not only allows better communication but the resulting models can also be formally validated. Based on these properties, it is an open de facto standard, which is widely used in academia as well as industry.

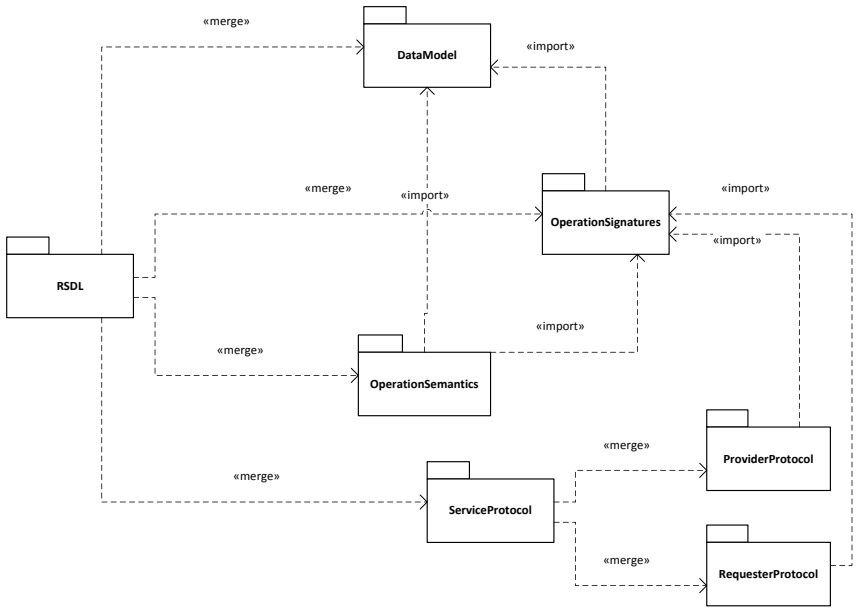

Figure 3.2: Package Structure of the RSDL Metamodel

Considering these features of UML and its widespread use in the industry [42], we select UML as the underlying language for RSDL. To realize the structure illustrated in Fig. 3.1, we specify each of its artifacts as a package containing its metamodel. These independent metamodels are linked together on the basis of the dependencies among them, which results into an integrated coherent metamodel for RSDL. Fig. 3.2 gives an overview of this package structure of RSDL.

The `RSDL` package represents the overall metamodel of the proposed language and merges the metamodels of different artifacts specified as the `DataModel` package, the `OperationSignatures` package, the `OperationSemantics` package, and the `ServiceProtocol` package. The `ServiceProtocol` package constitutes the metamodels for service partners' protocols specified in the `RequesterProtocol` package and the `ProviderProtocol` package. The `OperationSignatures` package imports the `DataModel` package to model types of the operation's input/output parameters. Simi-

larly, the `OperationSemantics` package imports the `DataModel` and the `OperationSignatures` packages to model the semantics for a particular operation. On the other hand, the `RequesterProtocol` and the `Provider-Protocol` packages also import the `OperationSignatures` package to model the operation invocations in the service protocol. Details of these dependencies among packages will be discussed in more detail later.

Data Model

The first structural artifact for an RSDL-based service description is the *data model* of the respective service partner. It comprises of the basic structural details, such as, the classes[2], their attributes and the associations among these classes. In this context, UML class diagram is selected as the suitable choice to specify such a data model. Fig. 3.3 shows some relevant parts of the UML 2.0 metamodel comprising the `DataModel` package in RSDL metamodel. [115] can be referred for complete details of the syntax of UML class diagram.

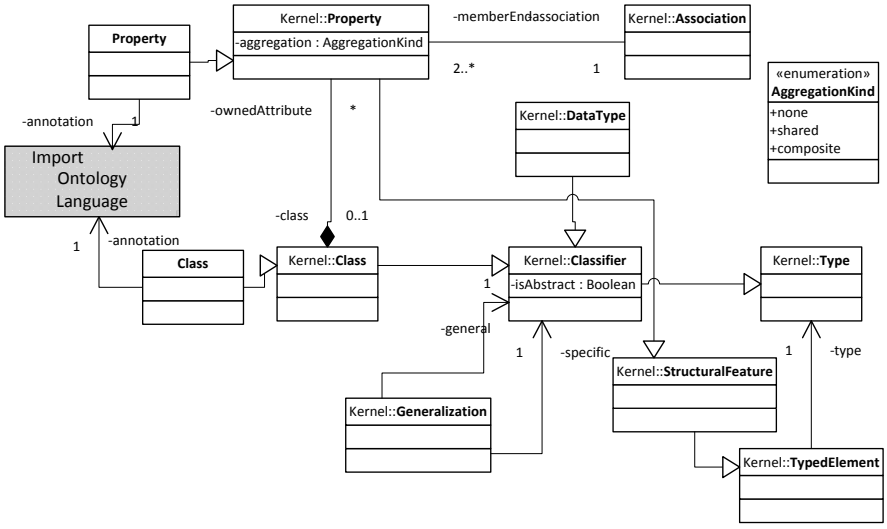

Figure 3.3: `DataModel` Package in the RSDL Metamodel

The `DataModel` package mainly comprises of the elements from the `Kernel` package in UML 2.0 specification constituting the UML class diagram. A visual representation is shown in Fig. 3.3, where the classes in the

[2]Other terms used in literature for the same purpose are data types, concepts, entities, etc.

data model are modeled through `Class`, which is a specialization of UML `Class`. A class is a `Classifier` describing a set of objects having common features, constraints and semantics. In a data model, the attributes of a class are represented through `Property`, i.e., a specialization of UML `Property` depicting a `StructuralFeature` of a `Class`. Each attribute as a `StructuralFeature` is a `TypedElement` and hence has a `Type`. The predefined primitive data types are represented through `DataType` and are used in the data model to type the attributes of the classes. In this direction, due to its widespread acceptance as a standard in SOC, RSDL conforms to XML data types[3] as the primitive data types.

The classes in the data model can be related through 2 types of relationships: `Generalization` and `Association`. A `Generalization` relationship relates a `general` classifier to a `specific` classifier. On the other hand, two classes can be related through an `Association`, which can have three different kinds modeled as enumeration values in `AggregationKind`: `none`, `shared` (representing the aggregation relationship) and `composite` (representing the composition relationship). The kind of an association is modeled as the `aggregation` attribute of the `Property` associated to that `Association`.

As mentioned earlier, RSDL also supports the specification of ontological semantics for the structural elements in the service description. In this direction, the definition of an underlying ontology language is not the focus of our work. We assume that any existing language like OWL [158] or RDFS [159], XML Schema, etc. is used for the specification of ontologies and its language specification can be imported and used as a package. For our work, we allow the annotation of the basic elements such as classes and their attributes in the data model with their semantic counterparts in the underlying ontology. To enable such an annotation, each `Class` and `Property` of the data model is linked to a concept in the ontology. A concept in the ontology represents a basic unit of information of an information domain captured in the respective ontology and is modeled differently in different languages, e.g., element `Class` in OWL[158] is used to model a concept. We argue that such an assumption about the underlying ontology language suffice for the annotation of data model elements in the given scenario and hence we do not go into its further details.

[3] http://www.w3.org/TR/xmlschema-2/

Operation Signatures

The second structural artifact in RSDL is the *operation signatures*. The `OperationSignatures` package is shown in Fig. 3.4 based on the elements of UML 2.0 metamodel meant to define an interface of the system with particular operations.

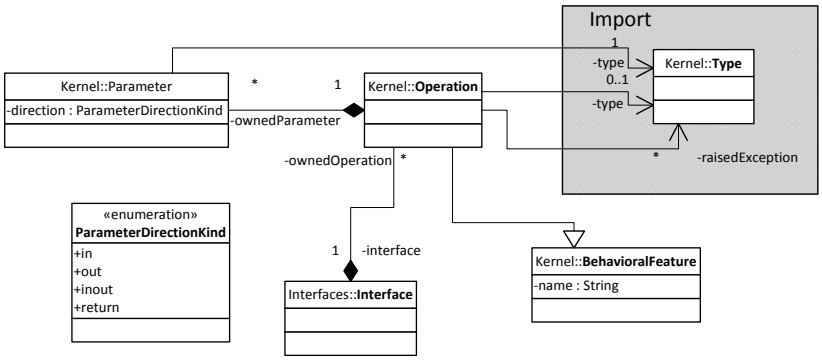

Figure 3.4: `OperationSignatures` Package in the RSDL Metamodel

According to this metamodel, an `Operation` is a `BehavioralFeature` owned by an `Interface`. Each operation has a name and can have input and output parameters modeled through `Parameter`. Each parameter has a `Type`, which is imported from the `DataModel` package and `direction`, whose value can be selected from the enumeration `ParameterDirectionKind`. The direction of a parameter determines whether it is an input or an output parameter. An operation can have at most one `return` parameter and the type of the operation is determined by the type of its return parameter. An operation may also raise exceptions of specific `Type`.

It is important to mention here that the service partners using RSDL for their service descriptions are recommended to type their operation parameters over the classes in the underlying data model instead of using primitive data types. Such a conscious effort facilitates an implicit matching of operation signatures while matching of operations' behavioral semantics hence ensuring better accuracy of matching results during service discovery process. This aspect will be clarified in more detail in Chap. 6, where we discuss our operation matching mechanism.

As WSDL is the current standard [162] for structural service descriptions, we claim that RSDL-based structural specification comprising of the data model and operation signatures directly correspond to the operation-based structural descriptions in WSDL and can be automatically translated to a

respective WSDL specification. Such a translation from UML to WSDL and vice versa is already extensively worked on. For instance, IBM's tool [78] allows a two-way model transformation between WSDL and UML. Such a translation is not the focus of this thesis, therefore we assume that the existing tools and techniques [43, 78] can be reused for this purpose.

Operation Behavioral Semantics

RSDL allows a functional behavior specification of a service in terms of the *behavioral semantics* for its requested/offered operations defined through its preconditions and postconditions, where the preconditions and postconditions specify the state of the system before and after the execution of the said operation, respectively.

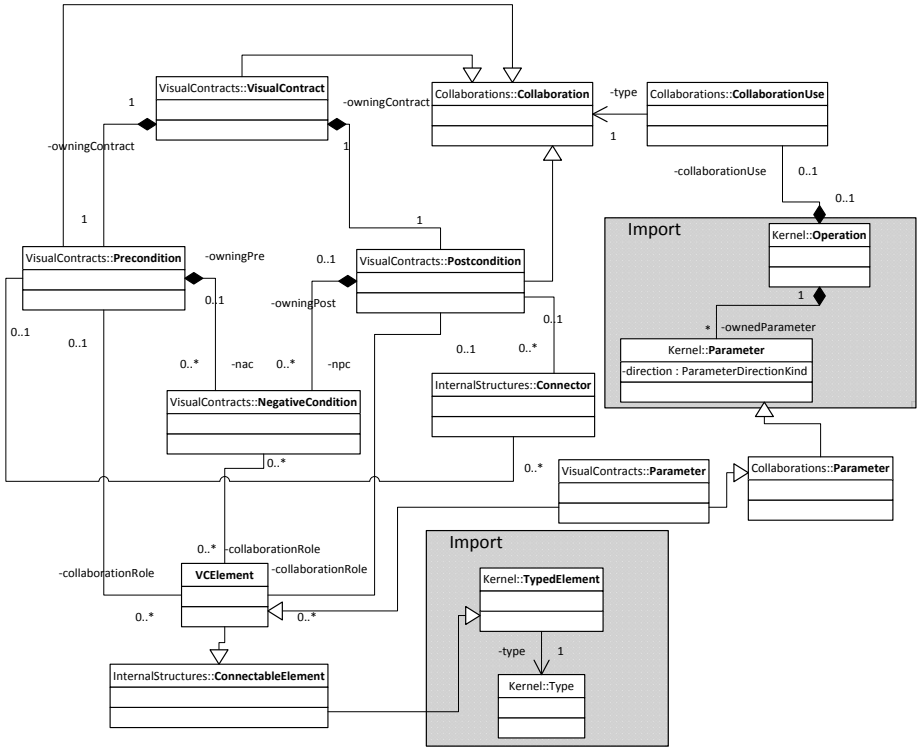

Figure 3.5: `OperationSemantics` Package in the RSDL Metamodel

In this direction, *Visual contracts (VC)* [96] is a UML-based visual modeling language for functional behavior specification of the software compo-

nents. A VC for an operation is specified as two UML object diagrams representing the preconditions and postconditions for its invocation where the objects are typed over the underlying data model specified as UML-class diagram. We select visual contracts as the notation to specify the *operation behavioral semantics* for the requested/offered operations in RSDL service description.

The `OperationSemantics` package in RSDL metamodel is shown in Fig. 3.5, which is based on the metamodel for visual contracts in [96].

A `VisualContract` is a `Collaboration` linked through `Collaboration-Use` to an `Operation` imported from `OperationSignatures` package. In UML 2.0, `Collaboration` provides basic elements to specify a Composite-Structure-Diagram that visualizes a snapshot of a system in a particular context to achieve a certain goal in terms of functionality. For this purpose, Roles are specified that are linked to each other with connectors. At runtime, these roles are taken over by the instances to reach the system state and hence achieve the goal specified in the *Composite-Structure-Diagram*. Based on this conceptualization, a visual contract in RSDL is a collaboration which visualizes the invocation of a particular operation. In this context, the parameters of the operations can also be reused as roles in the corresponding collaboration. This is realized through `Parameter` in `Collaboration`, which is a specialization of UML `Parameter`.

A VC further consists of two collaborations: `PreCondition` and `Postcondition`, each of which consists of multiple `VCElements` that are `ConnectableElements` connected through connectors. A parameter of the concerned operation can also be a part of the preconditions or postconditions of a VC as a `VCElement`. This is modeled through `VisualContracts:Parameter`, which is a specialization of `VCElement` and `Collaborations:Parameter`. As `TypedElements`, `VCElements` are typed over the `Types` imported from the RSDL-based data model.

A visual contract also allows the specification of negative pre- and post-conditions modeled as `Negativecondition`, which is also a collaboration. These negative conditions can be seen as an extension of preconditions and postconditions and specify the structures, which are not allowed to occur before and after the execution of the concerned operation. We refer the reader to [96] for further details of the visual contracts meta model.

Here we argue that in a realistic scenario, it makes sense that all the input parameters of an operation are part of the system state before its execution. Similarly, the output parameter is part of the system state after its execution. Keeping this argument in mind, RSDL applies a further constraint on the specification of a VC for an operation as follows:

Constraint: *each input and output parameter of the concerned operation, which is typed over a class in the underlying data model also occurs in the preconditions and postconditions of its VC, respectively.*

This constraint is important in the sense that it supports the operation signature matching while matching of the requested and offered operations. This aspect will be elaborated further during the discussion of our operation matching mechanism in Chap. 6. Here it suffices to say that our operation matching mechanism aims at matching the requested and offered operations on the basis of their structural as well as behavioral descriptions. Based on the given constraint where the parameters in the operation signatures are included in the respective behavioral semantics, the proposed operation matching mechanism only focuses on the matching of the behavioral semantics, i.e., VCs of the requested and offered operations. Consequently, this includes the matching of their respective operation signatures and no separate mechanism is required for operation signature matching of the requested and offered operations.

Requester Protocol

RSDL allows the specification of the dynamic behavior of service partners in terms of their respective service protocols specifying their required/allowed operation invocation sequences. For a requester's service protocol in the OTF computing scenario, there are certain motives and requirements that need to be met. The requester aims to realize a *single* end-to-end use case by invoking multiple operations of a discovered service in a specified sequence. This invocation is carried out in an active manner by the requester without waiting for any external event. For instance, in our running example, HRS offers a multi-step tour booking use case to the end user. For its realization, the HRS application developer wants to invoke multiple operations on a discovered service *TripPlanner* in a single sequence.

Keeping these considerations in mind, we argue that *UML sequence diagram* is the suitable choice for the *requester service protocol* in RSDL. As one of the interaction diagrams offered by UML, the main purpose of the sequence diagram is to describe inter-process communication for a particular scenario of a system by describing the interaction of its components arranged in time sequence. These interaction sequences are represented through messages exchanged among the participating components represented through their lifelines. Fig. 3.6 shows some of the relevant parts of the UML 2.0 metamodel for interaction diagrams comprising the `RequesterProtocol` package in the RSDL metamodel.

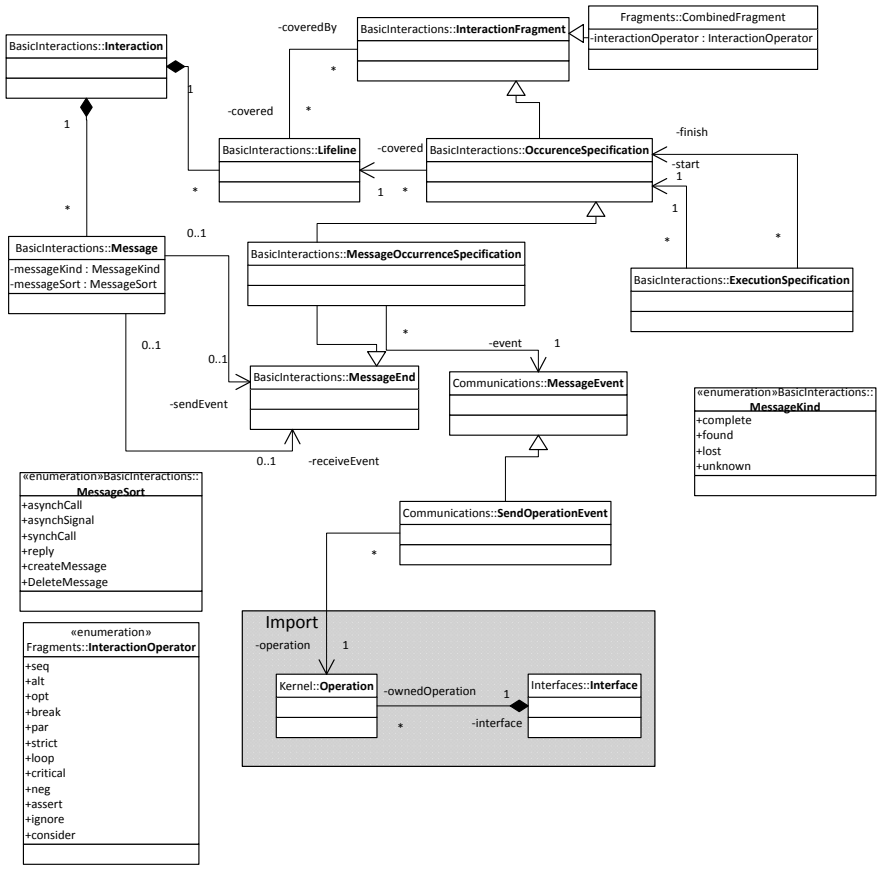

Figure 3.6: Excerpt of `RequesterProtocol` Package in the RSDL Metamodel based on UML sequence diagram

A requester protocol is a sequence diagram modeled as an `Interaction` with participants, i.e, the service requester and the invoked service modeled as `Lifelines`. An Interaction comprises of `InteractionFragments`, which includes complex constructs, e.g., if-else block, loop, and break etc. modeled through `CombinedFragment`. The executions on a lifeline is represented through an `ExecutionSpecification` graphically represented as a thin rectangle on lifelines, whose start and finish is represented through two `OccurrenceSpecifications`. The `OccurrenceSpecification` is the basic semantic unit of an `Interaction` modeling the basic occurrences along a lifeline. The lifelines communicate with each other through `Messages` that has a particular kind (complete, lost, found, and unknown) and a communication

sort (synchronous, asynchronous, etc.) modeled through the enumerations `MessageKind` and `MessageSort`, respectively. Each message has a sending and receiving `MessageEnd`. A message in the sequence diagram can possible represent the invocation of an operation, which is initiated through the `SendOperationEvent` on the sending `MessageEnd`. `SendOperationEvent` is a specialized `MessageEvent`, which specifies the sending of a request to invoke an operation on an object. The referred `Operation` is imported from `OperationSignatures` package. The occurrence of the message events on the message ends is modeled as `MessageOccurrenceSpecification`, which is a special type of `OccurrenceSpecification` and `MessageEnd`.

For complete details about these and other elements of the UML Interactions, we refer the reader to the UML specification [115].

Provider Protocol

For a provider's service protocol in OTF computing scenario also, there are certain requirements that need to be met. A provider aims to specify all the possible ways in which it allows its offered service to be invoked. As a result, the provider's service protocol can have multiple sequences of the offered operations, which can be invoked by a service requester who wants to use the service. For instance, in the running example, a hotel service may offer other scenarios as well apart from room booking, e.g., It can allow to the booking of events, etc. or a payment service can offer to make payment through different payment modes, e.g., credit card or a bank account, etc.

Considering these features, we select UML *protocol statemachine diagram* as the notation for the *provider service protocol* in RSDL. UML statemachine diagram allows the specification of a complete system in terms of various states that it can achieve as a result of the invoked transitions. In this context, UML protocol statemachine is a specialized statemachine diagram that particularly focuses the specification of the usage protocol of the system, i.e., the legal operation invocation sequences that can be invoked on a system by an external entity. Fig. 3.7 shows some relevant parts of the UML 2.0 metamodel for statemachine diagram comprising the `ProviderProtocol` package in RSDL metamodel. [115] can be referred for complete details of the syntax of UML Statemachine diagram.

A provider protocol is modeled as a particular type of `StateMachine`, i.e., `ProtocolStateMachine` that defines the protocol for an `Interface`. The statemachine comprises of `Regions`, which comprise of multiple `Vertices` and `Transitions`. A `Vertex` can be a `State`, a `Pseudostate`, or a `ConnectionPointReference`. The actual state of the component is

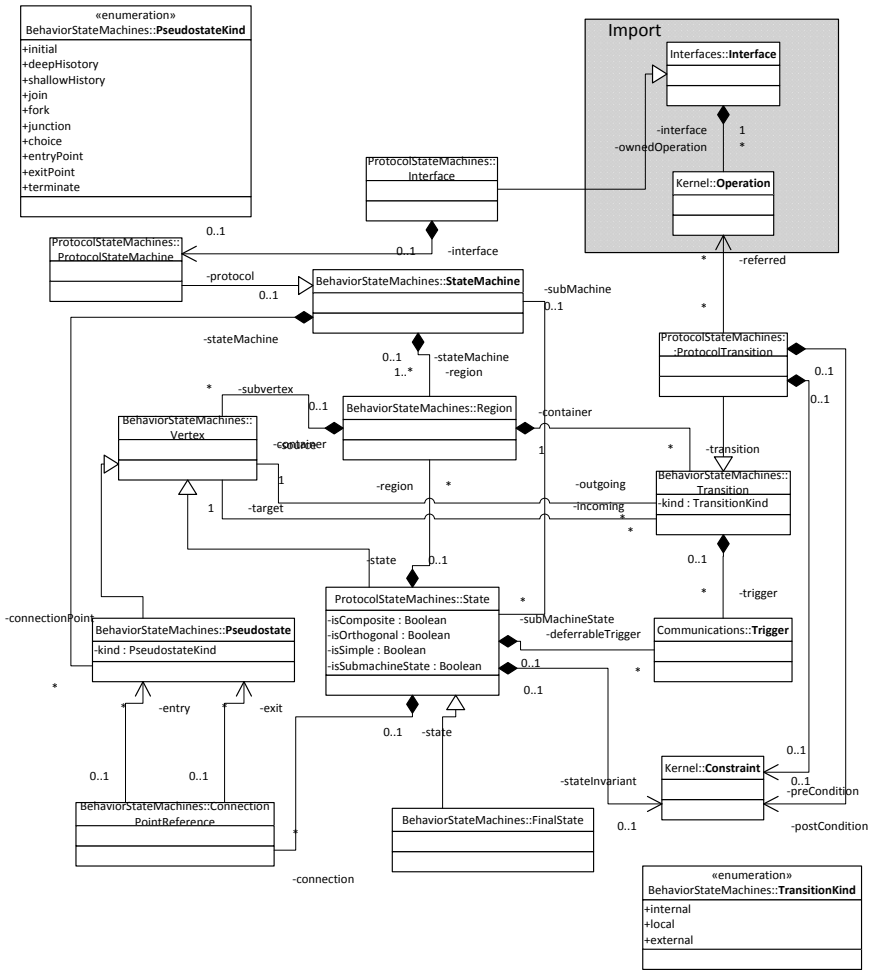

Figure 3.7: Excerpt of `ProviderProtocol` Package in the RSDL Metamodel based on UML statemachine diagram

modeled through `State`, which can be `simple`, `composed` or `submachine`. Additionally, different kinds of transient vertices which are used to connect multiple transition in a complex path, e.g., `join`, `fork`, `choice`, etc. are modeled through the abstraction `Pseudostate`. The initialization and termination of the regions in the statemachine are represented through `initial` pseudostate and `FinalState`, respectively. The transitions in the protocol statemachine are of a specialized type, i.e., `ProtocolTransition`. Each of these transitions refer to an `Operation` of the `Interface`. As a result of the

call `Trigger` for this `Transition`, the referred `Operation` will be called in the source `State` under the `preCondition` and at the end of the transition, the target `State` will be reached under the `PostCondition`. The referred `Operation` and its containing `Interface` are imported from `Operation-Signatures` package.

For the provider protocol, an invocation sequence that starts from the initial state and ends in the final state depicts a complete use case offered by the service where reaching the final state guarantees its successful completion. This means that if the final state is not achieved, this means that the use case is partially invoked and the validity of the results achieved through the service is not guaranteed in this case.

In the next section, we further elaborate the syntax of RSDL through a concrete example from our running example.

3.2.3 RSDL Service Descriptions for the running Example

After the specification of the RSDL in terms of its abstract syntax, we specify its concrete syntax through our running example in this section. Fig. 3.8 shows the RSDL service request for *HRS*. The data model is specified as a UML class diagram (Fig. 3.8(a)) that describes the classes, their attributes and their associations relevant for the *HRS* in the given scenario. The functionality that HRS requires from a service in order to fulfill his designed trip planning scenario is firstly specified in terms of operations of the required service `TripPlanner`. Fig. 3.8(b) shows these required operation signatures specified in terms of their names and input/output parameters that are typed over the classes in the corresponding data model and the XML data types.

Additionally, for each of these operations, their behavioral semantics are specified through visual contracts. A VC for a requested operation is interpreted in this way that for the particular operation, the requester guarantees to fulfill the preconditions specified in its VC and in return it requires that the specified postconditions are met. For instance, Fig. 3.8(c) shows the visual contract for the operation `makeReservation(...)` that is required to carry out a hotel reservation for a client based on his credentials and returns created hotel reservation. In this case, HRS ensures in the preconditions that there is a `Client` with its contact details as `Contact`, who has selected a certain `RoomStay` searched earlier. Additionally, the client provides credentials for his `PaymentCard`, which is required while doing online reservation. As a result of the invocation of the required operation, HRS expects that the specified postconditions are fulfilled, i.e, selected `RoomStay` is reserved for

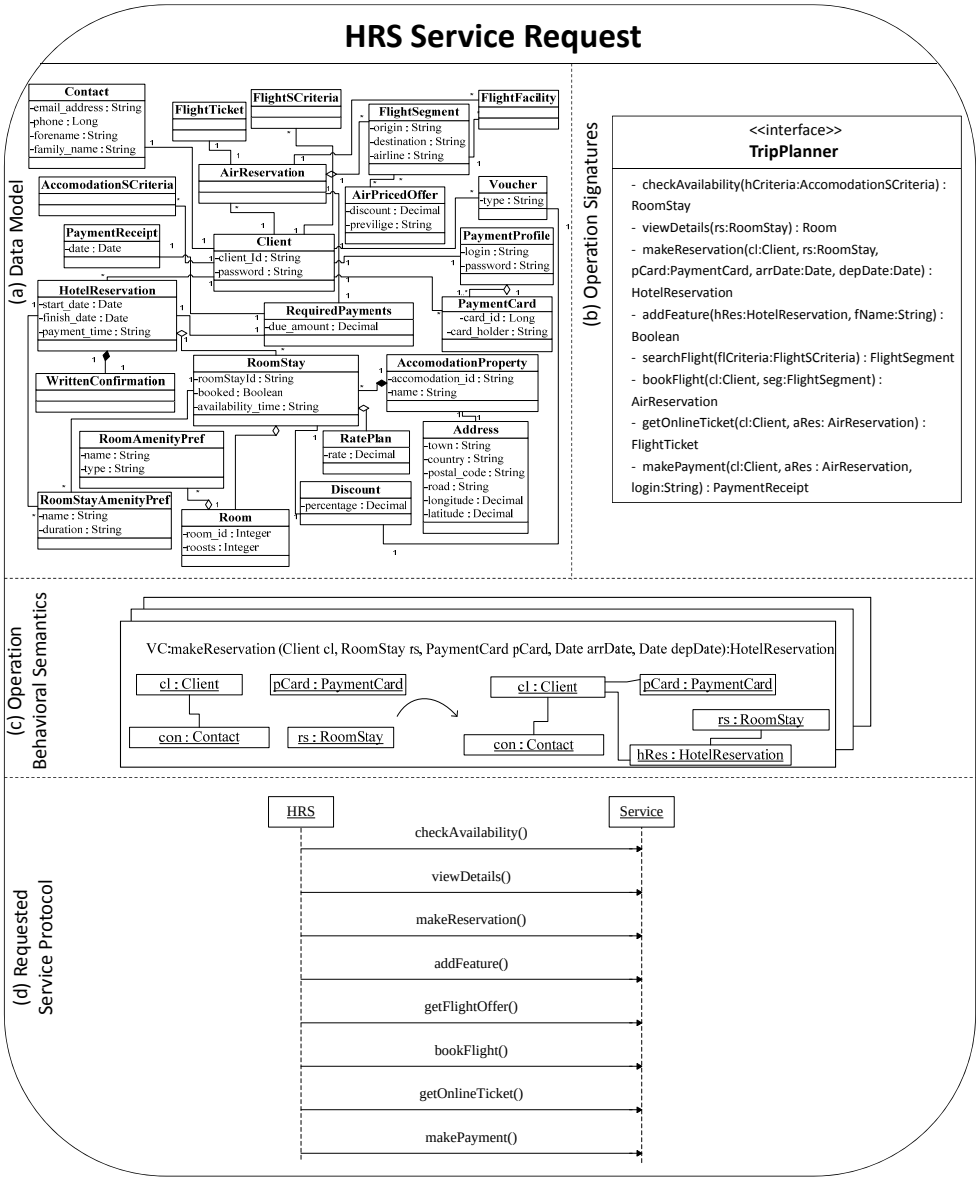

Figure 3.8: RSDL-based Service request by HRS

the `Client`. Consequently, a `HotelReservation` is created. Additionally, the provided credentials are validated by the system in order to be used for the hotel reservation and as a result the `PaymentCard` is associated to the

particular `Client`.

Additionally, HRS specifies its required service protocol shown in Fig. 3.8(d), which specifies the invocation sequence of the required operations that HRS expects the *TripPlanner* to offer.

This RSDL request of HRS is based on the independent knowledge of HRS application developer about the tourism domain. In this direction, the HRS application developer so far does not conform to a formal local ontology for the explicit specification of his domain knowledge. Hence, the HRS request does not comprise the ontological semantics at the time of its definition. However, during service discovery and composition process, our approach enables an automatic annotation of the service descriptions based on the global ontology of the OTF provider, which later serve as the basis to resolve their data model heterogeneity. This aspect will be elaborated in detail in Chap. 5.

Similar to the HRS request defined using RSDL, Fig. 3.9 shows the RSDL service offer for one of the hotel services *HotelX* published on the OTF Computing service market. Service offer of *HotelX* also comprises its data model (Fig. 3.9(a)), the offered functionality in terms of the offered operation signatures (Fig. 3.9(b)), the behavioral semantics for the offered operations in terms of their respective visual contracts (Fig. 3.9(c)), and the offered service protocol comprising all possible operation invocation sequences offered by *HotelX* service (Fig. 3.9(d)).

In this direction, the VC of an offered operation is interpreted differently from the VC of a requested operation. The VC of an offered operation specifies the preconditions that the service provider requires to be met in order to invoke the given operation. In return, it guarantees to fulfill the postconditions specified in the VC. For instance, Fig. 3.9(c) shows the VC for the offered operation `searchRoom(...)`. For the invocation of this operation, the service provider expects that a `Customer` already exists with his search criteria as `HotelCriteria`. If these preconditions are met, the offered operation can be invoked and the specified postconditions are guaranteed to be fulfilled, i.e., a `Package` belonging to a certain `Hotel` is searched with its further details, such as, the particular `Room` with its details, its `Price` structure, etc.

According to the offered service protocol, HotelX offers two main use cases, i.e., booking of hotel room package and booking of hotel facilities for some event. The basic activities carried out in these two different booking scenarios are contained in the composite states *s1* and *s5*. Each of these use cases have different variants, e.g., for the room booking use case, the customer has the option to also get a voucher if applicable to get any possible

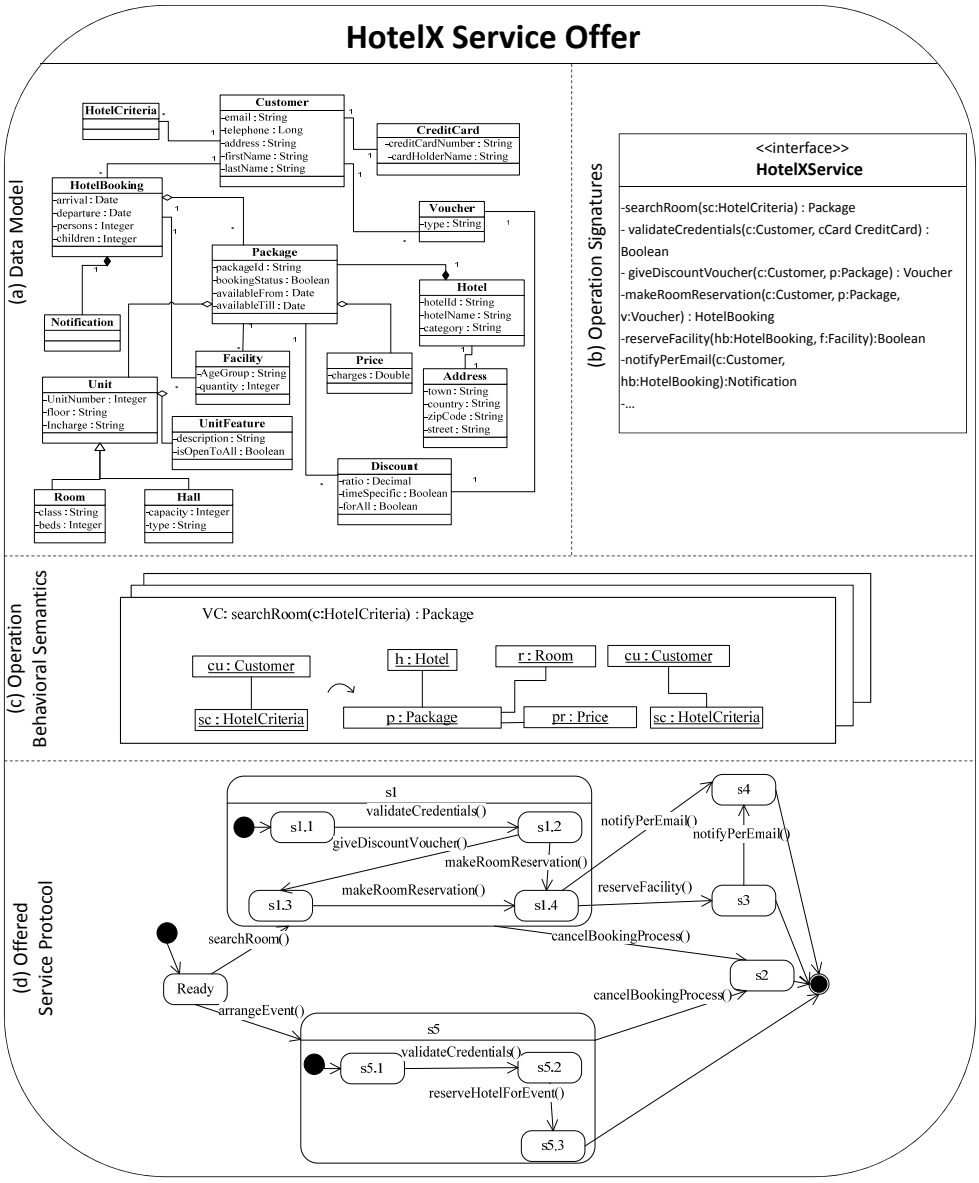

Figure 3.9: RSDL-based Service offer by HotelX

discount for his room booking. Similarly, at the end of the booking process, he also has the option to get a notification per email if required. Additionally, for both the offered use cases, the customer has the option to cancel the booking process at any time. Each of these possible use cases is specified

as an invocation sequence starting from the initial state and ending in the final state of the statemachine.

HotelX also does not conform to any formal local ontology for its service description and its ontological semantics are defined on the basis the global ontology when it is published on the service market.

3.3 Summary and Discussion

In this chapter, we introduced the rich service description language (RSDL), which allows the service partners to comprehensively specify the functional aspects of their service requests and offers.

Based on the criteria introduced in Chap. 2, we derived the requirements for a potential service description language. Next, we introduced the proposed RSDL in terms of its overall structure. As RSDL is based on UML as the underlying language, the abstract syntax of RSDL is specified as a metamodel mainly based on the UML metamodel [115]. The concrete syntax of RSDL is also based on the standard concrete syntax of UML and VC and is described on the basis of examples from our tourism example in OTF computing.

In the context of OTF Computing, such a comprehensive specification of the service requests and offers enables us to define our automatic mechanism for service discovery and composition. In the next chapter, we introduce the formal semantics of RSDL based on the approaches using typed graph transformation rules for this purpose [96, 137, 110].

4 Semantics of RSDL

After defining the syntax for RSDL, we define its linguistic semantics in this Chapter. Fig. 4.1 gives an overview of our approach for this linguistic semantic definition. The semantics of RSDL are mainly based on the semantics of its underlying language UML. Its semantics are a combination of informal and formal semantics for its different artifacts. For *data model* and *operation signatures*, the semantics are described informally conforming to the semantics of UML Class Diagram specified in UML Specification [115]. Additionally, the data model is also formally specified as an attributed type graph [33, 66], which serves as the basis for the formal semantics of *operation behavioral semantics*. The operation behavioral semantics are described in terms of graph transformation rules conforming to the semantics of visual contracts specified in [96]. For *service protocol*, we rely on the Dynamic Meta Modeling (DMM) approach [64] to describe the semantics in terms of labeled transition system (LTS). A detailed semantic specification for each of these artifacts is given in the following sections.

4.1 Semantics for the RSDL Data Model and Operation Signatures

The semantics for the RSDL data model and the operation signatures conform to the underlying UML specification [115], which are briefly discussed here and we refer the reader to [115] for further details, if desired.

In the context of the RSDL data model, a *classifier* defines a particular type and can be a *class* or a predefined *data type*. A *class* is meant to specify a classification of objects and the features that characterizes the structure and behavior of those objects. When an object of a class is instantiated, the attributes are also instantiated on the basis of the given initial value. If no initial value is given, the default value specification of that attribute is

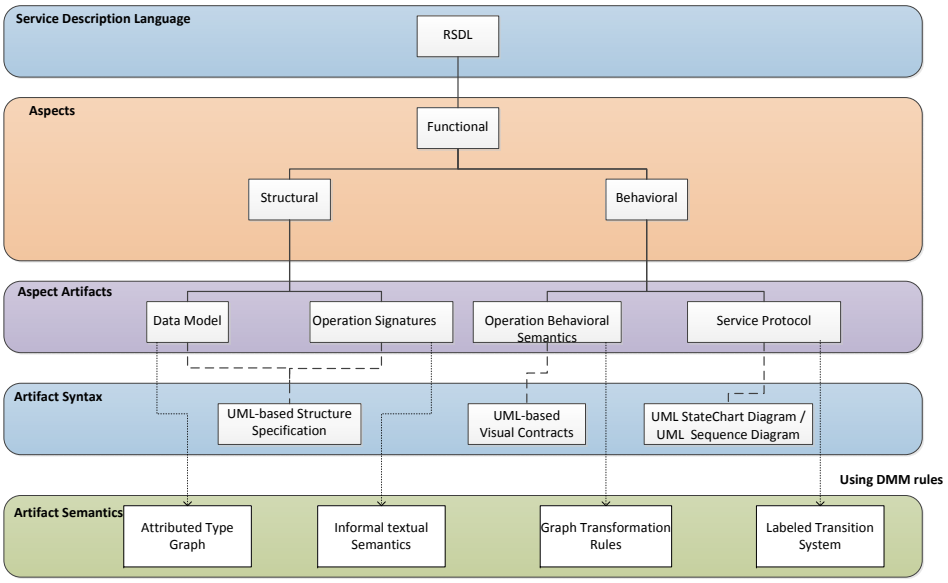

Figure 4.1: The hierarchical Structure of RSDL with Semantic Specification for different Artifacts

evaluated to set its initial value. The value for an attribute of an object is in accordance with its type and multiplicity. Similar to class, a *data type* also specifies an object classification but it has a basic difference from class, i.e., the instances of a data type are identified only by their value. This means that all the instances of a data type having the same value are considered to be the same instance.

A *generalization* relationship between two classifiers means that each instance of the *specific* classifier is also an instance of the *general* classifier and hence the features specified for the instances of the general classifier are implicitly specified for the instances of the specific classifier. Similarly, the constraints applicable to the general classifier are also valid for the instances of the specific classifier.

An *association* relationship between any classifiers means that there can be a link between the instances of the associated types. This link is represented through a tuple of the instances for each association end. An association may also represent a *shared* or a *composite* aggregation where a composite aggregation is a stronger form of an aggregation. It requires that an instance of the *part* classifier must be included in at most one instance of a *composite* classifier at a time. When a composite instance is deleted,

all its parts instances are also deleted.

For RSDL operation signatures, an interface declares a set of public features and obligations of a classifier representing the required/offered service. A realizing instance of the classifier has to publicly expose the properties conforming to the interface. As the interface is only a declaration, its instances cannot be created at runtime. The set of interfaces realized by a classifier can be termed as *provided* interfaces specifying the features that its instance offers to the client or *required* interfaces specifying the features that it requires to perform its functions. The operations owned by the interface are the behavioral features that can be invoked on the instance realizing the interface. The parameters of an operation specify how the arguments are passed in and out of that operation. The passed values can be restricted through the type and multiplicity of the parameters. At the invocation time of an operation, either the supplied value or in case when no value is supplied, the default value is evaluated and passed as an argument for each parameter. The parameter direction determines whether its value is passed into or out of the operation. The type of the operation is same as the type of its return parameter.

In addition to these informal semantics, we also formally specify the RSDL data model, which serves as a basis to define the formal semantics of the other RSDL artifacts. In literature, there are already different approaches that aim at a formal description of data model in general and UML class diagram in particular [151, 48, 96]. For instance, [151] specifies a class diagram and its semantics mathematically on the basis of set theory.

Particularly for our work, we select the formalization of a UML class diagram defined in [96]. It specifies a class diagram as an *attributed type graph* and use this formalization as a basis to define the formal semantics for visual contracts as graph transformation rules. Using graph-based formalism for our approach is a conscious decision because of its preciseness as well as understandability. Similarly, these visual and model-based semantic specifications integrate well with the concept of model-based system design and development.

For a more formal and theoretical account of the relevant graph theory concepts, we refer the reader to [33, 66, 96]. Here, we aim to provide a conceptual overview with the help of examples to explain them in the context of RSDL.

A *graph* can be simply described as a structure comprising of nodes and links, where each link connects a source node to a target node. In the software engineering paradigm, the graphs are differentiated as *type graph* and *instance/typed graph*. An example for this differentiation is a UML class dia-

gram as type graph and the corresponding object diagram as instance graph. In the graph theory, this object-oriented concept is introduced through *typed graphs* [33, 66], where the relation between a type graph and typed graph is specified through a mapping, i.e., a *graph homomorphism* between the two graphs. Based on this mapping, the nodes and links in a typed graph are associated to their respective types in a type graph.

Another important and relevant object-oriented concept is the attributes of the classes defined through their names and types. This is also catered in the graph theory domain through *attributed graphs* [34]. Informally, an attributed graph can be described as a graph comprising of a simple graph with object nodes and links and a graph for the representation of attribute values. The graph for attribute representation comprises of *data nodes*, which represent the attribute values conforming to their data types and are different from *object nodes*. An attribute in the attributed graph is specified through an *attribute link* from an object node to a data node.

Based on this, an attributed instance graph is typed over an attributed type graph.

In accordance to these concepts from graph theory, elements of a class diagram correspond to the elements of an attributed type graph as follows:

- A class in the class diagram is represented as a node in the type graph.
- An attribute in the class diagram is represented as an attribute link to a data type.
- Associations in the class diagram are represented as links in the type graph.

We explain this with the help of an example shown in Fig. 4.2. On the left hand side, an excerpt of the HRS data model is shown in the upper part and the corresponding type graph is shown in the lower part. For the attributes `email_address` and `client_id`, the value can be an element from the set containing values for type `String`. Therefore, these attributes are linked to the type `String` in the type graph. Similarly, attribute `phone` is linked to type `Long`. On the right hand side of the figure, the upper part shows an object diagram and the lower part shows the corresponding attributed typed graph. This typed graph includes data nodes that are linked to object nodes through attribute links. For instance, `c:Client` in the typed graph is typed over `Client` in the type graph and the attribute links in the typed graphs are connected to the data nodes containing their respective values.

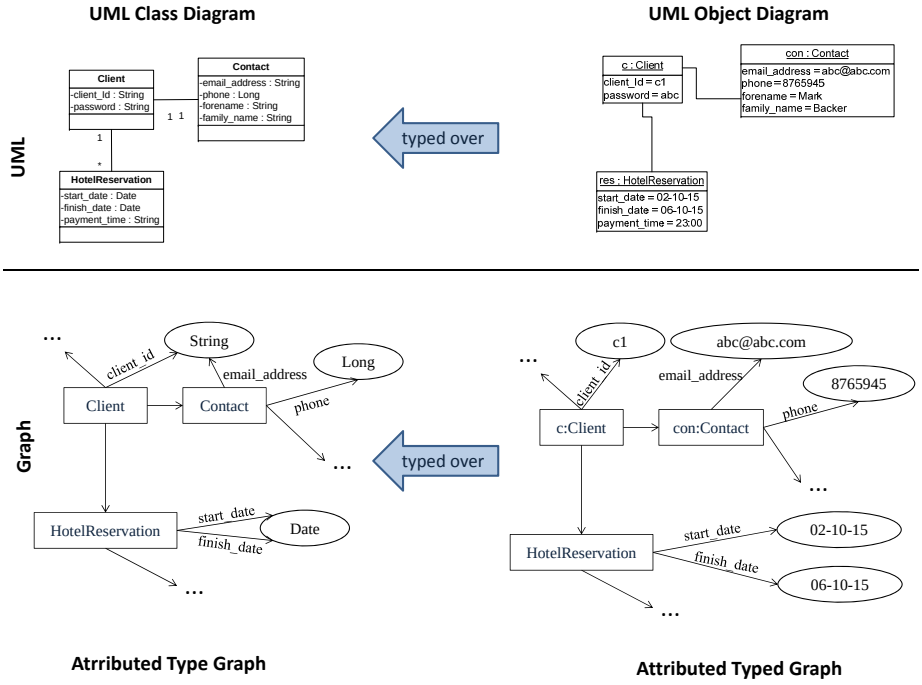

Figure 4.2: UML Class Diagram and UML Object Diagram vs. Attributed Typed Graph and Typed Graph

Based on this formalization of RSDL data model, the formal semantics for the operation behavioral semantics in RSDL are presented in the next section.

4.2 Semantics of RSDL Operation Behavioral Semantics

To define the semantics for the RSDL operation behavioral semantics, we rely on the graph theory-based [134, 46] semantics defined in [96] for visual contracts. According to this approach, a visual contract typed over a class diagram is interpreted as a graph transformation rule comprising *attributed typed graphs* typed over a type graph.

The general idea is that the structure of a graph can be changed through a *graph transformation* and these changes are specified as *graph transformation rules* [34]. A graph transformation rules comprises of a pair of graphs representing the left and right-hand sides of the rule. A graph transforma-

tion is applicable on a host graph if it contains a match for the left-hand side of the rule, i.e., the structure specified on the left-hand side of the rule is contained in the host graph. As a result of the application of the rule, the matched part of the host graph is replaced with the right-hand side of the rule.

This concept of graph transformation can also be used for attributed graphs to describe the state changes of a system specified through the visual contracts. As a continuation for our earlier discussion about the attributed type graph and instance graph, the elements of a visual contract correspond to the elements of graph transformation rule for attributed typed graph as follows:

- The preconditions and postconditions in the visual contract are represented as the attributed typed graphs on the left- and right-hand side of a graph transformation rule, respectively.
- A role in the preconditions or postconditions in the visual contract is represented as a node in the typed graph.
- An attribute of a role, which is assigned a value is represented as an attribute link to a data node in the typed graph.
- Links in the preconditions or postconditions are represented as the links in the corresponding typed graph.

Formal definitions of these concepts and further details of this interrelation can be seen in [96].

There are two classic approaches for the interpretation of graph transformations: Double-Pushout (DPO) [34] and Double Pullback (DPB) [67]. DPO defines a strict interpretation stating that a graph transformation rule completely specifies the transformation of a graph. This means that during the execution of the rule, no other changes occur in the graph except those that are explicitly specified in the rule. This strict stance is to make sure that the resulting graph after the transformation is still a *legal* graph and it contains not dangling edges without a source or a target node.

In the context of visual contracts, a VC is conceptualized as a mean for system specification extensively used during system analysis and design phase on a higher abstraction level. Hence, it specifies the minimum set of changes that should be made to the system state as a result of the operation execution. However, it is possible that according to the implementation details, there are additional effects that occur as a result of the execution of the operation at runtime. For instance, an instance of a further class can be

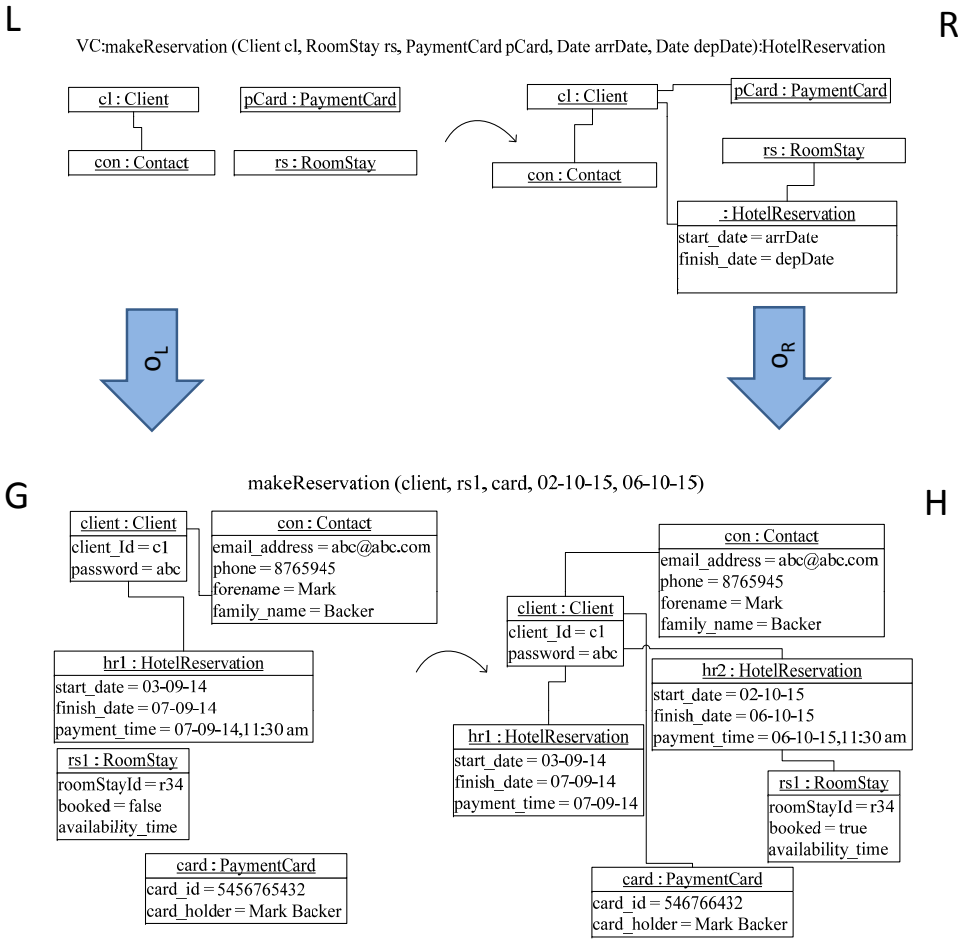

Figure 4.3: DPB-based graph transformation for makeReservation(...)

created supporting the implementation. Considering this loose interpretation of visual contracts and the strict stance of DPO, DPO is not a suitable interpretation for visual contracts.

Unlike DPO, DPB offers a more relaxed interpretation of graph transformation, which claims that a graph transformation rule only specifies the minimum set of changes that should be made to the host graph on the execution of the rule. This means that as a result of the transformation, there can be changes in the system state other than the ones specified in the executed graph transformation rule.

According to the conceptualization of VC, the concept of DPB-based graph transformations for the attributed typed graph is a suitable mean

to specify the semantics of visual contracts. Conforming to the formal set theory-based definition of a DPB-based graph transformation in [96], we explain such a graph transformation on the basis of the example shown in Fig. 4.3.

For the graph transformation rule with left and right-hand side specified as `L` and `R`, the system state before and after its application is shown as `G` and `H`, respectively. The graph transformation is applied in three steps: First, through a subgraph isomorphism `o` [34], subgraph o_L based on the left-hand side of the graph transformation rule should be found in the given graph `G`. In the given example, it can be seen that a subgraph comprising all the nodes and links specified in the left-hand side of the rule is indeed present in the actual graph `G`. Though `G` contains further elements that are not specified in the rule. Second, *at least* all the graph elements, i.e., nodes, links and attributes (attribute links) that correspond to $L \setminus R$, are deleted. In the given example, nothing is deleted in the rule, i.e., $L \setminus R$ is empty so the graph `G` remains the same. Third, the resulting graph from the second step should be combined *at least* with the graph corresponding to $R \setminus L$. This includes creation of nodes, links and attribute links that correspond to $R \setminus L$. The resulting graph `H` contains the subgraph o_R based the right-hand side of the rule, e.g., a node of the type `HotelReservation` is created, a link between the node of type `Client` and the newly created `HotelReservation` node is create, the attributes of the `HotelReservation` node are assigned values based on the parameter values `arrDate` and `depDate` of the rule. In addition to the effects specified in the given graph transformation rule, there may be other changes to the system state as well according to the DPB approach, e.g., the payment time for a hotel reservation is also set based on the departure date, i.e., the attribute `payment_time` in `HotelReservation` node is assigned a value. Similarly, the attribute `booked` for `RoomStay` is set to `true` notifying its status.

As a result of these three steps, the system state after the rule application is specified through `H`. In this case, the subgraph o_R from the right-hand side of the rule is indeed present in `H`. However based on the lose semantics concept of DPB, apart from the effects specified in the graph transformation rule, there are other effects as well that are present in `H`.

Here it is important to mention that as RSDL is a service description language and is not meant to be executed, such a runtime application of the VCs as graph transformation rules does not actually occur. However, this semantic definition is important to precisely define the meaning of the VCs as the behavioral specification of the operations comprising an RSDL service description.

In the next sections, we specify the formal semantics for RSDL service protocols.

4.3 Semantics of RSDL Service Protocol

For the semantic specification of RSDL service protocol, we select existing graph transformation-based approaches [137, 110]. These approaches allow the definition of the semantics by applying the Dynamic Meta Modeling (DMM) approach [64]. Before introducing these approaches, we first introduce the dynamic meta modeling (DMM) approach and its key concepts.

4.3.1 Dynamic Meta Modeling (DMM)

The DMM approach [64] is developed to allow a formal and precise specification of the semantics of a modeling language, which are also understandable at the same time. This approach is based on graph transformations and is particularly targeted at the modeling languages whose syntax is defined in terms of a metamodel. The two basic elements of DMM-based semantic specification are: *runtime metamodel* and *DMM rule set*. To define the semantics of a modeling language, the DMM extends the metamodel of the language with concepts that are used to describe its dynamic semantics. Such an extended metamodel of the language is called its *runtime metamodel* in the DMM approach. In order to describe the runtime behavior, DMM rules are defined as typed graph transformation rules, where the typed graphs are typed over the runtime meta model. As a result of the application of the DMM rule set on the graph typed over the runtime metamodel, a transition system is achieved. The overall concept of the DMM approach is given in Fig. 4.4.

In the following, we briefly introduce the approaches [137, 110] that define the semantics of UML sequence diagrams and UML statemachine diagram through the DMM approach. This introduction is based on a brief overview of some important elements of the respective runtime metamodels and some example DMM rules from the respective rule set. For complete details of these approaches and detailed semantics specification, we refer the reader to [137, 110].

4.3.2 DMM for Requested Service Protocol

[137] defines a runtime metamodel for the DMM specification of UML sequence diagram, which is an extension of the its existing metamodel. Based

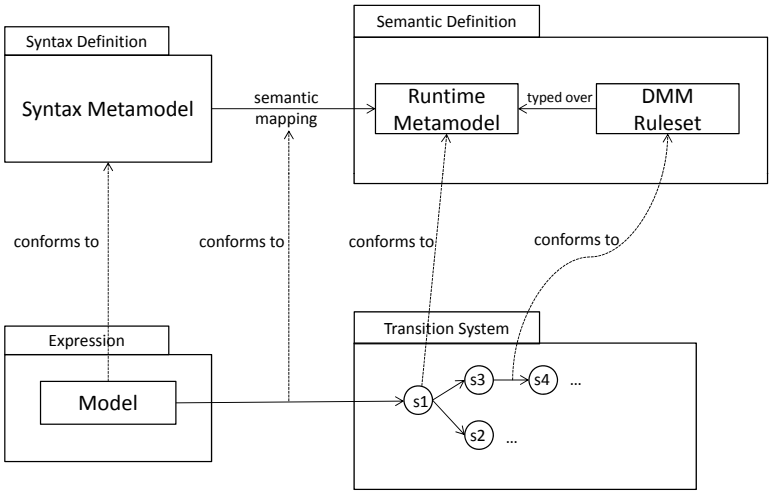

Figure 4.4: Overview of DMM Approach [64]

on this runtime metamodel, more than 250 DMM rules are defined to specify the behavior of the elements in UML sequence diagram.

Some of the important element from this runtime metamodel are the `InteractionExecution`, the `Sorter` and the `ActiveMessage`. They are mainly responsible for the management, basic functions and state specification of the sequence diagram. For instance, an `InteractionExecution` element is created for every `Interaction` element at runtime, which serves as a container for all the semantic elements and maintains the execution state of an Interaction. Similarly, the `Sorter` is another important element used by `InteractionExecution` to sort the execution of different `MessageEvents` on the `LifeLine`. For this purpose, it extracts the required information from the metamodel of the sequence diagram. `ActiveMessage` is created at runtime for `Messages` and is used for the identification of the execution state for the message.

As an example, we show the DMM rule for the start of an `Interaction` in Fig. 4.5.

In the runtime model of the a UML sequence diagram, an `InteractionExecution` element is created for every `Interaction` element in the instance model. At this time, the `InteractionExecution` of the hierarchically highest `Interaction` in the diagram is activated by setting the attribute `activated` to `true`. In this way, the execution of behavior starts from the execution of this particular interaction at runtime. An `InteractionExecution` starts by setting the `hasStarted` to `true`. As

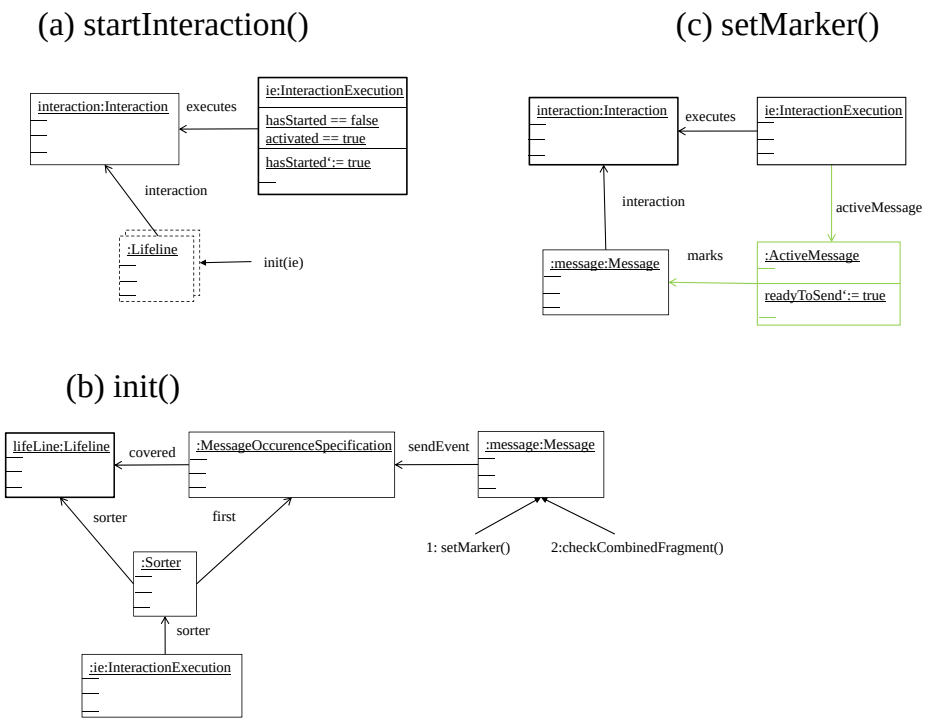

Figure 4.5: (a)DMM Rule `startInteraction()` to start an Interaction (b) `init()` to initialize `Active Message` Marker (c) `setMarker()` to create `ActiveMessage` Marker

a result, the corresponding `Interaction` starts. This is handled by the `startInteraction()` rule as shown in Fig. 4.5(a). This rule is invoked, as soon as an `InteractionExecution` is activated. As a result of this graph transformation, `hasStarted` is set to `true` and through the invocation of another rule `init()`, the event that starts the `Interaction` is determined. For instance, `init()` in Fig. 4.5(b) shows the case where an `Interaction` starts with a `Message` invocation. In this case a `MessageOccurrenceSpecification` which has to be a `sendEvent` and is classified as the first event on a `Lifeline` by its `Sorter` is determined. As a result of this graph transformation, an `ActiveMessage` marker is created for the `Message` with its `readyToSend` attribute set to `true` (rule `setMarker()` Fig. 4.5(c)). Consequently, the message is ready to be sent and the interaction can start.

Apart from `startInteraction()` rule, other basic DMM rules for UML sequence diagram include rules for message sending, message receiving, mes-

sage sequencing, etc.

4.3.3 DMM for Offered Service Protocol

For the UML statemachine diagram, [110] defines a runtime metamodel for the DMM specification of UML statemachine diagram, which is an extension of its existing metamodel. Based on this runtime metamodel, more than 90 DMM rules are defined to specify the behavior of the elements in UML statemachine diagram.

Similar to the runtime metamodel of the UML sequence diagram, the basic elements of the runtime metamodel for the UML statemachine diagram are the `StateMachineExecution` and the `Marker`. In this context, the `StateMachineExecution` is created for a `StateMachine` at runtime and serves as a container for the semantic elements. Additionally, the `Marker` element is used to control the sequence of the execution of the individual states and the transitions in the statemachine.

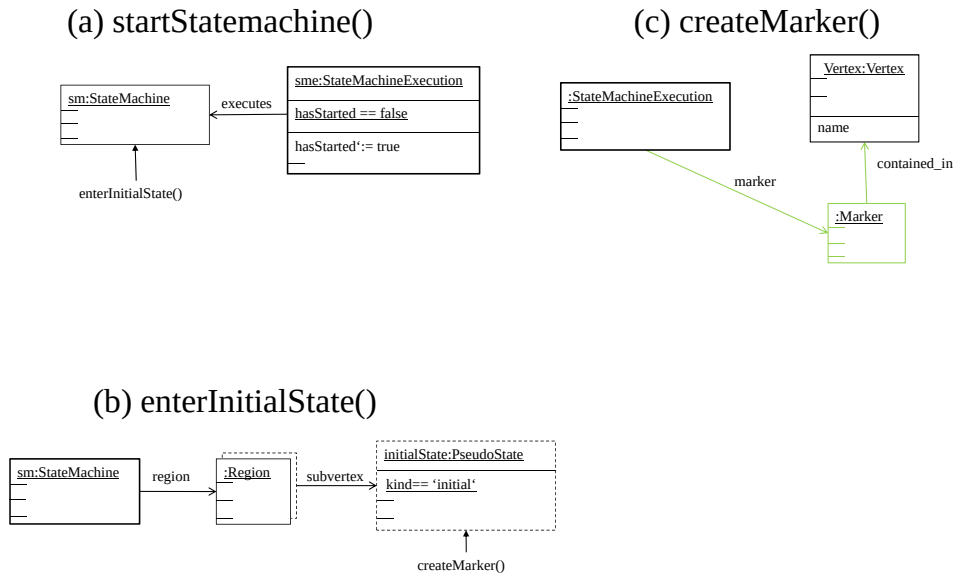

Figure 4.6: (a)DMM Rule `startStatemachine()` to start a Statemachine (b) `enterInitialState()` to enter the Statemachine in its initial State (c) `createMarker()` to create a Marker for a particular `Vertex`

As an example for a DMM rule to specify the behavior of a UML statemachine, we show the rule namely `startStateMachine()` in Fig. 4.6(a) for the

start of a `StateMachine`. In the runtime model for a UML statemachine diagram, a `StateMachineExecution` is created for `StateMachine` element with `hasStarted` attribute set to `false`. Similar to the `InteractionExecution` in the UML sequence diagram runtime model, a statemachine can start only if its `hasStarted` attribute is set to `true` as shown in Fig. 4.6(a). As a result of this graph transformation, the `hasStarted` attribute is set to `true`, which means that the statemachine has started and every orthogonal `Region` in the statemachine is in its `initial` state (through the invocation of `enterInitialState()` in Fig. 4.6(b)). Additionally, through the invocation of `createMarker()` (Fig. 4.6(c)), a `Marker` is created that points to the `Vertex` on which this rule is invoked (the initial state in this case). As a result of the invocation of this rule, the initial states have markers and the statemachine can start.

Apart from these rules, other basic DMM rules for UML statemachine diagram include rules for firing transitions, state execution, etc.

4.4 Summary and Discussion

In this chapter, we define the linguistic semantics for RSDL. For this purpose, we mainly rely on existing approaches in this direction. A collective basis for these semantics for different RSDL artifacts are graph transformation rules. In this direction, the RSDL data model is formally specified as a type graph and the operation behavioral semantics based on visual contracts are specified as typed graph transformation rules. Similarly, the DMM approach [64] is used for the semantic specification of service protocols, which focus the modeling languages like UML and specify their semantics through graph transformation rules typed over their runtime meta models.

5

Service Description Normalization through Data Model Matching

Service description normalization is the first phase of our service discovery and composition approach and it deals with the resolution of the data model heterogeneity of the service partners. That means whenever a requester accesses the service market with his request to search for possible service compositions, its service request is automatically annotated to semantic concepts on the basis of a global ontology maintained by the OTF provider. Later, on the basis of these ontological semantics, its data model is mapped to the global data model conforming to the global ontology. Consequently, these local-global data model mappings are used to translate the service request to a common representation typed over the global data model.

Analogously, a service description normalization is also performed, when a service provider accesses the OTF service market with his offer to publish his service. Our data model heterogeneity resolution mechanism is based on our detailed work in this direction presented in [140].

In the following section, we give an overview of our service description normalization approach. Later, we lay foundations for our mechanism through the explanation of some important concepts. In Sec. 5.3, we present our data model matching algorithm. Next, a service description normalization approach is introduced, which allows the normalization of the service description to a common representation based on the data model matching results.

5.1 Service Description Normalization Overview

An important aim of the OTF computing is that it enables the coordination among the heterogeneous service partners that function in their independent domains with their individual domain knowledge.

Consequently, such an independence of the service partners leads to the *data model heterogeneity* of their service descriptions, which can make their automatic matching during service discovery and composition difficult. For instance, a simple scenario that can lead to data model heterogeneity of service partners is their use of different terminologies to define their data models, e.g., a class defined as `Client` in the requester data model may be defined as a class `User` in the provider data model. Similarly, the data model heterogeneity can also arise due to different granularity levels of the elements in the data models, e.g., a concept `Address` in the requester data model may correspond to two concepts `Address` and `Coordinates` in the provider data model. An important task for the OTF provider is to automatically resolve this data model heterogeneity of the requested and offered service descriptions in order to enable their automatic matching on the OTF service market. For instance, approaches like [65, 108, 157] come up with comprehensive mechanisms for service discovery but ignore this important aspect altogether by using same data model.

However, as discussed in Sec. 2.3.1, a recent trend in the SOC is to explicitly define ontological semantics and use them for the matching of data elements in the service descriptions. In this context, the semantic web service approaches [121, 51, 39, 98, 136] particularly emphasize and present different mechanisms to define such ontological semantics for the service descriptions. In recent years, a considerably large number of approaches [14, 88, 31, 37, 81, 79, 100, 95, 127, 153] in the area of automatic service discovery and composition have defined mechanisms to match service descriptions while considering their ontological semantics. However, most of these approaches do not solve the problem as they are based on the assumption that a common ontology exists in the service market, which is shared by all the service partners. Conforming to this common ontology, the service partners describe their respective service descriptions and their ontological semantics, which are later used to match these service descriptions. However, according to the essence of SOC, such an assumption is not realistic and there is a requirement for approaches that deal with data model heterogeneity while allowing the service partners to conform to their respective domain knowledge, which can possibly be specified as their independent local ontologies.

As an improvement to these existing approaches, our approach allows the service partners to independently define their service descriptions in their respective domain and introduces an automatic mechanism for the OTF provider to resolve the data model heterogeneity of the service partners based on their automatically defined ontological semantics and bring them to a common representation before their matching. In this chapter, we will

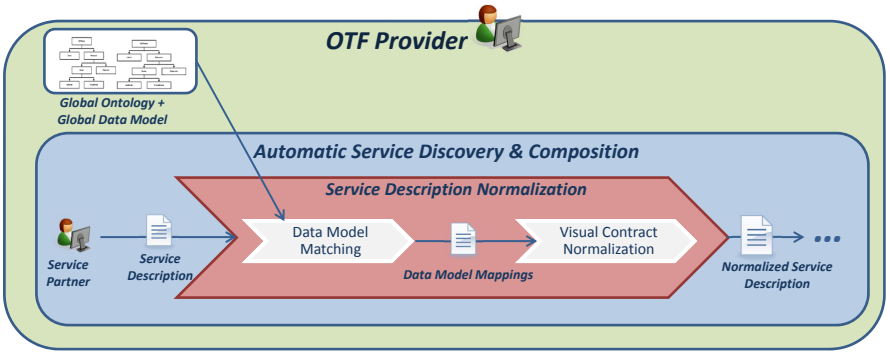

Figure 5.1: An Overview of the Service Description Normalization Phase in the Proposed Approach

explain different steps of this approach in detail in the context of our running example.

Fig. 5.1 gives an overview of this mechanism. In the setting of OTF computing, the OTF provider maintains a global ontology that comprehensively covers the information domain under consideration and captures the domain knowledge in an extensive manner. Additionally, a conforming global data model is also defined to define the structure of data elements in the global domain. The automatic mechanism for data model heterogeneity resolution is initiated when a service requester or provider access the OTF market with its service request or offer, respectively.

In the first step of this phase, the local data model of the service partner is matched to the global data model of the OTF provider. This automatic matching mechanism, which is explained in Sec. 5.2 and Sec. 5.3 is based on the structural as well as the semantic matching of the local and global data model elements based on the ontological semantics defined through the global ontology. The result of this step are the mappings between the local and the global data models.

In the second step of this phase, the service description of the particular service partner is normalized on the basis of the local-global data model mappings. This normalization of the service description is mainly concerned with the operations in the considered service description because the data model heterogeneity of the service partners mainly complicates the matching of their operations, whose structural and behavioral descriptions are typed over the respective data models.

The operation normalization in the considered service description can be brought down to the normalization of their behavioral descriptions, i.e., the

visual contracts. As far as the normalization of structural descriptions, i.e., operation signatures is concerned, the input/output parameters are included in the respective VC according to the RSDL specification in Chap. 3 and hence do not need to be normalized separately. The normalization of the operation names is not relevant for our approach as our n : m operation matching mechanism mainly relies on behavioral descriptions and does not consider the operation names while matching.

For the service protocol, which describes the operation invocation sequences in the considered service description, operation normalization suffices to overcome the data model heterogeneity and hence no further normalization is required for service protocol.

As a result of the service description normalization, the service requests and offers are typed over the global data model. Consequently, the *normalized* service request can be matched to the *normalized* service offers on the service market.

In the following sections, we describe these steps of the service description normalization phase in detail.

5.2 Data Model Matching - Foundations

Before going into the details of our algorithm for data model matching, it is important that we introduce some important concepts that lay the foundations for our data model matching mechanism. These concepts will be discussed in the following sections in detail.

5.2.1 Local-global Matching Approach

In order to overcome the heterogeneity of the data models, there are three different options specified as the matching techniques discussed in [163]: global, local and local-global. In the global approach, all the stakeholders in a system share a common data model conforming to a global ontology in the public domain. Hence, in this case the problem of data model heterogeneity does not arise. As mentioned earlier, this approach negates the very essence of SOC to allow the service partners to function independently. Hence, in this particular context, it is unrealistic to apply such a restriction on the service partners.

In the local approach, it is considered that each stakeholder of the system can have its own independent data model, which can possibly conform to its formal local ontology. In order to match the heterogeneous data models of the stakeholders, the data model of a stakeholder is matched to the data

model of every other stakeholder and the result is a mapping between every pair of the data models. In the context of SOC, this means that every time a service requester accesses the service market with its request, its data model is matched individually to the data model of every service offer available on the service market. Consequently, there will be an individual mapping for the requester data model with every offered data model. Although this approach allows the service partners to function independently in the SOC setting but with the ever-growing plethora of service requests and offers, such a data model matching approach is unrealistic and is difficult to scale with increasing number of service partners.

The third option is the local-global approach, where the stakeholders have their independent data models. Additionally, a global data model conforming to a global ontology is maintained in the public domain. In this case, the data model of every stakeholder has to be matched only once to the global data model. This means that before publishing a service offer on the service market, its local data model is matched to the global data model. Similarly, when a service requester accesses the market with its service request to search for suitable service offers, its data model is not matched with every offered data model individually but is only matched once to the global data model. Later, this local-global mapping can be used to match the service descriptions. In this case, the main challenge is the development and maintenance of a comprehensive global ontology and the global data model. If this challenge is met, this approach is most suitable as it reduces the complexity and makes the data model matching process highly scalable.

Being suitable for the highly dynamic and ever changing nature of the service market in OTF computing context, we select the local-global data model matching option for our approach.

5.2.2 Global Ontology and its conforming global Data Model

As explained in Chap. 1, an ontology [148] is a mean to explicitly and formally define the meanings of the terms and concepts in a particular domain, which results in the establishment of a common understanding among the stakeholders in that domain. In the context of the OTF computing, a global ontology has to be defined that represents a conceptualization of the domain by the OTF provider, which is shared by all the service partners.

Based on this understanding, our approach recommends the OTF provider to develop and maintain such a global ontology. Additionally, an OTF provider also maintains a global data model conforming to this ontol-

ogy to precisely represent the structure of the data elements in the public domain. This setting enables the normalization of a service request/offer typed over a local data model to a common representation typed over the global data model. In this direction, there are three main concerns for the OTF provider:

1. The development of a comprehensive global ontology;
2. The maintenance of the global ontology;
3. The development and maintenance of a global data model conforming to the global ontology.

These concerns are discussed in detail as follows.

Development of the global Ontology: In the OTF computing setting in this thesis, the OTF provider is responsible to cater to service requests and offers from diverse domains. For instance, in the running example, the service request of HRS and the required service offers are from tourism domain. Similarly, there can be another service discovery and composition scenario where the service request and offers belong to e-commerce domain. To cater to this scenario, the global ontology of OTF provider is developed as a combination of multiple ontologies.

An ontology can be mainly categorized as *general purpose ontology (GPO)* or a *domain specific ontology (DSO)* [53], where the former represents the common knowledge and the latter particularly focuses a particular domain of interest and represents the knowledge, i.e., the concepts, their relations and properties in the context of that particular domain.

The global ontology contains a *domain specific ontology (DSO)* for every domain catered by the OTF provider. In this direction, as the OTF provider aims to define a common understanding among for all the service partners through the global ontology, a comprising DSO has to holistically and comprehensively cover the information of the particular domain. In this context, the OTF provider can either manually develop a DSO with the help of a domain expert or can reuse some commercially available DSOs. For instance, ontologies like the Open Travel Alliance (OTA)[1], HarmoNET[2], Travel Guide[3], Accomodation[4] etc. that are commercially available in the tourism domain cover the knowledge in this domain quite comprehensively.

[1] http://www.opentravel.org
[2] http://www.harmonet.org
[3] https://sites.google.com/site/ontotravelguides
[4] http://ontologies.sti-innsbruck.at/acco/ns.owl

In a realistic setting, it is also possible that there are many commonalities and discrepancies among DSOs of different domains, e.g., the concept *accomodation* have totally different meanings in tourism and banking domains. Similarly, there are concepts, e.g., personal information concepts, which are same in different domains. If a service description spans over multiple domains of interest then these inter-connections between different DSOs have to be taken into account while defining its ontological semantics. However, for the work in this thesis, we restrict ourselves to the service discovery scenarios that do not span over multiple domains and are rather limited to a single domain. For instance, the running example is a service discovery scenario restricted to a single domain where the service request and offers belong to the tourism. In such cases, the definition of ontological semantics is concerned with the DSO of the particular domain and is not effected by its interconnections with other DSOs. Hence, the discrepancies, interconnections, and overlaps that can occur among different DSOs in the global ontology are not a focus of our work in this thesis.

In addition to its comprehensiveness in capturing domain information, there is another important aspect of a DSO in our approach. For our matching mechanism, it is also necessary that the DSO captures the domain information, i.e., the constituting concepts in a hierarchical classification based on their specialization-generalization relationship. Such a hierarchical classification of information is already an established concept in the ontology world [52], e.g., GPO like WordNet [130] capture the information where the depth of the hierarchy is 17. Similarly, DSOs like Travel Guide and HarmoNET, etc. also maintain a deep hierarchy of captured concepts. Ontology-based matching approaches [52], extract different kinds of information from this hierarchy, such as, number of common predecessors, number of common successors, etc. to determine the semantic distance of any two concepts. For our matching mechanism also, a deeper hierarchical structure leads to higher probability of accurate semantic matching between the elements of the local/global data models. This notion will be clarified further in the next section where we introduce our data model matching algorithm.

Maintenance of the Global Ontology: In a realistic scenario, it is possible that the OTF provider requires to update the global ontology. For instance, a situation can arise where some new information needs to be added to a DSO in the global ontology. A major concern in this direction is the maintenance of the existing mappings between the service descriptions and the global ontology. Some important questions that arise in this situation are: How can the existing ontology semantics of a service offer defined on the basis of the existing global ontology be translated to new seman-

tics based on the updated version of the ontology? How feasible is such a translation for all the existing mapping every time the ontology is updated? How are the different versions of the global ontology maintained in a consistent manner? This gives another dimension to the problem at hand, which can be categorized under the broader area of *model versioning*. The topic of model merging and model versioning is an extensively researched topic [56, 102], which we do not focus for the work in this thesis. Hence to avoid further complexity and to simplify the situation at hand, we only allow the udpate of the global ontology in a limited sense.

As mentioned earlier, we assume that the OTF provider builds the global ontology in a comprehensive manner, where each DSO holistically covers all the possible information of the particular domain. On the basis of this assumption, we claim that after its construction, the global ontology remains constant for the most part. This means that none of the existing information is removed or updated in the global ontology. However, addition of new information is allowed in a limited fashion.

Over the period of time, it can occur that new concepts are introduced in an information domain or the OTF provider decides to cater to some new information domain. In this case, the OTF provider can add new information to an existing DSO in the global ontology or can add a new DSO in the global ontology, respectively. In this case, such an extension of the global ontology is only allowed if it does not effect the existing semantic mappings for the service offers on the service market.

Development and Maintenance of the global Data Model: In order to materialize the local-global data model matching approach, the OTF provider also defines a global data model conforming to the global ontology. In this context, the co-existence of a global ontology and a global data model makes sense. The global ontology is meant to be a very detailed account of the domain knowledge. It particularly aims to capture the information that can support the semantic matching of concepts, e.g., the hierarchical structure, synonyms, homonyms, etc., are the different types of information, which are part of the ontology. On the other hand, for the definition of the global data model, the main aim is to achieve a standard representation of the structure of the data elements in the public domain. This structural representation is meant to be consistent and concise and does not encompass all the information of the ontology, e.g., the synonyms, homonyms or the hierarchical depth, etc.

The OTF provider defines and maintains the global data model as a UML class diagram, which is coherent with the domain knowledge of the OTF provider. This means that every element of the global data model

is annotated to a concept in the global ontology. Additionally, this global data model also has to be updated in case of the updation of the global ontology and the same restriction also apply in this case that inclusion of new elements in the data model should not effect the existing local-global data model mappings.

Fig. 5.2 shows the excerpt of a global data model for tourism domain, which conforms to a HarmoNET-based tourism DSO maintained by the OTF provider.

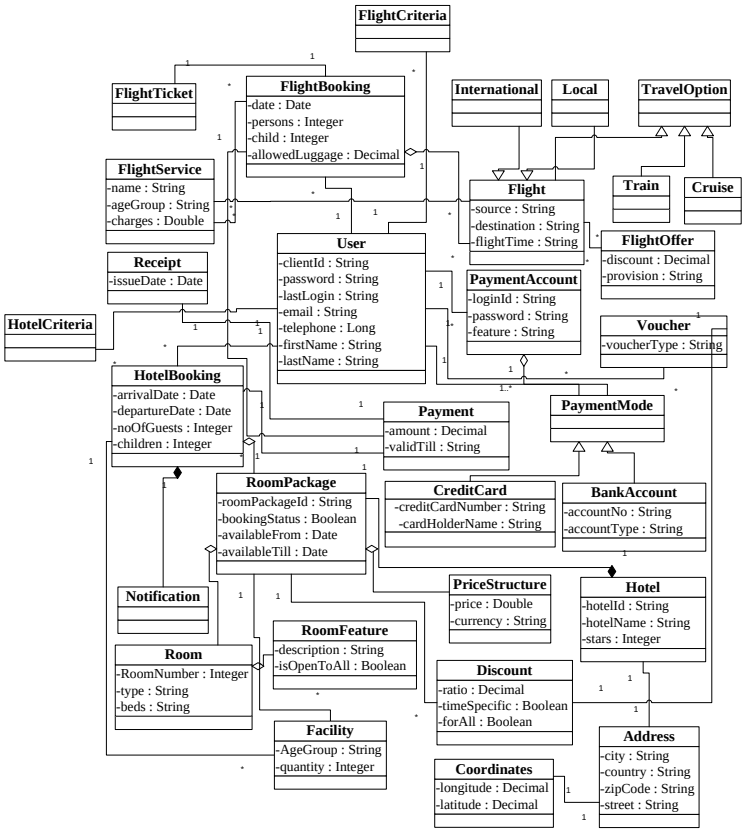

Figure 5.2: An Excerpt of the Global Data Model conforming to the Tourism DSO in the global ontology

5.3 Data Model Matching Algorithm

In this section, we will discuss our data model matching algorithm which is leveraged every time a service provider wants to publish his service offer on the service market or a service requester accesses the service market to search for a suitable service. In this case, the local data model of the service partner is matched to the the global data model on the basis of their annotations in the global ontology. The resulting local-global data model mappings are used to normalize the service request/offer under consideration. Fig. 5.3 gives an overview of the different steps of this algorithm. In the first step, all the attributes from the local data model are automatically annotated to concepts in the global ontology. On the basis of this annotation, a semantic matching is carried out between all the possible pairs of the attributes resulting in a similarity value in the next step. In the third step of this algorithm, possible attribute mappings are determined on the basis of the the calculated similarity values. In the next step, the local classes are also annotated in the global ontology in a similar fashion and a similarity value is calculated. For this purpose, the already determined attribute mappings are also considered. Lastly, on the basis of the resulting similarity values, possible class mappings are deduced.

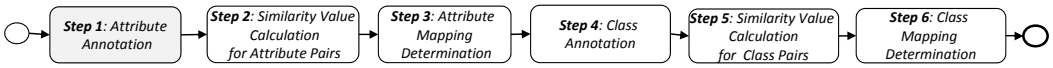

Figure 5.3: An Overview of the local-global Data Model Matching Algorithm

Below we will explain each of these steps in detail where the data model of the requester, i.e., HRS is matched to the global data model of the OTF provider shown in Fig. 5.2:

5.3.1 Attribute Annotation

The first step of the matching process is to automatically *annotate* the attributes in the local data model to the concepts in a DSO in the global ontology selected according to the domain of the service partner. This is done to exploit the hierarchy of the concepts in the ontology to enable the semantic matching of attributes. For this purpose, the identifiers (names) of the data model attributes are automatically matched to those of the ontology concepts. This lexical matching is not limited to an exact matching of the terms rather different lexical techniques are used [52], e.g., the names are tokenized, stopwords are eliminated, and finally the normalized names are

matched. As a result, the best possible match is selected to annotate an attribute in the local data model to a concept in the DSO in the global ontology. For a detailed technical account of this automatic annotation mechanism, we refer the reader to [140].

As mentioned earlier, we assume that the service partners define their data models using meaningful naming conventions and a DSO in the global ontology holistically covers the information in that given domain. Based on these assumptions, we claim that it is highly likely that for every local attribute a matching concept can be found in the global ontology. In a rare case, if the attribute name cannot be matched to any concept in the DSO, the service partner has to coordinate with the OTF provider and manually select a suitable concept in the DSO to annotate the local attribute. In this direction, the OTF provider can also decide to add some information in the DSO if possible under the given limitations for global ontology updation.

In addition to the annotations for the local attributes, the attributes in the global data model are already annotated in the global ontology. As already mentioned, the OTF provider defines its global data model conforming to its global ontology. This means that every element in the global data model is annotated to its *conforming* concept in the global ontology. This annotation of the global attributes is done once at the time of the creation of the global data model and is later used every time the data model matching algorithm is initiated.

For instance, in our running example, the global attributes like `first name`, `last name`, `longitude`, `latitude`, `price`, etc. are annotated to the concepts with matching names in the tourism DSO in the global ontology.

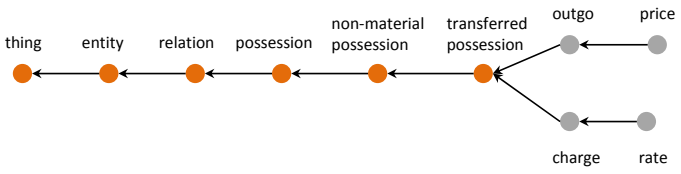

Figure 5.4: Annotation of the local and the global attribute `rate` and `price` in the Global Ontology

As a result of this annotation activity, Each attribute in the local and global data model is annotated in the global ontology.

An example for the annotation of a local and global attribute in shown in Fig. 5.4. In this case, the attribute `price` of the class `PriceStructure` in the global data model is annotated to its corresponding counterpart in the

global ontology. Similarly, the attribute `rate` of the class `RatePlan` in the local data model is automatically annotated to its matching concept in the DSO.

Once all the attributes in the local and global data models are annotated in the global ontology, their matching is performed in the next step.

5.3.2 Similarity Value Calculation for Attribute Pairs

After the annotation of the data model attributes, a similarity value is calculated for all the possible pairs each comprising a local and a global attribute. As specified in [52], the similarity between any two concepts can be specified in terms of a numerical value and the normalized similarity is a function from a pair of model elements to a number that ranges over the unit interval of real numbers [52]. According to [52], higher the similarity value between any two concepts, the more similar these concepts are.

Our matching mechanism calculates two types of similarity values between a pair of attributes: an upward cotopic similarity based on the global ontology and a type similarity value. These different similarity values are later aggregated into a single similarity value. Here, we will discuss these different types of similarity values briefly.

Definition 1 (Upward Cotopic Similarity)**.** *The upward cotopic similarity* $\sigma : o \times o \rightarrow [0, 1]$ *is a similarity over a hierarchy* $H = \langle o, \leq \rangle$*, such that:*

$$\sigma(c, c') = \frac{|UC(c, H) \cap UC(c', H)|}{|UC(c, H) \cup UC(c', H)|}$$

where $UC(c, H) = \{c' \mid \forall c, c' \in H \wedge c \leq c'\}$ *and* $c \leq c'$ *means that* c' *is more general than* c*.*

The upward cotopic similarity is one of the multiple similarity coefficients defined by [52] to assign a numerical value to the similarity between any two concepts. According to its definition specified in Def. 1, the upward cotopic similarity of any two concepts is the ratio of their shared hypernyms to the total number of their hypernyms in the taxonomy. For our mechanism, we calculate the upward cotopic similarity of two attributes on the basis of their annotated concepts in the global ontology. For instance, Fig. 5.4 shows the 10 hypernyms of the attribute pair `rate` and `price` in the global ontology, where the 6 shared hypernyms are also highlighted. Hence on the basis of its definition, `rate` and `price` have a upward cotopic similarity value of 6/10 = 0.6.

In addition to the upward cotopic similarity values, a primitive type similarity is also calculated for the attribute pair. For this purpose, a static look-up table is maintained to encode the similarity values for possible pairs of primitive types. According to this, if both the attributes are of numerical types, they have a higher type similarity as compared to the case where the attributes are of a numerical and an alphabetical type, respectively.

These two similarity values discussed here are aggregated into a single similarity value through the following equation [141]:

$$\hat{\delta} = \sum_{i=1}^{N} \delta_i \omega_i, \quad \sum_{i=1}^{N} \omega_i = 1, \quad \forall \omega_i : \omega_i > 0 \qquad (1)$$

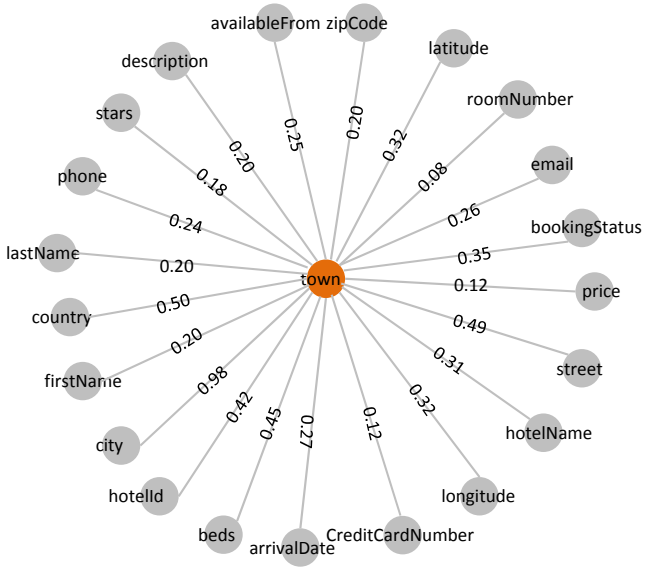

Figure 5.5: Similarity values of the local Attribute `city` with some of the global Attributes

This equation creates an aggregation of N different similarity values δ_i, where each is assigned a weight ω_i. The weights of different similarity values represent their significance in the similarity of the two attributes and can be adjusted according to the particular situation at hand. For instance, based on its claim of a comprehensive global ontology, the OTF provider claims that an upward cotopic similarity value for a local-global attribute pair has

higher probability of being a correct representation of their similarity and hence can be assigned a higher weight in the aggregated similarity value.

The equation given above also allows to increase the accuracy of matching mechanism in future by considering other similarity aspects mentioned in [52], e.g., similarity based on shared synonyms, similarity based on relations with other concepts, etc.

Fig. 5.5 shows some of the aggregated similarity values between an attribute, i.e., `town` in the local data model of HRS and the attributes of the global data model. These aggregated similarity values are calculated based on equation (1) and this figure shows that `town` has maximum similarity to `city`.

In the next step, attributes in the local and global data model are mapped on the basis of their aggregated similarity values.

5.3.3 Attribute Mappings Determination

After the calculation of similarity values for all possible attribute pairs, possible attribute mappings are determined on the basis of these similarity values for each pair.

Here it is important to mention that we assume that the attributes in the data models represent atomic information and hence we restrict ourselves to only $1:1$ attribute mappings in our approach so far. The aim of this step is to have most optimal attribute mappings so we consider this as an optimization problem where the goal is to determine as many attribute mappings as possible with maximum similarity values.

To determine such optimal attribute mappings with maximum similarity (or minimum dissimilarity), we use an existing algorithm minimum cost flow problem proposed in [152]. As an output of this algorithm, an excerpt of the local to global attribute mappings with

Local		Global
accomodation_id	0.85	hotelId
room_id	0.78	roomNumber
forename	1	firstName
family_name	1	lastName
roosts	0.4	beds
town	0.98	city
longitude	1	longitude
start_date	0.48	arrivalDate
email_address	0.85	email
rate	0.76	price
road	0.82	street
phone	1	telephone
finish_date	0.48	departureDate
latitude	1	latitude
country	1	country
due_amount	0.79	amount
availability_time	0.5	availableFrom
booked	0.34	bookingStatus
percentage	0.8	ratio
card_id	0.77	creditCardNumber
...		

Figure 5.6: Attribute Mappings with their Similarity Values

their respective similarity values is
shown in Fig.5.6.

After this automatic attribute mappings generation, class mappings are determined in the next step, which are based on these attribute mappings.

5.3.4 Class Annotation

Analogous to the attribute annotation in the Step 1, the classes in the local and the global data model are also annotated with the concepts from the global ontology. Similar to the global attributes, the global classes are already annotated in the global ontology. Additionally, the local classes are annotated in the global ontology based on the techniques used for Step 1.

5.3.5 Similarity Value Calculation for Class Pairs

Analogous to the similarity value calculation for the attribute pairs, a similarity value is also calculated for all the possible local and global class pairs. Based on equation 1, this similarity value is an aggregation of multiple similarity values.

Firstly, based on the mechanism explained in Step 2, an upward cotopic similarity is calculated between the class pairs based on their annotation in the global ontology.

In addition to the upward cotopic similarity of a class pair, their ratio of similar attributes is also considered while determining their overall similarity. For this purpose, the attribute mappings determined in Step 3 are taken into account. In this case, the argument is that the classes that share higher number of similar attributes have higher chances of being similar and vice versa. This similarity value is named as their *shared attribute similarity*. The shared attribute similarity value is based on relating the number of shared attributes to the total number of the attributes in a class pair. However, as the similarity between the shared attributes is not absolute rather represented through their similarity value, these similarity values of the shared attributes are also considered while the calculation of shared attribute similarity of two classes.

This is explained through an example shown in Fig. 5.7. This example shows the attribute mappings between the classes `RoomStay` and `RoomPackage` in the local and the global data models, respectively. So out of 7 attributes in total, the class pair has 3 shared attributes. Additionally, for these shared attributes, an attribute similarity value is calculated in Step 3.

Based on this information, the shared attribute similarity value for `RoomStay` and `RoomPackage` is calculated as $\delta = (0.6 + 0.34 + 0.36 + 3)/7 \approx 0.61$.

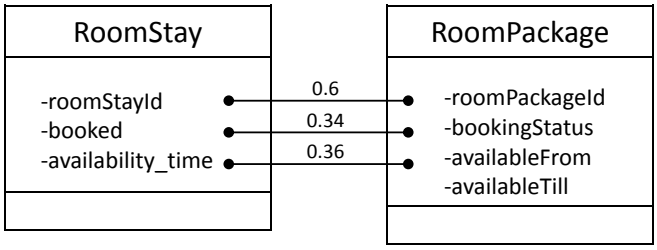

Figure 5.7: Attribute Mappings Between the local class `RoomStay` and the global class `RoomPackage`

On the basis of these different similarity values, an aggregated similarity value is calculated for every class pair based on equation (1). As mentioned earlier, there can be other similarity values that can be used in future to further strengthen the matching results, e.g., a similarity value based on the similar associations of the classes in a pair can also be considered in future.

The similarity values for class pairs are used to determine the possible class mappings in the next step of the algorithm.

5.3.6 Class Mappings Determination

In this step, possible class mappings are determined on the basis of the similarity values for the local-global class pairs calculated in Step 5. Unlike the *Attribute Mappings Determination* step where our approach aims at optimal attribute mappings leading to 1 : 1 attribute mappings, class mappings are mainly determined through a greedy strategy leading to complex class mappings.

For this purpose, the class mappings are created in a descending order gradually by traversing through class pairs, which are sorted on the basis of their similarity values in a descending order. At the same time,

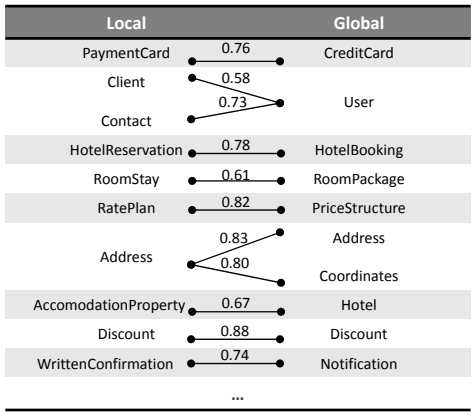

Figure 5.8: Class Mappings with their Similarity Values

a class pair is a candidate for a class mapping only if its similarity value is above a certain threshold, which is decided by the OTF provider. For a certain candidate class pair, if one of the classes is already a part of a class mapping, the other is also added to this class mapping. As a result of such a greedy strategy, disjoint 1 : 1, 1 : n, n : 1, and n : m class mappings can be determined.

Fig. 5.8 shows an excerpt of the class mappings for the running example that are determined through the proposed algorithm.

The output of this data model matching algorithm are the mappings between the elements in the local and the global data models, which are used as an input for the next phase, i.e., normalization of the VCs in the service descriptions.

5.4 Visual Contracts Normalization

Once the elements in the local data model of the service partner are mapped to those in the global data model, the proposed mechanism proceeds with the normalization of the service description (request/offer) of the service partners. As mentioned earlier, the service description normalization in our approach is mainly focused on the normalization of the requested and offered VCs as these behavioral semantics are mainly effected by the heterogeneity of the data models. Consequently, the normalization of VCs means that the service requests and offers conform to a common global data model and their data model heterogeneity is overcome.

In this context, we explain the normalization of the VCs here with the help of some examples.

In order to normalize a local VC, the elements of the local data model are replaced with their mapped counterparts from the global data model. Fig. 5.9 shows an example of such a normalization of the VC for the requested operation *makeReservation(...)*.

Starting with the normalization of objects in the VC, the objects typed over the local classes are replaced with their global counterparts, e.g., the objects of type `RoomStay`, `HotelReservation` are normalized to objects of type `RoomPackage`, `HotelBooking`, respectively. Analogously, for a 1 : n class mapping, the object of a local class is normalized to objects of the mapped global classes. Similarly, there is a n : 1 class mapping between the local classes `Contact` and `Client` and the global class `User`. Hence, the objects of type `Contact` and `Client` are replaced with an object of type `User`. A normalization on the basis of such an n : 1 class mapping has certain

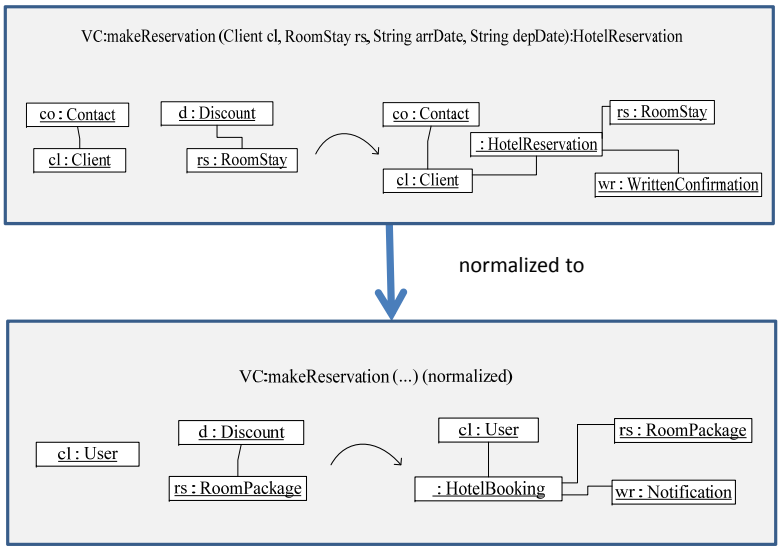

Figure 5.9: Visual Contract Normalization for the requested operation *makeReservation(...)*

restriction. In this case, the local VC must contain an object for every local class that is part of the mapping, otherwise the normalization is not possible. In the given example, if the VC does not contain an object of type `Contact` and only contains `Client` object, then the normalization to global class `User` is not possible. Later in the discussion, we will also discuss some other cases where such a situation can arise that the normalization cannot be carried out correctly and what can be possibly done in such cases.

After the normalization of the objects, the links between these objects need to be normalized. In our data model matching algorithm, we do not explicitly match the associations in the local and global data model. Instead, the links in the local data model are normalized exclusively on the basis of the class mappings. For instance, in the given example, the link between the objects of type `RoomStay` and `HotelReservation` can be normalized to a link between the objects of type `RoomPackage` and `HotelBooking` because the association between the local classes is implicitly mapped to the association between their mapped global classes. Similarly, the link between `Client` and `HotelReservation` is normalized to a link between `User` and `HotelBooking`. This means that the links between objects in the local VC are normalized to links between mapped objects in the normalized VC conforming to the

associations in the global data model.

At present, our normalization mechanism is restricted to the normalization of objects and links in the VCs and it does not consider the attributes normalization so far. This restriction is applied because our service matching mechanism introduced in the later phases of our approach is only concerned with the matching of objects and links in the VCs and is does not deal with attribute matching so far. Hence, we do not go into further details of the attribute normalization here and would recommend the reader to [140] for an insight into different methods, aspects and complications of attribute normalization.

As mentioned earlier, during this normalization of VCs, there is also a possibility that some information is lost or the normalization of some VCs is not possible at all. For instance, if certain elements in the local data model are not mapped to any element in the global data model, then these elements cannot be translated to any global elements during VC normalization. However, we argue that based on our assumption about the global data model comprehensively defining the structure of information in a certain domain, such a situation where some local elements cannot be mapped to the global elements is less likely to occur.

On the other hand, the difference of granularity levels between the local and the global data model can lead to situations where the VC normalization can be problematic. For instance, we mentioned earlier that in the given example, if the VC contains object of type `Client` but `Contact` object is not specified following the loose semantics rule, the VC cannot be normalized. This is because `Client` and `Contact` have a $n : 1$ mapping with `User` and the translation can only take place if objects for all the local elements in the mapping are contained in the VC. Another example which leads to a more complex situation is that if the VC in Fig. 5.9 additionally specifies that the object of type `Contact` and the link between `Client` and `Contact` object are deleted as a post-condition. In such a case, the VC cannot be normalized because the deletion of these objects cannot be captured in the normalized VC due to the $n : 1$ mapping mentioned earlier.

In such situations where certain VCs cannot be normalized, the service partner is prompted to make any suitable modifications to his service description if possible. Such modifications can be straightforward for the cases similar to the first variant mentioned above where the `Contact` object can be specified in the VC to allow its normalization.

For the work in this thesis, we assume that such an exceptional situation does not arise often where a successful normalization of a service description is not possible. Otherwise, a further processing of the service description in

not possible in the given approach.

However, here we also briefly introduce a possible option to deal with such a situation, which needs to be investigated and analyzed in detail in future. If it is not possible for the service partner to make required modifications, the OTF provider can decide to allow its further processing without being normalized. In this exceptional case, the service partner cannot fully utilize the benefits of the proposed approach though.

In case of the requester, such an unmodified request is indeed matched to the available service offers by the local data model matching approach described in Sec. 5.2.1. In this direction, the requester data model is matched to the data model of each available service offer and the resulting mappings are used to normalize the request accordingly. Consequently, the service discovery and composition process can be quite time-consuming in this scenario and the results may not be guaranteed to be completely correct.

In case of the provider, such an *unnormalized* offer is published on the service market by adding it to a specific pool of unnormalized service offers in the repository. These offers are not considered during a normal course of the service discovery and composition process. Rather, these offers are taken into consideration only if the requester is not satisfied with the service discovery and composition results produced by through the proposed approach. In this case, the requester compromises on the late response time and explicitly selects to also consider these pooled offers during his service discovery process. Subsequently, the service discovery and composition process may be re-initiated to also consider the unnormalized pooled service offers.

The feasibility of such a technique to deal with the unnormalized service descriptions needs to be further analyzed in future.

After the successful normalization of the service request/offer at hand, the service discovery mechanism can start in the next phase.

5.5 Summary and Discussion

This chapter encompasses the first phase of our service discovery and composition approach, i.e., service description normalization. An automatic service discovery and composition can be adversely effected due to the data model heterogeneity of the service request and the available offers. To resolve this issue, our approach allows a normalization of the service requests/offers, i.e., their translation to a common representation and hence enabling their actual matching. For this purpose, whenever a service requester/provider

accesses the service market with its service request/offer, its local data model is matched to a global data model conforming to a global ontology maintained by the OTF provider. This global ontology provides a standardized and comprehensive account of the information in a particular domain. Our local-global data model matching algorithm allows the data models matching based on their ontological semantics defined through the global ontology. The service request/offer under consideration is normalized based on the resulting local-global data model mappings. In this direction, the constituting VCs, which are typed over the local data model so far are translated to a normalized VC which is typed over the global data model. The resulting normalized service description can later by considered for the next phase, i.e., service discovery phase of the proposed approach.

So far, our data model mechanism does not takes the particular case into account, where the independent domain knowledge of the service partners is defined as their local ontology and the local data model conforms to this local ontology in turn. In future, it needs to be further investigated how the information in such a local ontology can be used to further improve the data model matching results.

So far, the data model matching mechanism mainly relies on the hierarchical structure of the ontology for the semantic matching of the concepts. However, this mechanism can be extended in future to consider other information contained by an ontology as well, e.g., concept associations, axioms, synonyms, homonyms, etc.

Additionally, during the course of our research, we deduced that the problems encountered during VC normalization step as discussed in Sec. 5.4 mostly occur if the local data model is more fine-grained as compared to the global data model. This means that the chances of the occurrence of these problems are higher in case of n : 1 or n : m model mappings. This is because there are chances of information loss during the translation of local elements to their coarse-grained global counterparts. On the other hand, 1 : 1 and 1 : n mappings between the elements in the local and the global data model normally lead to seamless VC normalization. However, this claim needs to be interrogated further to come up with better solutions to deal with cases where the service discovery process cannot proceed due to the issues in the service description normalization phase.

6 Multi-level Service Discovery

After the service request normalization, next phase in the proposed approach is service discovery. As input, it takes a *normalized* service request and matches it to the set of available service offers (also normalized) using a multi-level approach. This approach considers different factors, e.g., the category assigned to the request and the offer, the structure as well as the behavior of the requested and offered operations, etc. The output of this phase is the *Requester Operation Mapping*, which is the set of possible operation correspondences between the request and the available offers. A detailed overview of this phase is given in Fig. 6.1, which will be discussed in detail in the following section.

6.1 Service Discovery Overview

When a service requester accesses the OTF provider to initiate the service discovery and composition process, the actual matching of the requested and offered service descriptions starts from the service discovery phase. This phase consists of multiple levels to match the service request under consideration to the service offers available on the market, where the result set of discovered service offers is gradually reduced after each level.

The first level of the service discovery phase is the *category matching*, where the service description matching is on the basis of their categorization. As a result, a subset of the service offers are selected. The concept of *category matching* is not new in the context of service discovery. Service registries [103, 129] allow the publishing and discovery of offered services on the basis of their categorization. For instance, UDDI [103] allows the categorization of published web services by using standard taxonomies like North American Industry Classification System (NAICS) [155]. Similarly, [4] proposed a categorization ontology development approach for the service registry *programmableWeb* [129], which allows to automatically develop and

maintain an ontology enabling semantic-based publishing and discovery of services in their respective categories.

Similarly, OTF computing also allows a categorization of service partners broadly based on the information domains that OTF provider caters to. However, this categorization mechanism and category matching of the service descriptions is not in focus for our work in this thesis and we briefly introduce it in Sec. 6.2.

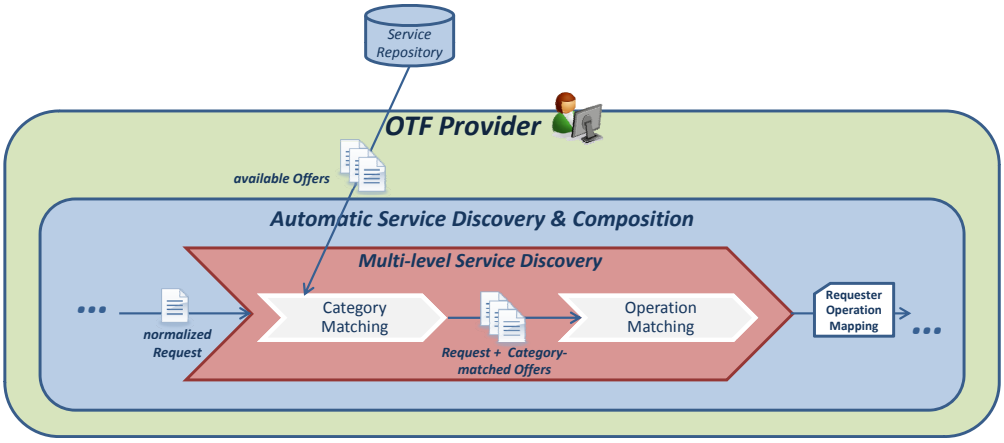

Figure 6.1: An Overview of the Service Discovery Phase in the Proposed Approach

After category matching, the service descriptions have to be matched on the basis of the requested and offered functionality. In this context, the basic unit of functionality are the requested and offered operations in the matched service descriptions. Hence, an *operation matching* is performed for the request and the selected service offers on the next level.

As mentioned earlier, the current standard for service descriptions, i.e., WSDL [162] allows to specify a service description in terms of the structural aspects of the operations, i.e., operation signatures. Therefore, a lot of research effort has been directed towards devising mechanisms for automatic service discovery based on operation signature matching. Approaches like [87, 147, 150] have devised automatic mechanisms to match the requested and offered operations on the basis of their structural elements, such as, operation names, input and output parameters.

Although these earlier approaches automatically match the WSDL-based service requests and offers quite elaborately, they do not ensure accuracy in the service discovery results. As we claimed in Chap. 1 and 2, an accurate

service discovery is only possible if in addition to the structural aspects, the behavioral aspects of the service requests and offers are also specified and matched. This means that an operation matching mechanism should not only consider the structural elements of the operations, rather its should also take their behavioral semantics into account while matching. Additionally, according to the OTF computing vision and the essence of SOC, the service partners describe their requested and offered functionality independently leading to *granularity level heterogeneity* among their service descriptions. Considering such heterogeneous granularity levels of a service request and offer, an operation matching mechanism should not be restricted to 1 : 1 operation matching rather it should also allow to determine, i.e., 1 : n , n : 1 and n : m correspondences among the requested and offered operations. In a comprehensive service description, the service protocol is meant to describe the required/allowed sequence in which the requested/offered operations can be invoked. This information in the service protocols can be exploited while determining the complex operation correspondences to further improve the accuracy of operation matching. Based on this discussion, we define the requirements for an elaborate operation matching mechanism, which are as follows:

R1 It must allow the matching of the requested and offered operations on the basis of their structural as well as behavioral description.

R2 In order to overcome the granularity level heterogeneity between the requested and offered operations, it must be able to determine complex correspondences between them while considering their requested/allowed invocation sequences in their respective service protocols.

From our detailed study and evaluation of the service matching approaches presented in Chap. 2, it can be deduced that a wide range of approaches, e.g., [65, 108, 84, 157, 25, 14, 15] acknowledges the need of a comprehensive matching of service descriptions and hence also allow to match behavioral semantics in addition to the structural elements of the operations. However, most of these approaches consider *atomic* service requests and offers comprising of a single requested/offered operation enabling a 1 : 1 operation matching between them. Some approaches [88, 14, 108, 79] allow complex correspondences in a limited sense as they enable a 1 : n matching among atomic service descriptions. But due to atomicity of matched service descriptions, the invocation sequence is not relevant in these approaches. Some others [107, 35, 59] lay particular focus on the determination of 1 : n

and n : 1 operation correspondences on the basis of the invocation sequence. However in these approaches, the matching is only based on the structural elements of the operations.

Considering these shortcomings of the existing approaches, we proposed an elaborate operation matching mechanism, which fulfills the requirements specified earlier.

The output of the service discovery phase is the set of all possible operation correspondences between the requester and the selected service offers. In the following sections, we will give details of the different levels of the service discovery phase in our approach.

6.2 Category Matching

As mentioned earlier, the creation and maintenance of a categorization hierarchy as well as the categorization and discovery of published services on its basis has been a topic for extensive research in recent years [4, 16, 122]. Similarly, OTF Computing also proposed the categorization of the service requests and offers on the basis of a categorization hierarchy created and maintained by the OTF provider.

According to this approach, in order to publish its service offer on the service market, the service provider manually categorizes its service offer using the categorization hierarchy of the OTF Provider. Similarly, the service requester also has to categorize its service request to discover the service offers satisfying its request. During the service discovery phase, the categorization of service request and the available service offers are matched and a subset of service offers is selected as a result.

As the development of such a categorization hierarchy is not the focus of the research in this thesis, we assume that the OTF Provider defines a sophisticated mechanism for its creation, maintenance, and category matching of the service descriptions. For this purpose, it may also reuse the existing approaches [155, 4, 16, 122]. Here, we give a brief idea of how such a categorization hierarchy can look like.

Fig. 6.2 visually depicts an excerpt of such a categorization hierarchy of the OTF Provider and also the categorization of the service offers and requests. This hierarchy is broadly based on the information domains that the OTF provider caters to. Each of these domains is then further divided into the sub-categories belonging to this domain. Each service request and offer can be assigned the categories from the particular domain that they belong to. For the running example, the request and the service offers belong

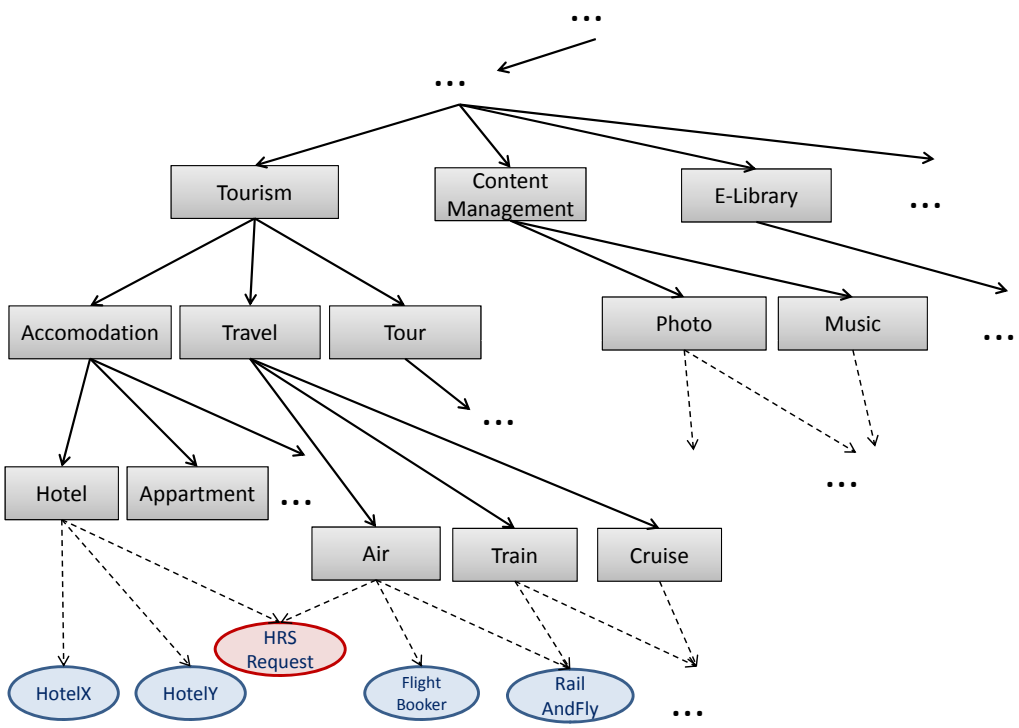

Figure 6.2: An Excerpt of Service Categorization Ontology maintained by OTF Provider

to the broader category `Tourism`. One service description can be assigned to one or more categories in its domain, e.g., *RailAndFly* is a service to search and book flight and train connections. Hence, it is assigned to the subcategories `Air` and `Train` in the `Travel` category. Similarly, HRS request is also categorized under different sub categories of the `Tourism` category.

Based on this categorization of service request and offers, a subset of available service offers can be selected through category matching that can be further processed during operation matching.

6.3 Operation Matching

To fulfill the requirements for operation matching specified in Sec. 6.1, we have come up with an *operation matcher* for RSDL service requests and offers, which will be explained in detail in this section.

As shown in Fig. 6.1, the request under consideration and the subset of

available offers selected after category matching act as inputs for the operation matching mechanism. The proposed mechanism allows an elaborate matching of the operations in the request and the offers on the basis of their structural and behavioral aspects specified in RSDL (satisfying R1). Additionally, it acknowledges the difference of granularity level heterogeneity between the service partners and determines complex operation correspondences between the requested and offered operations while considering their invocation sequence in the respective RSDL service protocols (satisfying R2).

For a comprehensive operation matching in terms of their structural and behavioral aspects, our mechanism mainly focus on the matching of behavioral semantics specified as visual contracts for the operations while considering their invocation sequence specified in the service protocol. For this purpose, we define a set of operation matching strategies that allow to determine complex operation correspondences by matching their VCs. In this direction, we claim that an explicit matching of operation signatures is not required for RSDL service descriptions. In an RSDL service description, an operation's signature is in conjunction with its behavioral semantics. As explained in Sec. 3.2.2, RSDL highly recommends to use non-primitive data types, i.e., data model classes to type the input and output parameters in the operation signatures. Additionally, it applies the restriction that all such parameters with non-primitive data types in an operation signatures are bound to appear as objects in the corresponding VC of that operation. Considering this coherence between an operation's signature and its VC, it makes senses to claim that the input/output parameters are automatically matched while the matching of their VCs and hence do not need to be matched separately. However, there can be situations where it is inevitable for the service partners to define input/output parameters with primitive types, e.g., for the requested operation `makeReservation(...)`, two input parameters of type `Date` are necessary to specify the arrival and departure date. To match such primitive parameters in the operation signatures, our mechanism allows their explicit matching, which will be explained later in more detail.

It is also important to mention here that our operation matching mechanism based on VC matching focus particular aspects of the requested and offered VCs so far. A VC mainly describes the behavior of an operation in terms of the structural changes, such as, the deletion, creation and preservation of the objects and the links after its execution. This description can be further refined in a stepwise manner by also incorporating the attributes and their modification before and after the execution of the operation. Fur-

thermore, negative application condition (NAC) and positive application condition (PAC) can further constrain and precisely define the behavior of the particular operation.

So far, our matching mechanism is concerned with the structural changes in the requested and offered VCs and do not focus further elements in these VCs, e.g., the attributes, NACs and PACs. In future, the accuracy of these matching results can be further improved and *false positives* can be avoided in the result set by extending the mechanism to also match these additional elements in the requested and offered VCs.

Before going into the details of the proposed operation matching strategies, we explain some core concepts that are extensively used.

Element Correspondence in Visual Contracts: In order to match the VCs of the requested and offered operations, the operation matching mechanism define certain criteria to match the respective preconditions and postconditions. As the matching of preconditions and postconditions in the requested and offered VC depends on matching their comprising elements, i.e., objects and their links, there is the requirement to define the notion of *element correspondence*.

For the proposed approach, the notion of *element correspondence* between the VC elements is mainly based on the similarity of their *types*. In this context, it is important to mention that the requested and offered VCs are already normalized during operation matching phase and hence the elements in these VCs are typed over the global data model of the OTF provider.

According to [123], the similarity of two concepts can be categorized based on their degree of similarity as `exact`, `plugin`, `subsume` and `fail` by matching their underlying ontological semantics. In our case, this also holds valid for the types of VC elements. For instance, two elements of type `CreditCard` have exact type similarity, whereas an element of type `CreditCard` has a plugin type similarity to an element of type `PaymentMode` based on the specialization-generalization relationship of the types. On the contrary, the element of type `PaymentMode` has a subsume type similarity to the element of type `CreditCard`. If none of these holds true for any two elements, then they have a failed type similarity.

These different kinds of type similarity can be can be ordered according to their strength. Here, it suffices to say that an exact similarity between two types depicts the most strong similarity. On the contrary, a plugin and subsume type similarity have a lesser degree of strength. Whereas, `fail type similarity` is the least strong. For instance, for an element of type

`CreditCard`, is strongly type similar to another element of type `CreditCard` as compared to an element of type `PaymentMode`.

The weaker types of similarities, i.e., plugin and subsume are more relevant for a matching approach where a fuzzy matching between the service request and offer is considered, i.e., the case where an offer does not completely satisfy the requirements rather it *approximately* matches the service request. In such an approach, such weaker similarity between the types of requested and offered elements can be considered to determine varying degree of matching between the request and offer. However, in our service matching approach, we so far do not consider an approximate matching between the service descriptions and hence do not consider weaker type similarity between the elements. Additionally, the *plugin* and *subsume type similarity* between the VC elements can lead to complex 1 : n and n : 1 correspondences between the elements in the matched VCs adding further complexity to the operation matching process. Keeping these considerations in mind, our operation matching mechanism restricts itself to an *exact type similarity* between the corresponding elements in the matched VCs so far.

This means that for two elements in the requested and offered VCs to be corresponding, their has to be an *exact similarity* between their types. A formal definition of such a correspondence between the elements in the requested and offered VCs is presented later, when we present the notion of matching of VCs for operation matching.

Apart from element correspondence, another important concept for our operation matching mechanism is the requester operation mapping based on the operation correspondences between the requested and offered operations. This is discussed in the next section in detail.

Requester Operation Mapping: The output of the operation matching mechanism is the set of all possible operation correspondences for the request under consideration. As mentioned earlier, an operation correspondence can be of type 1 : 1, 1 : n, n : 1, or n : m. We already discussed that for the complex correspondences, their invocation sequence in the respective service protocols has to be taken into account. For instance, for a 1 : n operation correspondence between the requested and the offered operations, n stands for a sequence of offered operations that can be invoked in the given order in the offered service protocol. Below, we give a formal definition for such an operation sequence and different types of operation correspondences that can be determined between a request and an offer.

Definition 2 (Operation Sequence)**.** *Given an RSDL service description* desc *with its respective service protocol* prot *and some of its operations*

$op_i, \cdots, op_j$, $op_i \rightarrow \cdots \rightarrow op_j$ represents an operation sequence, if these operations can be invoked in the given order in prot. □

Definition 3 (Operation Correspondence)**.** *Given are an RSDL request r and an RSDL offer o. Further, given are a requested operation op_r and a requested operation sequence seq_r in r and an offered operation op_o and an offered operation sequence seq_o in o. (seq_r,seq_o) represents an operation correspondence between r and o and can have one of the following types:*

- *A **1 : 1 operation correspondence**, where seq_r and seq_o comprise of single operations op_r and op_o, respectively and op_r matches to op_o through operation matching.*
- *A **1 : n operation correspondence**, where seq_r comprise of single operation op_r, seq_o comprise of multiple operations and op_r matches to seq_o through operation matching.*
- *A **n : 1 operation correspondence**, where seq_r comprise of multiple operations whereas seq_o comprise of single operation op_o and seq_r matches to op_o through operation matching.*
- *A **n : m operation correspondence**, where seq_r and seq_o comprise of multiple operations and seq_r matches to seq_o through operation matching.* □

The notion of operation matching is defined in detail in the following. The aim of the operation matching is to determine the set of all possible operation correspondences for the request under consideration. We term this set of requester's operation correspondences as its *operation mapping*. All the offers that participate in the *requester operation mapping* constitute the set of service offers that are considered for further matching during the service composition phase. A formal definition for a *requester operation mapping* is given in Def. 4.

Definition 4 (Requester Operation Mapping)**.** *For a RSDL request r, the requester operation mapping $OpMap_r$ is the set of all possible* operation correspondences *with the available service offers.* □

In the next sections, we describe the set of proposed operation matching strategies that constitute our operation matching mechanism to determine different types of operation correspondences. In a bottom up approach, we also devise a mechanism that use these operation matching strategies to determine *operation mapping* for a requester with the available offers. This mechanism will also be explained later.

6.3.1 1 : 1 Operation Matching

In this section, we elaborate our strategy for 1 : 1 operation matching, which results in a 1 : 1 operation correspondence.

Fig. 6.3 shows a diagrammatic representation of a 1 : 1 operation correspondence resulting from a matching between the requested operation op_r^i and the offered operation op_o^j. A requested operation matches an offered operation if the offered operation is invocable and as a result, it satisfies all the requirements specified by the requested operation. As we discussed earlier, our operation matching mechanism is mainly concerned with the matching of the behavioral semantics, i.e., VCs of the requested and offered operations. Hence, op_r^i has a 1 : 1 operation correspondence with op_o^j if their VCs match.

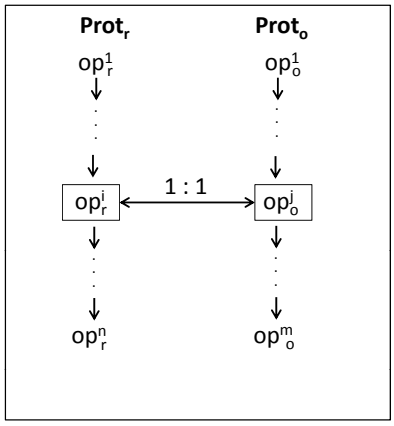

Figure 6.3: A diagrammatic Representation of a 1 : 1 Operation correspondence

In the next section where we formally define the structure of a 1 : 1 operation correspondence and later describe how their VCs can be matched to determine such a correspondence.

Definition for 1 : 1 Operation Correspondence

The definition for a 1 : 1 operation correspondence is proposed by [65, 96], which allows a matching between *atomic* service requests and offers by matching their respective visual contracts.

It is important to mention here that for a VC as graph transformation rule, the elements in the left- and right-hand side can be differentiated as

nodes and links in the graph. However, for the work presented in [96], these elements do not exhibit different behavior and hence are not required to be differentiated into these two different types. As a result, they are rather considered on a general level only as *elements* of the VC and the left- and right-hand side of a VC are represented as a set of VC elements, respectively. In this direction, the set operators, such as, union, intersection, etc. are also applicable on these sets.

In this direction, the VC of the requested operation as a graph transformation rule is represented as $op_r = L_r \rightarrow R_r$. Its left-hand side declares the preconditions, i.e., the elements that the requested operation guarantees to provide. As a result, it requires that its postconditions declared on right-hand side are satisfied, i.e., elements in $L_r \setminus R_r$ are meant to be deleted, elements in $R_r \setminus L_r$ are meant to be added, and the elements $L_r \cap R_r$ are meant to be preserved, if they are present. On the other hand, the VC of an offered operation is represented as $op_o = L_o \rightarrow R_o$. Its left-hand side declares the preconditions that the offered operations requires to exist in order to make its invocation possible. As a result of its invocation, it guarantees that it satisfies the postconditions declared on the right-hand side, i.e., elements in $L_o \setminus R_o$, $R_o \setminus L_o$, and $L_o \cap R_o$ are deleted, added, and preserved, respectively.

To determine a $1:1$ correspondence between the requested and offered operation, these requirements and guarantees declared in their respective VCs have to be compared. A formal definition for $1:1$ operation correspondence is given in [65, 96] as follows:

Definition 5 ($1:1$ Operation Correspondence)**.** *Given the following:*

- *A request r;*
- *An offer o;*
- *A requested operation op_r^i in r with visual contract $L_r^i \rightarrow R_r^i$;*
- *An offered operation op_o^j in o with visual contract $L_o^j \rightarrow R_o^j$;*

The pair (op_r^i, op_o^j) is termed as a $1:1$ correspondence, if there exists an element correspondence ***corr*** *(according to Def. 6) with the following properties:*

P1: $\forall y \in L_o^j \; \exists x \in L_r^i : x \; \textbf{\textit{corr}} \; y$*, i.e., for every element in L_o^j there is a corresponding element in L_r^i;*

P2: $\forall x \in L_r^i \setminus R_r^i \; \exists y \in L_o^j \setminus R_o^j$: $x \; \boldsymbol{corr} \; y$, *i.e., for every element in* L_r^i *that is meant to be deleted by* op_r^i *corresponds to an element that is indeed deleted by* op_o^j*;*

P3: $\forall x \in L_r^i \cap R_r^i \; \exists y \in L_o^j \cup R_o^j$: $x \; \boldsymbol{corr} \; y \Rightarrow y \in L_o^j \cap R_o^j$, *i.e., every element in* L_o^j *corresponding to an element that is meant to be preserved by operations in* op_r^i *must indeed be preserved by* op_o^j*;*

P4: $\forall x \in R_r^i \setminus L_r^i \; \exists y \in R_o^j \setminus L_o^j$: $x \; \boldsymbol{corr} \; y$, *i.e., for every element in* R_r^i *that is meant to be added by* op_r^i *corresponds to an element that is added by* op_o^j*;* □

Definition 6 (Element Correspondence)**.** *Given the following:*

- *A request* r *with operations* $op_r^1, \cdots, op_r^n$ *and their respective visual contracts* $L_r^1 \rightarrow R_r^1 \; , \cdots, L_r^n \rightarrow R_r^n$*;*
- *An offer* o *with operations* $op_o^1, \cdots, op_o^m$ *and their respective visual contracts* $L_o^1 \rightarrow R_o^1 \; , \cdots, L_o^m \rightarrow R_o^m$*;*

corr *is a binary relation over the sets* $L_r^1 \cup R_r^1 \cup \cdots \cup L_r^n \cup R_r^n$ *and* $L_o^1 \cup R_o^1 \cup \cdots \cup L_o^m \cup R_o^m$ *with the following properties:*

1. ***type-compatible:*** *This means that the corresponding objects have an exact type similarity and corresponding links have corresponding source and target objects.*
2. ***Partial:*** *This means that not all the elements in the sets* $L_r^1 \cup R_r^1 \cup \cdots \cup L_r^n \cup R_r^n$ *and* $L_o^1 \cup R_o^1 \cup \cdots \cup L_o^m \cup R_o^m$ *have to occur in the relation* ***corr****.*
3. ***One-to-One:*** *This means that every element can occur at most once in the relation* ***corr*** *and hence has a correspondence with only one element.* □

According to this Def. 5, the requested operation op_r^i matches an offered operation op_o^j, if the offered operation is invocable, i.e., for all elements in its preconditions, there are corresponding elements in preconditions of the requested operation.

Additionally, all the postconditions of the requested operation are satisfied by the offered operation. This means that all the elements added and deleted by the requested operation have corresponding elements added and deleted by the offered operation, respectively. Similarly, all the preserved

elements of the offered operation that correspond to some preserved element in the requested operation are also preserved in the offered operation.

In this case, the corresponding elements are determined on the basis of Def. 6, which formalizes our notion of element correspondences between VC elements discussed earlier in Sec. 6.3. This formal definition is also based on the *element correspondence* specified in [96, 65]. For our approach, as the request and the offer comprise of multiple operations with their respective VCs, the element correspondence spans over the elements of all the VCs contained in the request and the offer. According to this definition, the correspondence between objects in the requested and offered VCs is based on their exact type similarity for objects and correspondence of links is based on the correspondence between their source and target objects. Additionally, it is not necessary that every VC element in the request corresponds to a VC element in the offer and vice versa, e.g., some elements in preconditions of a requested operation may not be required by any offered operation for its invocation. In this case, these VC elements of request do not have corresponding VC elements in the offer. Lastly, this element correspondence has to be $1:1$. This means that any VC element in the request can correspond to at most one VC element in the offer and vice versa.

This structure of $1:1$ operation correspondence is explained through the examples in next section, where we present an algorithm to determine such a $1:1$ operation correspondence between the request under consideration and an offer.

1 : 1 Operation Matching Algorithm

Apart from the definition of the formalized structure of different kind of operation correspondences, it is also important for our operation matching mechanism to define a strategy for matching the requested operations with the offered operations resulting in these different types of operation correspondences.

Hence, we define a $1:1$ operation matching algorithm which matches a requested operation under consideration with the offered operations in an offer and determine any possible $1:1$ operation correspondence.

This algorithm is specified in Listing 1. As input, it takes a requested operation op_r, and the set of offered operations Ops_o. As output, it returns any possible $Corr$, i.e., a $1:1$ correspondence between op_r and op_o satisfying the properties specified in Def. 5. For example, as shown in Fig. 6.4, our operation matching algorithm determines a *1 : 1 operation correspondence*

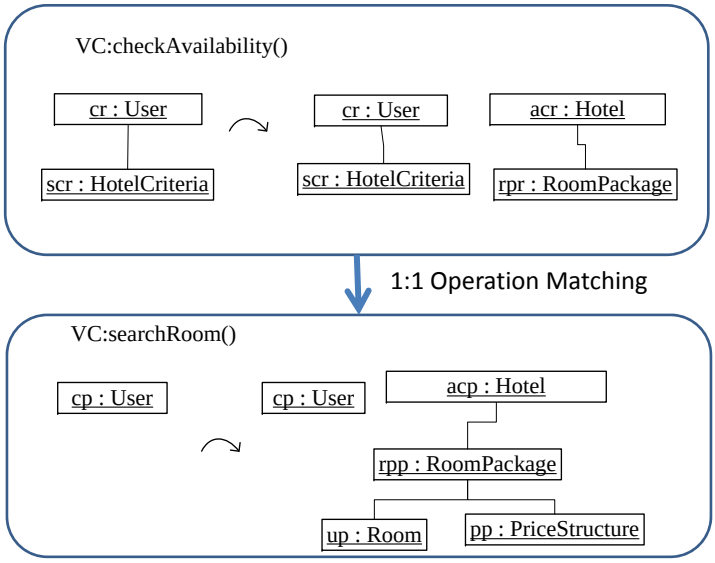

Figure 6.4: 1 : 1 Operation correspondence Example

between *checkAvailability(...)* of *HRS* and *searchRoom(...)* by *HotelX*.

The algorithm in Listing 1 works as follows:

Step ① In order to find a corresponding operation for op_r, the algorithm matches it with every offered operation in Ops_o. In the given example, to find a possible 1 : 1 operation correspondence for the requested operation *checkAvailability()* of *HRS*, the first offered operation in the set of offered operations by *HotelX* considered for further matching is *searchRoom()*.

Step ② For op_r, $Pre(op_r)$ and $Post(op_r)$ represent the preconditions that it guarantees to provide and the postconditions that it requires to be satisfied in terms of deletion, addition and preservation of elements, respectively. Similarly, For op_o, $Pre(op_o)$ and $Post(op_o)$ represent the preconditions that it requires in order to be invoked and the postconditions that it guarantees to satisfy, respectively. In the second step, it is checked whether the requested operation satisfies all the preconditions required for the invocation of the offered operation. This is determined through the function $\texttt{satisfies}_{\texttt{pre}}(Pre(op_r), Pre(op_o))$ according to `P1` in Def. 5. In the given example,

Listing 1: Algorithm for 1 : 1 operation matching between a requester operation o_r and an offered operation in Ops_o

```
Input: Requested Operation op_r
Input: Set of Offered Operations Ops_o
Output: A resulting operation Correspondence Corr

OneToOneMatching(op_r, Ops_o)
    Corr = null;
    for next unmatched op_o in Ops_o do                          // Step ①
        if satisfies_pr(Pre(op_r) , Pre(op_o)) then              // Step ②
            if compSatisfies_po(Post(op_o) , Post(op_r)) then
                Corr = create1To1Corr(op_r , op_o);              // Step ③
                Exit;
            end
        end
    end
    return Corr;                                                 // Step ④
end
```

searchRoom() requires a `cp:User` object in its preconditions in order to be invoked. According to Def. 6 for element correspondence, *checkAvailability()* has a corresponding element `cr:User` in its preconditions. Hence, $\texttt{satisfies}_{\texttt{pre}}(Pre(checkAvailability()) , Pre(searchRoom()))$ holds true. Additionally, it is important to note that the requested operation may provide more information than required to invoke an offered operation, i.e., there are more elements in the preconditions of the requested VC as compared to those in the offered VC. For instance, in the given example, the requester additionally provides `scr:HotelCriteria` and a link between `cp:User` and `scr:HotelCriteria` but the offered operation *searchRoom()* does not require these elements.

If the preconditions of the offered operation are satisfied, then it is checked whether the offered operation under consideration *completely satisfies* the postconditions of the requested operation according to `P2`, `P3`, `P4` in Def. 5. This is determined through $\texttt{compSatisfies}_{\texttt{po}}(Post(op_o) , Post(op_r))$, which holds `true` in case the requested postconditions are completely satisfied.

For the given example, all the elements created by *checkAvailability()*, i.e., `acr:Hotel`, `rpr:RoomPackage`, and a link between these two ob-

jects have corresponding elements created by *searchRoom()*. Similarly, *searchRoom()* indeed preserves the element `cp:User` that correspond to an element `cr:User` preserved by *checkAvailability()*. As there is no deleted element in the requested operation, so `P2` in Def. 5 is trivial in this case. Additionally, the offered operation can have additional postconditions to those required by the requested operation. For instance, in addition to the searched *RoomPackage*, *searchRoom()* also provides additional information about the *Room* and the *PriceStructure* of that *RoomPackage*. These postconditions however are not relevant for the requester. From this matching of the requested and offered postconditions, it can be deduced that $\texttt{compSatisfies}_{\texttt{po}}(Post(searchRoom())$, $Post(checkAvailability()))$ holds true.

On the basis of this matching, an element correspondence *corr* indeed exists, which satisfies the properties `P1`, `P2`, `P3`, and `P4` in Def. 5.

Step ③ If the offered operation under consideration is a valid match for the requested operation, then a 1 : 1 operation correspondence is created between the requested and the offered operation through `create1To1Corr()`. in the given example, *checkAvailability()* has a 1 : 1 correspondence with *searchRoom()*. At this point, further offered operations do not need to be considered and hence the further processing is stopped.

Step ④ At the end, the determined 1 : 1 operation correspondence $Corr$ is returned. It is empty, if no offered operation matches the requested operation under consideration.

Through this algorithm, the operation matching mechanism can determine any possible 1 : 1 operation correspondence for a requested operation under consideration in an offer. In the next section, we discuss the n : 1 operation correspondence and our operation matching algorithm to determine such n : 1 operation correspondences between the request and the offer.

6.3.2 n : 1 Operation Matching

A n : 1 operation correspondence exists between a requested operation sequence and an offered operation, when the offered operation can be invoked as its preconditions are satisfied and it satisfies the requirements specified in the requested operation sequence. Fig. 6.5 shows a diagrammatic representation of such a n : 1 operation correspondence between the requested operation sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$ and the offered operation op_o^j.

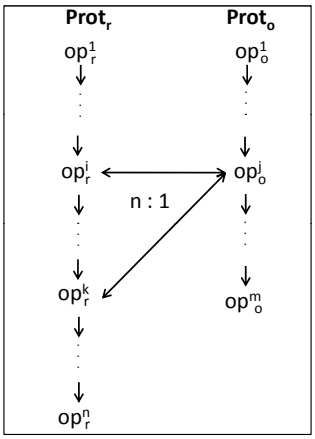

Figure 6.5: A diagrammatic Representation of a n : 1 Operation correspondence

There can be different scenarios where a n : 1 operation correspondence is possible. Below, we briefly explain these different scenarios.

Different Scenarios for a n : 1 Operation Correspondence

While matching a service request under consideration with a particular offer, there are two potential scenarios that can lead to a n : 1 operation correspondence. These different scenarios are as follows:

Scenario 1: In contrast to Fig. 6.5 that depicts a general scenario for an n : 1 operation correspondence between a requested operation sequence and an offered operation, Fig. 6.6 particularly emphasizes the scenario where initially a 1 : 1 correspondence is established between a requested operation op^i_r and an offered operation op^j_o and it is later extended to a n : 1 correspondence.

This happens because the offered operation also completely satisfies the postconditions of some subsequent operations in the requested invocation sequence specified in the requester . This means that $\texttt{compSatisfies}_{\texttt{po}}(Post(op^j_o)$,

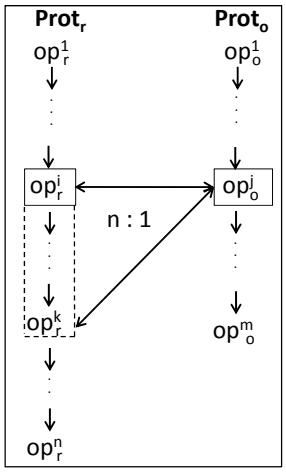

Figure 6.6: A Diagrammatic Representation for Scenario 1 of n : 1 Operation Correspondence

$Post(op_r^x)$) holds true for every requested operation op_r^x in the operation sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$.

For example, as shown in the last section, *checkAvailability()* of *HRS* has 1 : 1 correspondence to *searchRoom()* of *HotelX*. However, Fig. 6.7 shows that since the postconditions of the next requested operation in the requested invocation sequence *viewDetails()* are also completely satisfied by *searchRoom()*, this correspondence can be extended to a n : 1 correspondence between *checkAvailability()*→*viewDetails()* and *searchRoom()*.

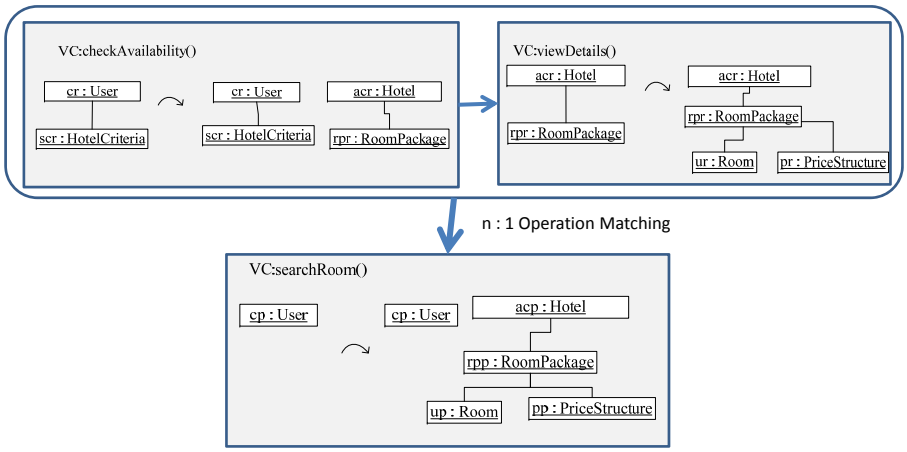

Figure 6.7: n : 1 Operation correspondence Example for `Scenario 1`

Scenario 2: The second possible scenario for a n : 1 operation correspondence is shown in Fig. 6.8. In contrast to the general scenario in Fig. 6.5, it particularly highlights the situation where a 1 : 1 operation correspondence is not possible between op_r^i and op_o^j because although op_o^j completely satisfies the postconditions of op_r^i but the preconditions required for its invocation are not satisfied by op_r^i. In this case, some subsequent operations of op_r^i in the requested invocation sequence participate to satisfy the preconditions of op_o^j. This leads to a n : 1 correspondence between the sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$ and op_o^j where the combined preconditions of the operations in the sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$ satisfy the preconditions of op_o^j. In this case, the postconditions of the participating subsequent operations of op_r^i are also completely satisfied by op_o^j, i.e., $\texttt{compSatisfies}_{\texttt{po}}(Post(op_o^j), Post(op_r^x))$ holds `true` for every requested operation op_r^x in the requested operation sequence. Similar to Scenario 1, this n : 1 operation correspondence can be further extended with some subsequent requested operations

in the invocation sequence if their postconditions are also satisfied by op_o^j.

This scenario can be better explained with the help of an example shown in Fig. 6.9. In this example, the requester requires to validate the payment credentials of the user through *validatePaymentCredentials()*. The requirements specified in this requested operation are completely satisfied by the offered operation *payForBooking()*, i.e., $\texttt{compSatisfies}_{\texttt{po}}(Post(payForBooking()), Post(validatePaymentCredentials()))$ holds `true`. But $\texttt{satisfies}_{\texttt{pr}}(Pre(validatePaymentCredentials()), Pre(payForBooking()))$ does not hold `true` because *payForBooking()* also requires a `HotelBooking` object and its link to `User` while payment, which is not offered in the preconditions of *validatePaymentCredentials()*.

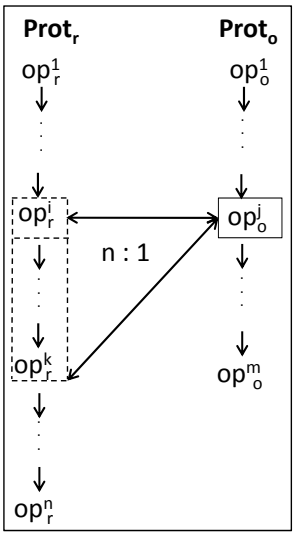

Figure 6.8: A Diagrammatic Representation for Scenario 2 of n : 1 Operation Correspondence

In this case, a 1 : 1 correspondence is not possible between the requested and the offered operation because although the offered operation satisfies the postconditions of the requested operation but its preconditions are not satisfied. These are satisfied by a subsequent operation in the requester invocation sequence *payForReservation()*, which offers a `HotelBooking` object linked to `User` in its preconditions. In this case, the postconditions of *payForReservation()* are also completely satisfied by the offered operation *payForBooking()*. Hence, a n : 1 operation correspondence can be defined between *validatePaymentCredentials()* $\rightarrow$ *payForReservation()* and $payForBooking()$. On further investigation, it is deduced that the postconditions of next requested operation in the invocation sequence *generateReceipt()* are also satisfied by *payForBooking()*. Consequently, the requested operation sequence in the determined n : 1 operation correspondence is further extended.

Here it is worth mentioning that the n : 1 operation correspondence in the given scenario can be termed as an *exceptional* case of an operation correspondence. This is because although such an operation correspondence can be determined during operation matching process during service discovery phase but at the run time, its invocation can lead to a possible deadlock situation. On one hand, the invocation of the offered operation op_o^j is not possible until all its preconditions are satisfied through the invocation of

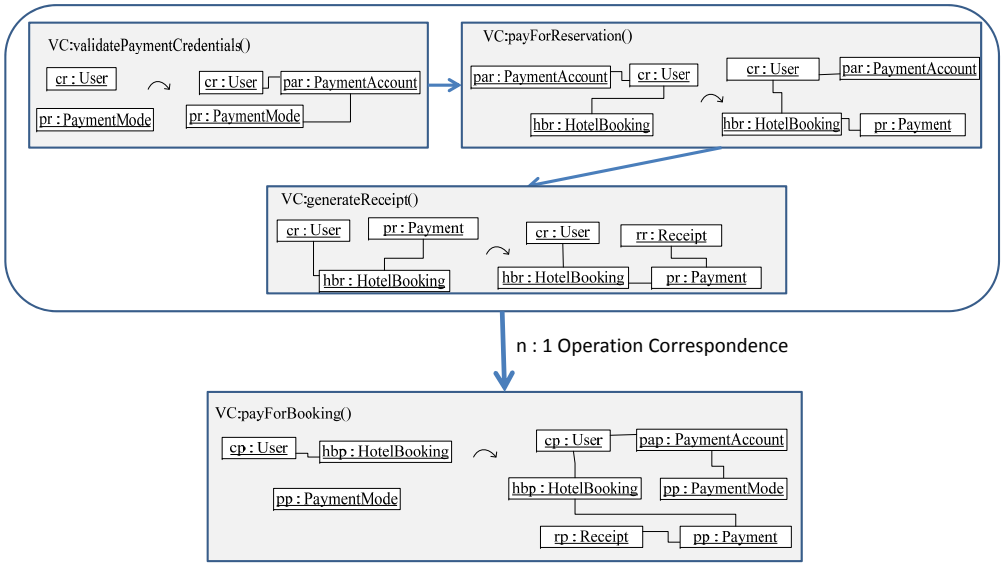

Figure 6.9: n : 1 Operation correspondence Example for `Scenario 2`

the corresponding requested operations in the sequence $op_r^a \rightarrow \cdots \rightarrow op_r^k$. On the other hand, one requested operation in the sequence can only be invoked after its predecessor's invocation is successful and its postconditions are satisfied.

In such a situation, the operation correspondence determined at design time cannot be invoked successfully at runtime. A possible solution is that after the operation matching process is complete, the service requester can be prompted by OTF provider to deal with such exceptional operation correspondences by restructuring its request such that all the preconditions of the offered operation op_o^j are available to it before its invocation. Such a restructuring leads to an n : 1 operation correspondence conforming to scenario 1.

For instance, in the example operation correspondence, the requester can be prompted to ask if he can restructure the request and also provide a `HotelBooking` object and its link to `User` in the preconditions of *validatePaymentCredentials()*. If this is possible, the determined operation correspondence can also be invoked successfully at runtime.

In the next section, we will give the formal definition for a n : 1 operation correspondence encompassing these two scenarios.

Definition for n : 1 Operation Correspondence

Similar to the definition of a 1 : 1 operation correspondence given in Sec. 6.3.1, we define a n : 1 operation correspondence as follows:

Definition 7 (n : 1 Operation Correspondence)**.** *Given the following:*

- *A request r;*
- *An offer o;*
- *A requested operation sequence seq_r in r represented as $op_r^i \rightarrow \cdots \rightarrow op_r^k$ with visual contracts $L_r^i \rightarrow R_r^i \ , \cdots , L_r^k \rightarrow R_r^k$;*
- *An offered operation op_o^j in o with visual contract $L_o^j \rightarrow R_o^j$.*

$(seq_r \ , op_o^j)$ is termed as a n : 1 correspondence, if there exists an element correspondence `corr` *(according to Def. 6) with the following properties:*

P1: $\forall y \in L_o^j \ \exists x \in \bigcup_{p=i}^{k} L_r^p$ *: x* ***corr*** *y, i.e., for every element in L_o^j there is a corresponding element in* $\bigcup_{p=i}^{k} L_r^p$*;*

P2: $\forall x \in \bigcup_{p=i}^{k} L_r^p \setminus R_r^p \ \exists y \in L_o^j \setminus R_o^j$ *: x* ***corr*** *y, i.e., for every element in* $\bigcup_{p=i}^{k} L_r^p$ *meant to be deleted by the operations in $op_r^i \rightarrow \cdots \rightarrow op_r^k$, there is a corresponding element that is indeed deleted by op_o^j;*

P3: $\forall x \in \bigcup_{p=i}^{k} L_r^p \cap R_r^p \ \exists y \in L_o^j \cup R_o^j$ *: x* ***corr*** *$y \Rightarrow y \in L_o^j \cap R_o^j$, i.e., every element in L_o^j that corresponds to an element meant to be preserved by operations in $op_r^i \rightarrow \cdots \rightarrow op_r^k$ is indeed preserved by op_o^j;*

P4: $\forall x \in \bigcup_{p=i}^{k} R_r^p \setminus L_r^p \ \exists y \in R_o^j \setminus L_o^j$ *: x* ***corr*** *y, i.e., for every element in* $\bigcup_{p=i}^{k} R_r^p$ *meant to be created by the operations in $op_r^i \rightarrow \cdots \rightarrow op_r^k$, there is a corresponding element that is indeed created by op_o^j.* □

We will explain the definition of a n : 1 correspondence with the help of two examples shown in Fig. 6.7 and Fig. 6.9, which represent the two different scenarios of n : 1 operation correspondence mentioned earlier.

According to Def. 7, op_r^i is the first operation in the given requested operation sequence whose postconditions are completely satisfied by op_o^j, i.e., $\texttt{compSatisfies}_{\texttt{po}}(Post(op_r^i), Post(op_o^j))$. On the other hand, it may satisfy some or all preconditions of op_o^j.

In the first scenario of n : 1 correspondence shown in Fig. 6.7, *checkAvailability()* acts as op_r^i in the seq_r = *checkAvailability()* → *viewDetails()*. `P1` specifies that for every element in the preconditions of the offered operation, there is a corresponding element in the preconditions of an operation in the operation sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$. For the example under consideration, for `cp:User` in the preconditions of offered operation, there is a corresponding element `cr:User` in the preconditions of op_r^a. `P2` is trivial in this case as no elements are deleted by any operation in seq_r. `P3` specifies that every element in the preconditions of the offered operation that corresponds to some element preserved by any operation in seq_r is also preserved by the offered operation. `cp` in the preconditions of *searchRoom()* corresponds to the preserved element `cr` in *checkAvailability()* and is indeed preserved. `P4` specifies that for every element created by any operation in seq_r, there is a corresponding element created by the offered operation. In the given example, for every element created by operations in *checkAvailability()* → *viewDetails()*, there is indeed a corresponding element created by *searchRoom()*. Based on this, it can be deduced that a **corr** comprising of corresponding elements exists that satisfies `P1`,...,`P4` and hence there is a n : 1 operation correspondence of the form (*checkAvailability()* → *viewDetails()* , *searchRoom()*).

For the example in Fig. 6.9, *validatePaymentCredentials()* represents op_r^i in seq_r and the later operation *payForReservation()* participates to satisfy the preconditions of *payForBooking()*. Additionally, the postconditions of next requested operation in the sequence *generateReceipt()* is also satisfied by *payForBooking()*. Satisfying `P1`, for the elements in the preconditions of *payForBooking()*, there are corresponding elements in the combined preconditions of requested operation sequence. `P2` is trivial as there are no elements deleted by any operation in $op_r^i \rightarrow \cdots \rightarrow op_r^k$. Satisfying `P3`, `cp:User`, `hbp:HotelBooking`, and `pp:PaymentMode` in preconditions of *payForBooking()* corresponding to preserved elements in the requested operations are also preserved by *payForBooking()*. Similarly, `P4` concerning the creation of the elements is also satisfied. Hence, there is a n : 1 operation correspondence depicting scenario 2 of the form

(*validatePaymentCredentials()* → *payForReservation()* → *generateReceipt()* , $payForBooking()$).

Apart from the different scenarios and the structure of n : 1 operation correspondence, a strategy is also required to determine such correspondences between the requested and offered operations. In the next section, we specify our n : 1 operation matching algorithm, which allows to determine an offered operation corresponding to a sequence of requested operations leading to valid n : 1 operation correspondences conforming to Def. 7.

n : 1 Operation Matching Algorithm

Our n : 1 operation matching algorithm is specified in Listing 2. As input, it takes the requested operation op_r^i and the offered operation op_o^j with which op_r^i has a 1 : 1 operation correspondence or where such a 1 : 1 operation correspondence is not possible according to the situation in scenario 2. Additionally, it takes as input the end-to-end invocation sequence $InSeq_r$ in the requester Protocol. As output, it returns a possible n : 1 correspondence $Corr$ between a requested operation sequence containing op_r^i and the offered operation op_o^j.

For the example in Fig. 6.7, after a 1 : 1 correspondence between *checkAvailability()* of *HRS* and *searchRoom()* of *Hotel X* is detected, n : 1 operation matching algorithm can be initiated for the requested and offered operation. According to our particular scenario, the requester protocol has a single end-to-end invocation sequence $checkAvailability() \rightarrow viewDetails() \rightarrow makeReservation() \rightarrow addFeature() \rightarrow searchFlight() \rightarrow bookFlight() \rightarrow makePayment()$.

The algorithm in Listing 2 works as follows:

Step ① Seq_{result} is initiated to represent the requested operation sequence that is built gradually during the course of this algorithm and has a n : 1 operation correspondence with the offered operation op_o^j. For the requested operation op_r^i, its postconditions are *completely satisfied* by op_o^j. Therefore, op_r^i is the first requested operation added to Seq_{result}. For the example in Fig. 6.7, *checkAvailability()* becomes the first operation of Seq_{result} as it has a 1 : 1 correspondence with *searchRoom()*. Additionally, a variable *PreCheck* is initiated to track the *satisfied* preconditions of the offered operation op_o^j. Initially, it contains the preconditions of first requested operation under consideration op_r^i. $PreSatisfied$ is used to build the sequence of subsequent operations of op_r^i whose combined preconditions satisfy the preconditions

Listing 2: Algorithm for n : 1 operation matching between a requested operation sequence from $InSeq_r$ including op_r^i and an offered operation op_o^j

Input: Offered Operation op_o^j
Input: Requested Operation op_r^i
Input: End-to-End Invocation Sequence $InSeq_r$ in requester's Service Protocol
// $Prot_r = op_r^1 \rightarrow ... \rightarrow op_r^i \rightarrow ... \rightarrow op_r^n$
Output: An operation Correspondence $Corr$

`nToOneMatching`$(op_r^i, InSeq_r, op_o^j)$

$Seq_{result} = op_r^i$; // Step ①
$PreCheck = Pre(op_r^i)$;
$PreSatisfied = null$;
$Corr = null$;
$op_x = InSeq_r$.`nextOp`$(\mathtt{op_r^i})$;
while *(!* `satisfies`$_{\mathtt{pr}}$*(*$PreCheck$ *,* $Pre(op_o^j)) \wedge (op_x != null)$ *)* **do**
// Step ②
  if `compSatisfies`$_{\mathtt{po}}$*(*$Post(op_o^j)$ *,* $Post(op_x)$ **then** // Step ③
    $PreSatisfied = PreSatisfied + op_x$;
    $PreCheck = PreCheck \cup Pre(op_x)$;
    $op_x = InSeq_r$.`nextOp`$(\mathtt{op_x})$;
  end
  else
    $Exit$; // Step ④
  end
end
if `satisfies`$_{\mathtt{pr}}$ *(*$PreCheck$ *,* $Pre(op_o^j))$ **then** // Step ⑤
  $Seq_{result} = Seq_{result} + PreSatisfied$;
  while *(*$compSatisfies_{po}(Post(op_o^j)$ *,* $Post(op_x)) \wedge (op_x != null))$
  do // Step ⑥
    $Seq_{result} = Seq_{result} + op_x$;
    $op_x = InSeq_r$.`nextOp`$(\mathtt{op_x})$;
  end
end
if Seq_{result}.*HasMultOps()* **then** // Step ⑦
  $Corr$ = `createNTo1Correspondence`$(Seq_{result}$ *,* $op_o^j)$;
end
return $Corr$; // Step ⑧
end

of op_o^j. $Corr$ represents the operation correspondence resulting from the operation matching. op_x tracks the currently traversed requested operation in the invocation sequence.

Step ② Next, the invocation sequence $InSeq_r$ is traversed through `nextOp()` until either $PreCheck$ satisfies the preconditions of op_o^j or the invocation sequence ends. For the example in Fig. 6.7, further traversal is not required as $PreCheck$ already satisfies the preconditions of *searchRoom()*. However, in Fig. 6.9, *validatePaymentCredentials()* does not satisfy preconditions of *payForBooking()*. Therefore, the traversal starts from *payForReservation()*.

Step ③ If the postconditions of the traversed requested operation op_x are completely satisfied by op_o^j, then it is added to the sequence $PreSatisfied$. $PreCheck$ is updated to include the preconditions of the traversed requested operation. Additionally, the next operation in the invocation sequence is selected for the next iteration. For the given example, *payForReservation()* is added to $PreSatisfied$ and $PreCheck$ is updated to contain the combined preconditions of *validatePaymentCredentials()* and *payForReservation()*. The next operation to be traversed is *generateReceipt()*. As $PreCheck$ built so far satisfy the preconditions of *payForBooking()*, further traversal is not carried out.

Step ④ If for a traversed requested operation, its postconditions are not satisfied by op_o^j, a valid $PreSatisfied$ cannot be built and hence further traversal is stopped.

Step ⑤ If the $PreCheck$ built so far satisfies the preconditions of op_o^j, the invocation sequence can be further traversed to check if the postconditions of any further requested operations are also satisfied by op_o^j. In this case, Seq_{result}, which so far contains op_r^i is combined with $PreSatisfied$ to create the requested operation sequence whose postconditions are completely satisfied by op_o^j. At this step, Seq_{result} contains *checkAvailability()* for the example in Fig. 6.7 and *validatePaymentCredentials()* → *payForReservation()* for the example in Fig. 6.9.

Step ⑥ $InSeq_r$ is traversed further until all the subsequent operations whose postconditions are satisfied by op_o^j are added to Seq_{result}. After this step, Seq_{result} contains *checkAvailability()* → *viewDetails()* for the example in Fig. 6.7 and *validatePaymentCredentials()* → *payForReservation()* → *generateReceipt()* for the example in Fig. 6.9.

Step ⑦ If an operation sequence Seq_{result} could be determined that comprise of requested operations other than op_r^i, then an n : 1 correspondence $Corr$ is created through `createNTo1Correspondence()`.

Step ⑧ The determined n : 1 correspondence is returned as an output of the algorithm, which will be empty if no valid Seq_{result} is determined.

In the given case, the n : 1 operation matching algorithm successfully maps *checkAvailability*() → *viewDetails*() of HRS to *searchRoom()* of Hotel X and the resulting correspondence satisfies Def. 7. Similarly, the proposed algorithm also determines the n : 1 correspondence between *validatePaymentCredentials()* → *payForReservation()* → *generateReceipt()* and *payForBooking()* shown in Fig. 6.9.

Apart from the scenarios for n : 1 operation correspondence, there are also certain scenarios where a 1 : n operation correspondence may be established between the service request and the offer. In the next section, we will describe those scenarios and discuss in detail our strategy to determine such 1 : n operation correspondences.

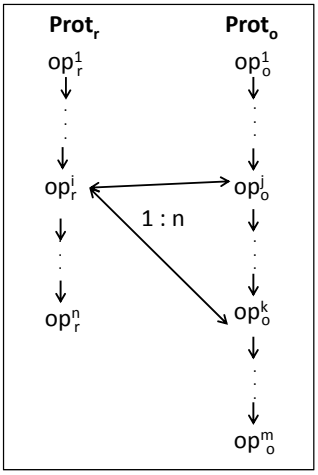

Figure 6.10: A diagrammatic Representation of a 1 : n Operation correspondence

6.3.3 1 : n Operation Matching

A 1 : n operation correspondence between a requested operation and a sequence of offered operations can be described as a correspondence where

the requirements specified in the requested operation are fulfilled by the corresponding sequence of invocable offered operations whose preconditions are satisfied. Fig. 6.10 shows a diagrammatic representation of such a 1 : n operation correspondence between the requested operation op_r^i and the offered operation sequence $op_o^j \rightarrow \cdots \rightarrow op_o^k$.

Different scenarios for 1 : n correspondence are discussed in the next section in detail.

Different Scenarios for a 1 : n Operation Correspondence

Similar to n : 1 operation correspondence, certain scenarios can arise while matching the service request and offer that can lead to a 1 : n operation correspondence. These are as follows:

Scenario 1: In contrast to the general scenario presented in Fig. 6.10, Fig. 6.11 visualizes a particular scenario for 1 : n operation correspondence, which arises from the situation discussed in scenario 2 in Sec. 6.3.2. This means that a 1 : 1 correspondence is not possible between a requested operation op_r^i and an offered operation op_o^k because although op_o^k completely satisfies the requirements specified in op_r^i but its invocation is not possible by op_r^i, i.e., $\texttt{compSatisfies}_{\texttt{po}}(Post(op_o^k), Post(op_r^i))$ is `true` but $\texttt{satisfies}_{\texttt{pr}}(Pre(op_r^i), Pre(op_o^k))$ is not `true`.

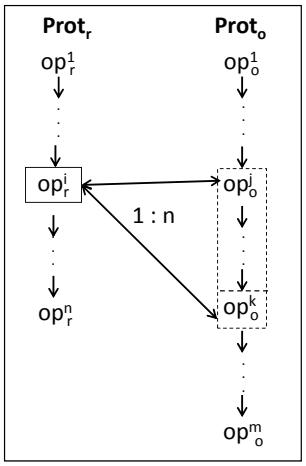

Figure 6.11: A Diagrammatic Representation for Scenario 1 of 1 : n Operation Correspondence

In this case, the postconditions of some operations earlier to op_o^k in an offered invocation sequence, i.e., $op_o^j \rightarrow \cdots \rightarrow op_o^{k-1}$ participate to satisfy the preconditions of op_o^k resulting in a 1 : n operation correspondence between op_r^i and $op_o^j \rightarrow \cdots \rightarrow op_o^k$. In this case, it is also made sure that the operations in $op_o^j \rightarrow \cdots \rightarrow op_o^{k-1}$ are also invocable, i.e., their preconditions are satisfied by combining the preconditions of op_r^i and postconditions of the earlier offered operations in the determined operation sequence. This scenario is explained with the help of an example given in Fig. 6.12.

Due to different granularity levels of the service request and offer, such a scenario can occur where the provider offers more functionality as compared to what is required by the requester. This can lead to situations where

certain offered operations do not fulfill any requester requirements but they have to be a part of the operation correspondences as they have created elements that are required in order to allow invocation of the other offered operations in the correspondence.

As seen in this example, $\texttt{compSatisfies}_{\texttt{po}}(Post(bookRoom()), Post(makeReservation()))$ as a `HotelBooking` is created and is linked to the respective `User` and the selected `RoomPackage`. On the other hand, $\texttt{satisfies}_{\texttt{pr}}(Pre(makeReservation()), Pre(bookRoom()))$ is not true because in order to create the `HotelBooking`, *bookRoom()* also requires the `Facility` selected by the `User` and his `Voucher` apart from the selected `RoomPackage`, which are not provided in the preconditions of *makeReservation()*. In this case, we assume that a correspondence according to scenario 2 of n : 1 operation correspondence is also not possible. Rather, these elements are created in some earlier operations *selectFacility()* and *getDiscountVoucher()* in the offered invocation sequence and are reused by *bookRoom()*. This results in a 1 : n operation correspondence between *makeReservation()* and *selectFacility()* → *getDiscountVoucher()* → *bookRoom()*. In this case, *selectFacility()* and *getDiscountVoucher()* are a part of the correspondence because their preconditions are also satisfied by *makeReservation()*.

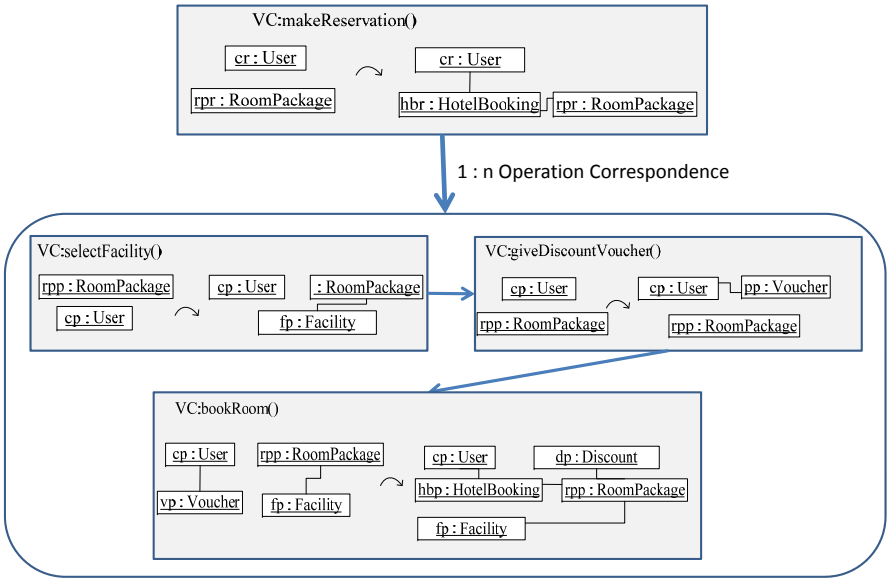

Figure 6.12: Example for `Scenario 1` of 1 : n Operation correspondence

Scenario 2: This scenario is shown in Fig. 6.13 and results from the situation where it is not possible to establish a 1 : 1 correspondence for a requested operation op_r^i with any offered operation because none of the offered operations completely satisfy the requirements specified in the requested operation under consideration.

In such a situation, there is a possibility to find an offered operation op_o^j, which *partially* satisfies the postconditions of op_r^i, i.e. it creates or deletes some of the elements created or deleted by op_r^i. To this extent, this is formulated as $\texttt{partSatisfies}_{\texttt{po}}(Post(op_o^j), Post(op_r^i))$. In this case, a sequence of subsequent offered operations $op_o^{j+1} \rightarrow \cdots \rightarrow op_o^k$ in the invocation sequence contribute to satisfy the postconditions of op_r^i. Similar to scenario 1, all the offered operations in the sequence are invocable.

As a result, a 1 : n operation correspondence can be established between op_r^i and $op_o^j \rightarrow \cdots \rightarrow op_o^k$, where the postconditions of all the offered operations operations in the sequence participate to completely satisfy the postconditions of the requested operation. Additionally, the preconditions of every offered operation in the sequence is satisfied by combining the preconditions of the requested operation and the postconditions of its earlier operations in the given operation sequence.

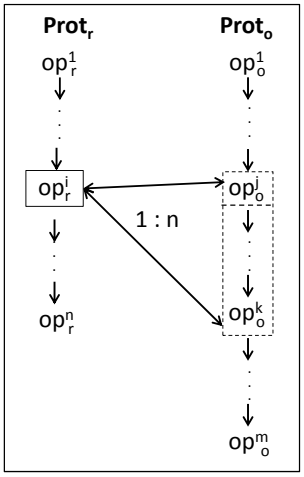

Figure 6.13: A Diagrammatic Representation for Scenario 2 of 1 : n Operation Correspondence

An example for this scenario is given in Fig. 6.13 where the postconditions of the requested operation *makeReservation()* are partially satisfied by the offered operation *bookRoom()*, i.e., a `HotelBooking` is created for the selected `RoomPackage` and its respective links with `User` and `RoomPackage` are created. This means that $\texttt{compSatisfies}_{\texttt{po}}(Post(bookRoom()), Post(makeReservation()))$ does not hold but $\texttt{partSatisfies}_{\texttt{po}}(Post(bookRoom()), Post(makeReservation()))$ holds true.

In this case, a 1 : n correspondence can be determined between *makeReservation()* and the sequence of the offered operations, i.e., *bookRoom()* $\rightarrow$ *sendNotification()*, as every offered operation partially satisfy postconditions of the requested operation and as a result, the postconditions of the requested operation are completely satisfied. On the other hand,

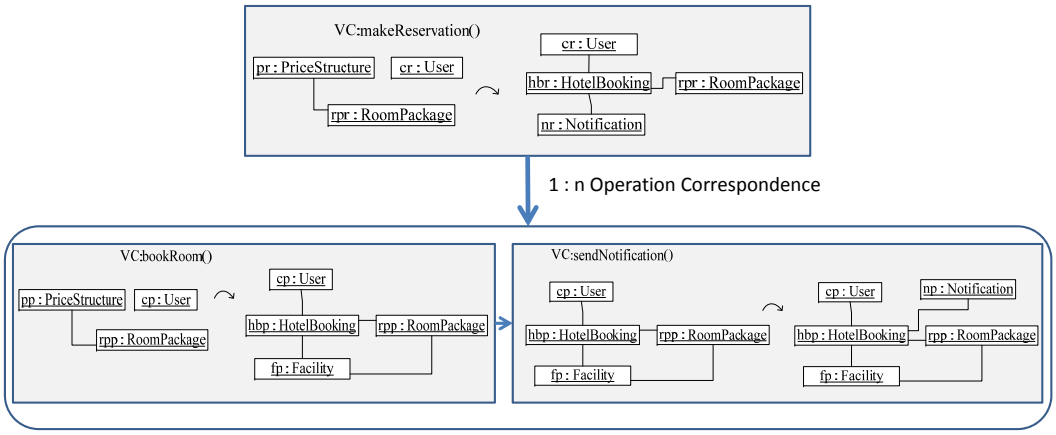

Figure 6.14: Example for `Scenario 2` of 1 : n Operation correspondence

the preconditions of every offered operation are satisfied by a combination of the preconditions of the *makeReservation()* and the postconditions of the earlier operations is the sequence.

For example, $\texttt{satisfies}_{\texttt{pr}}(Pre(makeReservation()), Pre(bookRoom()))$ holds true and the preconditions of the offered operation *sendNotification()* is satisfied by a combination of the preconditions of *makeReservation()* and the postconditions of *bookRoom()* where the *Facility* linked to the booked *RoomPackage* required for creating the *Notification* is created by *bookRoom()* which additionally book certain facilities when a room is booked.

Scenario 3: This scenario is diagrammatically represented in Fig. 6.15 and also results from the situation depicted in scenario 2 where a 1 : 1 correspondence cannot be established for a requested operation op_r^i but there is an offered operation op_o^j, such that, $\texttt{partSatisfies}_{\texttt{po}}(Post(op_o^j), Post(op_r^i))$. However, unlike scenario 2, every offered operation in the corresponding sequence may not partially satisfy the postconditions of the requested operation. Rather, it is a part of

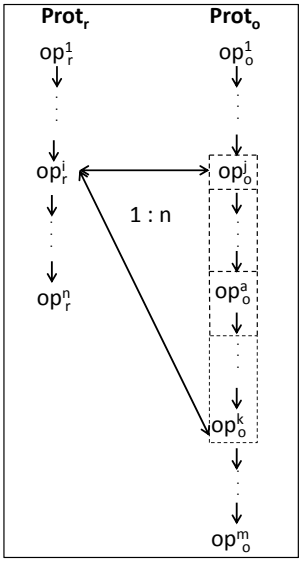

Figure 6.15: A Diagrammatic Representation for Scenario 3 of 1 : n Operation Correspondence

the corresponding operation sequence because it enables the invocation of some subsequent operation in the sequence through its postconditions. In Fig. 6.15, op_o^a in $op_o^j \rightarrow \cdots \rightarrow op_o^k$ shows such an operation in the matched sequence, which only participates to make the invocation of some later operations possible.

In this scenario, there is also the possibility similar to scenario 1, where the corresponding operation sequence may have to be extended backwards to some earlier operations in the invocation sequence in order to satisfy the preconditions of some operation in this sequence.

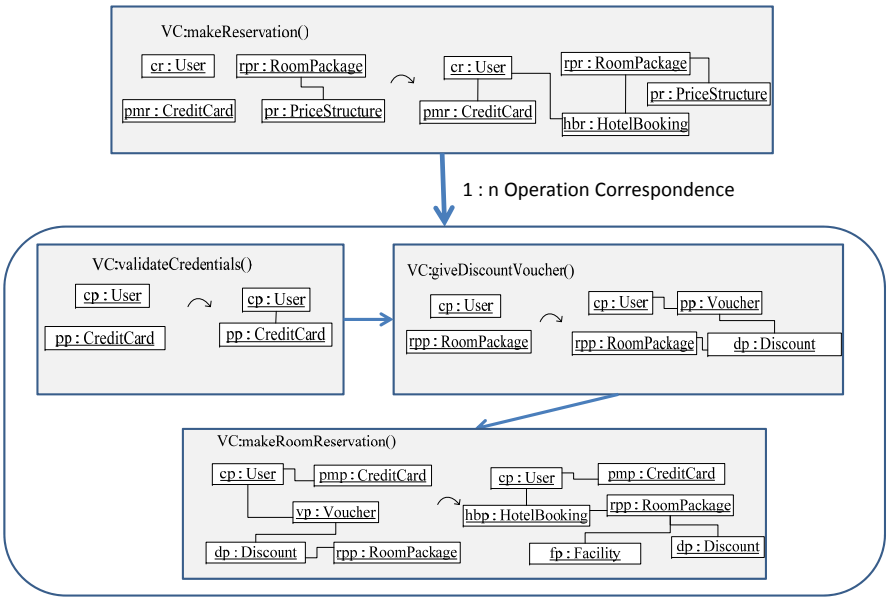

Figure 6.16: Example for `Scenario 3` of 1 : n Operation correspondence

An example for this scenario is shown in Fig. 6.16. According to this example, $\texttt{partSatisfies}_{\texttt{po}}(Post(validateCredentials()), Post(makeReservation()))$ holds `true` and *makeReservation()* has a 1 : n operation correspondence to *validateCredentials()* $\rightarrow$ *getDiscountVoucher()* $\rightarrow$ *makeRoomReservation()*. In this case, *getDiscountVoucher()* is included in the corresponding sequence only because it satisfies some postconditions of the subsequent operation *makeRoomReservation()*, i.e., it creates elements of type `Voucher`, `Discount` and their respective links, which are used to satisfy the preconditions of *makeRoomReservation()*. Here it is important to note that like other operations in the corresponding sequence, such intermediate operations also have to be invocable, i.e., their preconditions

are satisfied like other operations in the sequence. In the given example, *getDiscountVoucher()* becomes a part of the corresponding sequence as its preconditions are satisfied by the preconditions of *makeReservation()*.

For scenario 2 and 3 described above, their are certain considerations, which are explained in detail as follows:

Firstly, an exceptional situation arises when for some element that is preserved by the requested operation, there is a corresponding element that is newly created in the corresponding offered operation. In this case, during operation matching phase, if a subsequent offered operation in the sequence requires the said element in its preconditions, it can select between the element provided in the preconditions of the requested operation and the one created in the earlier offered operation. However at runtime, this can lead to a complex situation where two corresponding elements point to different objects with different attribute values. As a result, a subsequent offered operation that requires the said element in its preconditions has to choose from two different objects. It can result in erroneous results at run time. This exceptional situation can be explained with the help of an example shown in Fig. 6.17.

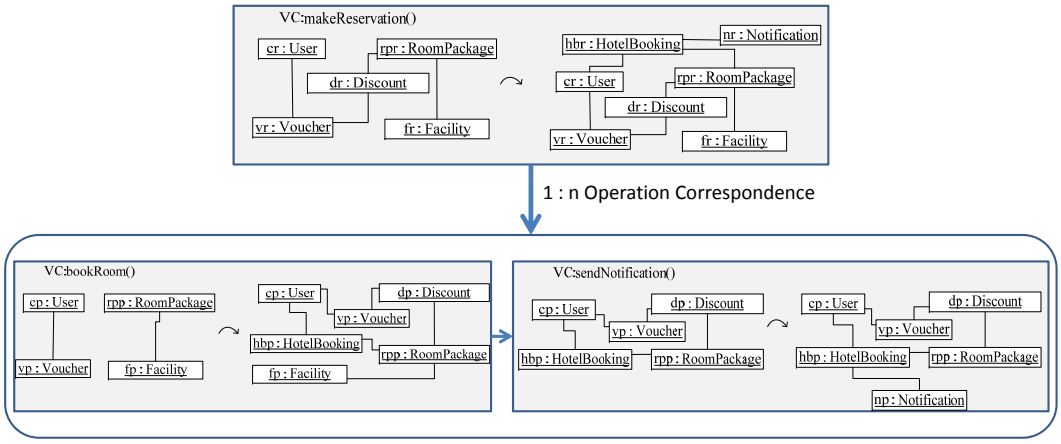

Figure 6.17: Example for a 1 : n Operation correspondence with corresponding Elements referring to different Objects at Runtime

In this case, the requested operation *makeReservation()* allows the `User` with the `Voucher` and an attached `Discount` to book a `RoomPackage` with certain `Facility`. In contrast, the first offered operation in the matched sequence *bookRoom()* books the `RoomPackage` for the `User` and also creates the `Discount` on the basis of the `Voucher` during the booking process. Hence

in this case, `dr:Discount` in the requested operation and `dp:Discount` in the offered operation point to different objects at runtime. In this case, the next offered operation *sendNotification()*, which requires the `Discount` object and the link between `Discount` and `Voucher` can be fulfilled by the preconditions of *bookRoom()* or the newly created elements in the postconditions of *sendNotification()*.

In such a situation, after the operation matching process, the OTF provider can prompt the service requester about such an exceptional 1 : n operation correspondence. In this case, the requester can choose to adapt its service request in order to enable the given operation correspondence at runtime. If such an adaption is not possible, the service requester may discard the given operation correspondence. For instance, in the given example, the service requester can decide that instead of offering a `Discount` attached to `Voucher` in the preconditions of *makeReservation()*, it can rely on the service offer to create a `Discount` on the basis the `Voucher` and hence, the service requester can be modified accordingly and consequently, the determined 1 : n operation correspondence can be opted by the service requester.

Another exceptional situation is encountered when a postcondition in the requested operation is negated by some contradicting postconditions of the corresponding offered operations. For instance, there can be a situation where an element created by the requested operation has a corresponding element created by an offered operation. However, this newly created element may be deleted by some subsequent offered operation in the matched sequence. This is explained with the help of an example in Fig. 6.18, i.e., a variant of Fig. 6.17.

According to this example, the requested operation *makeReservation()* specifies that while booking the `RoomPackage`, a `Discount` needs to be created on the basis of the `Voucher` for the booked `RoomPackage`. This requirement is indeed fulfilled by the first operation in the corresponding offered operation sequence, i.e., *bookRoom()*. However, in the next offered operation *sendNotification()*, the service provider along with sending a `Notification` to the `User` for his `HotelBooking`, also deletes his `Voucher` and the related `Discount` as they are already consumed and can't be reused for any future booking.

In such a situation, although *apparently* all the postconditions of the requested operation are satisfied by the corresponding sequence of offered operations but in reality, some of these postconditions are later on negated in a subsequent offered operation in the corresponding sequence. Therefore, it is not a valid 1 : n operation correspondence and must be discarded during

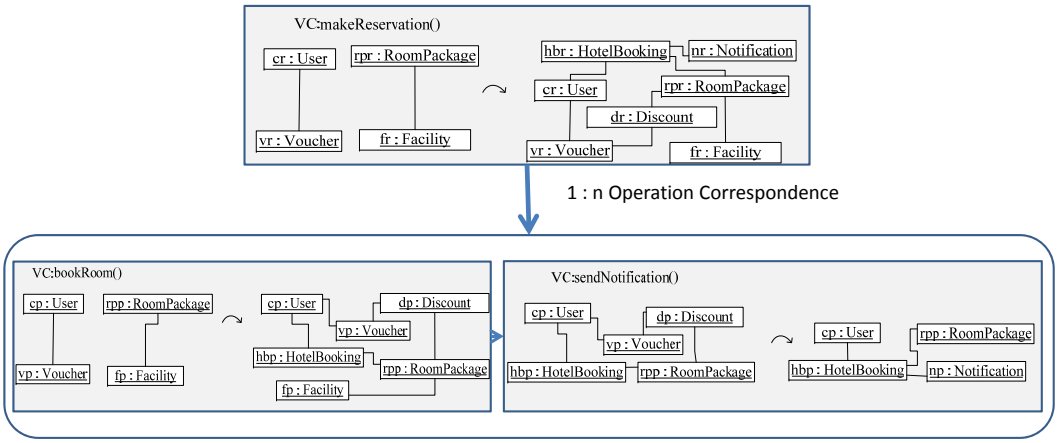

Figure 6.18: Example for a 1 : n Operation correspondence with contradicting postconditions of the offered Operations

the matching process.

Definition for 1 : n Operation Correspondence

A definition for a 1 : n operation correspondence is as follows:

Definition 8 (1 : n Operation Correspondence)**.** *Given the following:*

- *A request r;*
- *An offer o;*
- *A requested operation op_r^i in r with visual contract $L_r^i \rightarrow R_r^i$;*
- *An offered operation sequence seq_o in o represented as $op_o^j \rightarrow \cdots \rightarrow op_o^k$ with visual contracts $L_o^j \rightarrow R_o^j$, $\cdots$, $L_o^k \rightarrow R_o^k$;*

$(op_r^i$, $seq_o)$ is termed as a n : 1 correspondence, if there exists an element correspondence **`corr`** *(according to Def. 6) with the following properties:*

P1: For every operation op_o^a in seq_o, $\forall y \in L_o^a \; \exists x \in L_r^i \vee y \in \bigcup_{p=j}^{a-1} R_o^p \setminus L_o^p$:
*x **corr** y, i.e., for every element in L_o^a of every operation op_o^a in seq_o, there is a corresponding element in L_r^i or it is already created by an earlier operation in the sequence $op_o^j \rightarrow \cdots \rightarrow op_o^{a-1}$;*

P2: $\forall x \in L_r^i \setminus R_r^i \; \exists y \in L_o^a \setminus R_o^a$ *for some* op_o^a *in* seq_o *:* x ***corr*** y*, i.e., for every element in* L_r^i *that is meant to be deleted by* op_r^i *there is a corresponding element that is indeed deleted by some operation* op_o^a *in* seq_o*;*

P3: $\forall x \in L_r^i \cap R_r^i \; \exists y \in L_o^a \cup R_o^a$ *for every operation* op_o^a *in* seq_o *:* x ***corr*** y $\Rightarrow y \in L_o^a \cap R_o^a$*, i.e., every element in* L_o^a *of every operation* op_o^a *in* seq_o *that corresponds to an element meant to be preserved by* op_r^i *is indeed preserved by* op_o^a*;*

P4: $\forall x \in R_r^i \setminus L_r^i \; \exists y \in R_o^a \setminus L_o^a \wedge y \notin \bigcup_{p=a+1}^{k} L_o^p \setminus R_o^p$ *for some* op_o^a *in* seq_o *:* x ***corr*** y*, i.e., for every element in* R_r^i *that is meant to be created by* op_r^i *there is a corresponding element that is indeed created by some operation* op_o^a *in* seq_o*. However, this created element is not deleted by some subsequent operation of* op_o^a *in* seq_o*.* □

We will explain the definition of a 1 : n correspondence with the help of the example shown in Fig. 6.16. In this case, `P1` specifies that for every element in the preconditions of every offered operation in the operation sequence, either there is a corresponding element in the preconditions of the requested operation or this is an element that is newly created by an earlier operation in the sequence. For the example under consideration, for `cp:User`, `pmp:CreditCard`, `rpp:RoomPackage` and their respective links in the preconditions of the operations in *validateCredentials()* → *getDiscountVoucher()* → *makeRoomReservation()*, there are corresponding elements `cr:User`, `pmr:CreditCard`, `rpr:RoomPackage` and the respective links in the preconditions of *makeReservation()*. Additionally, `vp:Voucher`, `dp:Discount` and their respective links in the preconditions of *makeRoomReservation()* are created by an earlier operation *getDiscountVoucher()*. Hence, `P1` is satisfied for the example under consideration. `P2` is trivial in this case as no elements are deleted by *makeReservation()*. `P3` specifies that every element in the preconditions of every offered operation in the sequence that corresponds to some element preserved by the requested operation is also preserved by the offered operation. In the given example, the elements meant to be preserved by the requested operation are the objects `cr`, `cor`, `pmr`, and `rpr` with their respective links. `cp` in the preconditions of every offered operation in the sequence, which corresponds to the preserved element `cr` in *makeReservation()* is indeed preserved by its constituting offered operations. Similarly satisfying `P3`, elements corresponding to `cor`, `pmr`, `rpr`, etc. are also preserved by their constituting offered operations. `P4`

specifies that for every element created by the requested operation, there is a corresponding element created by some offered operation in the sequence and these newly created elements are not deleted in a subsequent operation in the sequence. In the given example, for the newly created elements, `hbr:HotelBooking` and its links with `cr:User` and `rpr:RoomPackage`, a corresponding element `hbp:HotelBooking` and its respective links are created in *makeRoomReservation()*. Hence for the given example, an element correspondence exists, which satisfies the properties specified in Def. 8 and there is a valid 1 : n correspondence between *makeReservation()* and *validateCredentials()* → *getDiscountVoucher()* → *makeRoomReservation()*.

In the next section, we describe our 1 : n operation matching algorithm which determines a sequence of offered operations that matches a given requested operation leading to a 1 : n operation correspondence.

1 : n Operation Matching Algorithm

Listing 3: Algorithm catering different Scenarios for 1 : n Matching between a Requested Operation op_r^i and an offered operation sequence in $Prot_o$

Input: Requested Operation op_r^i
Input: Offered Operation op_o^j
// $\texttt{compSatisfies}_{\texttt{po}}$ ($Post(op_o^j)$, $Post(op_r^i)$)
Input: The offered Service Protocol $Prot_o$
Output: A resulting operation Correspondence $Corr$

`oneToNMatching`($op_r^i, op_o^j, Prot_o$)
  $Corr = null$;
  if $op_o^j != null$ **then** $Corr =$ `FirstScenarioMatching`($op_r^i, op_o^j, Prot_o$) ;
  else $Corr =$ `SecondScenarioMatching`($op_r^i, Prot_o$) ;
  return $Corr$;
end

Our 1 : n matching algorithm is specified in Listing 3 and is invoked if one of the situations explained in the scenarios is encountered. As input, it takes the requested operation under consideration op_r^i and an offered operation op_o^j, which completely satisfies the postconditions of op_r^i . Additionally, it takes the offered service protocol $Prot_o$ as input. As output, it returns $Corr$, i.e., a 1 : n correspondence between op_r^i and an offered operation sequence in $Prot_o$, where the preconditions of every offered operation in the

Listing 4: Algorithm for Scenario 1 for 1 : n Matching between a Requested Operation op_r^i and an offered operation sequence in $Prot_o$

Input: Requested Operation op_r^i
Input: Offered Operation op_o^j
// $\texttt{compSatisfies}_{\texttt{po}}$ ($Post(op_o^j)$, $Post(op_r^i)$)
Input: The offered Service Protocol $Prot_o$
Output: A resulting operation Correspondence $Corr$

```
FirstScenarioMatching(op_r^i, op_o^j, Prot_o)
    Corr = null;
    InSeq_o = null;
    Seq_result = op_o^j;
    while InSeq_o = Prot_o.nextInvSeq() do                    // Step ①
        if InSeq_o.contains(op_o^j) then
            Earlier = extendToEarlierOps(op_o^j, op_o^j, Pre(op_r^i), InSeq_o);
            Seq_result = Earlier + Seq_result;
            if Seq_result.HasMultOps() then
                Corr = create1ToNCorrespondence(op_r^i , Seq_result);
                Exit;
            end
        end
    end
    return Corr;
end
```

sequence are satisfied by op_r^i or its earlier operations in the sequence and the postconditions of op_r^i are completely satisfied by the offered operation sequence.

As seen in the Listing 3, the algorithm is further subdivided into two sub-algorithms that cater to different scenarios of 1 : n operation correspondence. If some op_o^j completely satisfies the postconditions of op_r^i then a 1 : n operation correspondence according to scenario 1 might be possible, which is determined through Listing 4. Here we briefly explain, how it works. Every end-to-end invocation sequence $InSeq_o$ in the offered protocol $Prot_o$ is considered and if op_o^j occurs in $InSeq_o$ then `extendToEarlierOps()` is invoked to determine any possible sequence of earlier operations from op_o^j in the invocation sequence $InSeq_o$ whose postconditions can be combined with preconditions of op_r^i to satisfy the preconditions of op_o^j. The working of `extendToEarlierOps()` will be explained in more detail later. If such sequence of earlier operations $Earlier$ exists then matching offered operation sequence Seq_{result} is created by combining $Earlier$ to op_o^j. This means that a 1 : n correspondence is possible between op_r^i and Seq_{result}. Such a 1 : n operation correspondence is created through `create1ToNCorrespondence()`. As a valid 1 : n correspondence is already found for op_r^i, further traversal the offered protocol $Prot_o$ is also terminated at this point.

Similarly, Listing 5 is invoked to determine a 1 : n operation correspondence according to scenario 2 and 3, if no offered operation completely satisfies the postconditions of op_r^i and hence op_o^j is $null$. As an example for such a 1 : n operation matching, we consider a scenario from our running example shown in Fig. 6.16. In this case, there is no offered operation of *HotelX* that completely satisfies the requirements specified in the requested operation *makeReservation()* of *HRS*. Hence, 1 : n operation matching algorithm has to be invoked to determine any possible 1 : n operation correspondence according to scenario 2 or 3. For the example under consideration, op_o^j is $null$ and the sub-algorithm in Listing 5 is invoked, which takes the requested operation op_r^i and the offered service protocol $Prot_o$ as input and returns any possible 1 : n operation correspondence as output.

The algorithm in Listing 5 works as follows:

Step ① At the start, $Corr$, $InSeq_o$, Seq_{result} as variables are initiated for the resulting operation correspondence, the traversed invocation sequence in the protocol and the resulting operation sequence, respectively. In order to match the requested operation op_r^i to an operation sequence in the offered protocol $Prot_o$, every end-to-end operation invocation

Listing 5: Algorithm for Scenario 2 and 3 for 1 : n Matching between a Requested Operation op_r and an offered operation sequence in $Prot_o$

Input: Requested Operation op_r^i
Input: The offered Service Protocol $Prot_o$
Output: A resulting operation Correspondence $Corr$

```
SecondScenarioMatching(op_r^i, Prot_o)
    Corr = null;
    InSeq_o = null;
    Seq_result = null;
    while InSeq_o = Prot_o.nextInvSeq() do                    // Step ①
        op_start = InSeq_o.firstOp();
        for op_start = InSeq_o.nextTraversableOp(op_start) do
            if partSatisfies_po(Post(op_start) , Post(op_r^i)) then
                if satisfies_pr(Pre(op_r^i) , Pre(op_start)) then
                Seq_result = op_start;                        // Step ②
                else
                    Earlier =
                    extendToEarlierOps(op_start, op_start, Pre(op_r^i), InSeq_o);
                    Seq_result = Earlier + op_start;
                end
                if Seq_result! = null then                    // Step ③
                    Corr = matchFurther(op_r^i, op_start, InSeq_o, Seq_result);
                    if Corr! = null then                      // Step 11
                        Exit;
                    end
                end
            end
        end
    end
    return Corr;                                              // Step 12
end
```

Listing 6: A function that further extends the matched operation sequence Seq_{result} from the matched offered operation op_{start} in the invocation sequence $InSeq_o$ and creates a possible correspondence $Corr$

Input: Requested Operation op_r^i
Input: The traversed invocation sequence $InSeq_o$
Input: The matching offered operation sequence built so far Seq_{result}
Input: The offered operation to start further matching op_{start}
Output: A resulting operation Correspondence $Corr$

```
matchFurther(op_r^i, op_start, InSeq_o, Seq_result)
    Corr = null;
    Earlier = null;
    op_x = InSeq_o.nextOp(op_start);                                  // Step ④
    PreCheck = Pre(op_r^i);
    PostCheck = Post(op_start);
    for every op in Seq_result do
        PreCheck = combinePre(PreCheck , Post(op));
    end
    while (!compSatisfies_po(PostCheck , Post(op_r^i))
    ∧ (op_x != null)) do                                              // Step ⑤
        if (!satisfies_pr(PreCheck , Pre(op_x))) then                 // Step ⑥
            Earlier = extendToEarlierOps(op_x , Seq_result.firstOp(),
            PreCheck , InSeq_o);
            Seq_result = Earlier + Seq_result;
            if Earlier == null then                                   // Step ⑦
                Seq_result = null;
                Exit;
            end
        end
        else if (contradicting (op_r^i, PostCheck , op_x)) then       // Step ⑧
            Seq_result = null;
            Exit;
        end
        Seq_result = Seq_result + op_x;                               // Step ⑨
        PreCheck = combinePre(PreCheck, op_x);
        PostCheck = combinePost(PostCheck , Post(op_x));
        op_x = InSeq_r.nextOp(op_x);
    end
    if (! compSatisfies_po(PostCheck , Post(op_r^i))) then
    Seq_result = null;                                                // Step ⑩
    else  Corr = create1ToNCorrespondence(op_r^i , Seq_result) ;
    return Corr
end
```

sequence $InSeq_o$ in $Prot_o$ is traversed. For the invocation sequence under consideration $InSeq_o$, every next operation op_{start} is a candidate as the potential first operation of the resulting operation sequence Seq_{result} if it *partially satisfies* the postconditions of op_r^i. In this case, `nextTraversableOp()` makes sure that op_{start} is the operation that has not be traversed and checked for matching in an earlier iteration so far. This aspect will be clarified in more detail later. In our given example, we assume that the first traversed invocation sequence in the provider protocol is $searchRoom() \rightarrow validateCredentials() \rightarrow giveDiscountVoucher() \rightarrow makeRoomReservation() \rightarrow notifyPer\text{-}Email()$. In this case, *validateCredentials()* is the first operation, which *partially satisfies* the postconditions of the *makeReservation()*, i.e., the credentials for the provided `CreditCard` are validated and as a valid `CreditCard` for the `Client`, a link is created between the two.

Step ② If the preconditions of op_{start} are satisfied by op_r^i, it becomes the first operation of the resulting sequence. For instance in the given example, the preconditions of *validateCredentials()* are satisfied by *makeReservation()*. Otherwise, some operations earlier to op_{start} in $InSeq_o$ can participate to satisfy the preconditions of op_{start}, i.e., the elements created in some earlier operations are used in op_{start}. To determine such a sequence of operations $Earlier$, a function `extendToEarlierOps()` traverses the $InSeq_o$ from op_{start} backwards. In case such a sequence of earlier offered operations $Earlier$ exists, it becomes part of the offered operation sequence Seq_{result}.

Step ③ At this point, Seq_{result} is either empty or it contains an offered operation op_{start} that partially satisfies the postconditions of the requested operation op_r^i and its preconditions are satisfied by combining the preconditions of op_r^i and earlier offered operations in Seq_{result}. In case Seq_{result} is empty, it means that a matching operation sequence cannot be built starting from op_{start} and the next operation in $InSeq_o$ that partially satisfies the postconditions op_r^i has to be considered. Any further matching will only be carried out if the Seq_{result} contains an op_{start} whose preconditions are satisfied.

In order to further build the resulting operation sequence Seq_{result} and create resulting 1 : n operation correspondence, function `matchFurther()` is called, which takes as input the requested operation op_r^i, the traversed invocation sequence $InSeq_o$, the matched operation sequence built so far Seq_{result}, the offered operation op_{start} in

$InSeq_o$ that partially satisfies the preconditions of op_r^i and from where further extension of Seq_{result} has to start. Here, we explain how this function carries out further matching.

Step ④ `matchFurther()` starts by initiating *PreCheck* and *PostCheck*, which are two variables that track the stepwise satisfaction of the preconditions of the currently-traversed offered operation and the postconditions of op_r^i. Initially, *PreCheck* only contains the preconditions of op_r^i and the *PostCheck* contains the postconditions of the first offered operation op_{start} that partially satisfies the postconditions of op_r^i. However, as the preconditions of every subsequent matched offered operation in the sequence is satisfied through a combination of preconditions of op_r^i and postconditions of the earlier operations, *PreCheck* needs to be updated accordingly with the postconditions of the operations in Seq_{result}.

For this purpose, special operator `combinePre` is used, which updates the *PreCheck* by combining it with newly created element in the postconditions of the offered operations in the matched sequence Seq_{result}. For the given example, *PreCheck* is updated by combining its existing elements, i.e., the preconditions of *makeReservation()* to the newly created elements in the postconditions of *validateCredentials()*. As a result, in addition to the existing elements, such as, the objects of type `User`, `CreditCard`, `PriceStructure`, and `RoomPackage` and a link between `RoomPackage` and `PriceStructure`, *PreCheck* is updated by adding a link between the objects of `User` and `CreditCard`.

Step ⑤ With the operation next to op_{start} as the starting operation, the $InSeq_o$ is iterated until either the postconditions of op_r^i are *completely satisfied* or the offered path ends. In the example, after the addition of *validateCredentials()* to the matched sequence, further traversal starts from *giveDiscountVoucher()*.

Step ⑥ Every traversed offered operation op_x is a candidate to be added to the matched sequence if it can be invoked in the given circumstances, i.e., its preconditions can be satisfied through *PreCheck*. If this is not the case, there can be a possibility that the matched sequence Seq_{result} could be extended backwards with some earlier operation in $InSeq_o$ to satisfy preconditions of op_x. For this purpose, `extendToEarlierOps()` traverses $InSeq_o$ backwards starting from the first operation in Seq_{result} to determine a sequence of earlier operations as *Earlier* whose postconditions can be combined with *PreCheck*

to satisfy the preconditions of op_x. The resulting *Earlier* is combined with Seq_{result}. In the given example, the preconditions of *giveDiscountVoucher()* are satisfied by *PreCheck* and no backward extension of Seq_{result} is required at this point.

Step ⑦ In case where *PreCheck* does not satisfy the preconditions of op_x and there is no possible backward extension *Earlier* that can contribute to satisfy the preconditions of op_x, Seq_{result} cannot be constructed further in this case. This means that for the particular op_{start} under consideration, Seq_{result} could not be constructed to form a 1 : n operation correspondence between op_r^i and Seq_{result}. Therefore, the traversal through the function `matchFurther()` has to be stopped and the `SecondScenarioMatching()` has to move on to the next *traversable* op_{start} in $InSeq_o$, which partially satisfies the postconditions of op_r^i and starts constructing Seq_{result} again. By *traversable* op_{start} we mean that the algorithm will skip the operations in $InSeq_o$ that have already been considered and added to Seq_{result} in the last iteration and through $InSeq_o$.`nextTraversableOp()`, an op_{start} will be selected that has not been traversed so far. This avoids redundant traversal of those parts of the invocation sequence that have already been considered for matching and do not require to be matched again in the next iteration.

Step ⑧ As explained earlier and specified in `P4` in Def. 8, any postconditions of an offered operation in the matched sequence must not *contradict* with some postconditions of the requested operation, i.e. it should not negate some requested postconditions that are already satisfied by some earlier operations in the matched sequence. In the algorithm, this is checked through the function `contradicting()`, which checks if the offered operation under consideration op_x contradicts with op_r^i in the sense that it negates some of its postconditions already satisfied by *PostCheck*, i.e., the variable to track the satisfied postconditions of op_r^i. An example for such a contradiction has already been discussed in detail with the help of the example shown in Fig. 6.18. If `contradicting()` holds true, then Seq_{result} cannot be constructed further for op_{start} and hence the traversal has to be started anew from the next *traversable* operation in $InSeq_o$.

Step ⑨ Once it is determined that the preconditions of the offered operation under consideration op_x can be satisfied either by the *PreCheck* constructed so far or through the participation of some earlier operations, i.e., *Earlier*, this means that Seq_{result} can be updated by extending

it backwards with *Earlier* and further extending it with op_x. Additionally, *PreCheck* is updated by combining it with the postconditions of op_x. Similarly, *PostCheck* is also updated by further adding the postconditions of op_x to it. Additionally, the operation next to op_x in $InSeq_o$ is selected for further traversal and further construction of Seq_{result}.

For the given example, *giveDiscountVoucher()* is added to the matched sequence Seq_{result}, which already contains *validateCredentials()* because its preconditions are satisfied by *PreCheck* constructed so far.

After *giveDiscountVoucher()*, next traversed operation is *makeRoomReservation()*. The preconditions of this offered operation are indeed satisfied by *PreCheck*. In this case, some elements in its preconditions, such as, the objects of type `Voucher`, `Discount` and their respective links are satisfied by the newly created element in the earlier operation in the sequence *giveDiscountVoucher()*. As a result, *makeRoomReservation()* is added to Seq_{result}. Additionally, *makeRoomReservation()* also satisfies the postconditions of *makeReservation()* that remain to be satisfied, e.g., creation of object of `HotelBooking` type and link between `User` and `HotelBooking` objects, etc.

Step ⑩ During the traversal, if the postconditions of op_r^i are not completely satisfied so far and the end of the invocation sequence is reached, this means that a valid Seq_{result} could not be found that can lead to a 1 : n operation correspondence between op_r^i and Seq_{result}. In this case, Seq_{result} has to be emptied so that its construction starts afresh for the next traversed invocation sequence in $Prot_o$. Otherwise, if the postconditions of op_r^i are completely satisfied, a 1 : n correspondence is created between the requested operation op_r^i and the matched sequence of offered operations Seq_{result}, which is returned as a valid result. In our example, after the traversal of $InSeq_o$, the found 1 : n operation correspondence for *makeReservation()* is $validateCredentials() \rightarrow giveDiscountVoucher() \rightarrow makeRoomReservation()$. The resulting 1 : n correspondence is returned by the `matchFurther()` function.

Step 11 In `SecondScenarioMatching()`, it is checked that if the returned correspondence from `matchFurther()` is a valid 1 : n operation correspondence, then the further traversal of $Prot_o$ is not required and has to be stopped.

Step 12 At the end of the algorithm, the determined correspondence is returned

$Corr$, which either contains a valid 1 : n operation correspondence for the requested operation under consideration op_r^i or it is *null* if no matching offered operation sequence in $Prot_o$ could be found.

The function `extendToEarlierOps()` is specified in Listing 7. It takes as input the offered operation under consideration op_o whose preconditions are not satisfied by the $PreCheck$, which is constructed gradually during 1 : n matching to satisfy the preconditions of the matched offered operations. Therefore, it takes the invocation sequence under consideration $InSeq_o$ as input and backtracks it from the offered operation op_{first} in $InSeq_o$ that represents the starting point of the matched operation sequence. As an output, it returns any possible operation sequence of the earlier operations $result$, whose preconditions can be combined with $PreCheck$ to satisfy the preconditions of op_o.

As an example consider Fig. 6.12, where the offered operation *bookRoom()* completely satisfies the postconditions of the requested operation *makeReservation()*. However, as the preconditions of *bookRoom()* are not satisfied by *makeReservation()*, so earlier operations in the offered invocation sequence have to be considered to satisfy the preconditions of *bookRoom()*. A step by step description of `extendToEarlierOps()` in the context of this example is as follows:

Step ① A variable Pre is initiated with $PreCheck$ to track the satisfied preconditions of of op_o. For the given example, there is only one operation in the matched sequence so op_o and op_{first} point to the same operation *bookRoom()*. $PreCheck$ at this point only contains the preconditions of the requested operation *makeReservation()*. Additionally, the backward traversal is started from the operation previous to op_{first} in the invocation sequence $InSeq_o$. For the given example, this traversal starts from *giveDiscountVoucher()*. $result$ represents the resulting operation sequence of the earlier operations that is built as a result of backward traversal.

Step ② This backward traversal is done to the point when either the preconditions of op_o are completely satisfied by the variable `Pre` or the start of the invocation sequence is reached. For every traversed operation op_x, Pre is updated with the postconditions of op_x using `combinePre` operator discussed earlier. Additionally, op_x is added to $result$. For the given example, Pre is updated to contain the newly created elements by *giveDiscountVoucher()*, i.e., objects of type `Voucher` and its link to `User`. Additionally, *giveDiscountVoucher()* is added to the

Listing 7: A Function that backtracks an offered invocation sequence $InSeq_o$ from a particular operation op_{first} to determine any earlier operations that can contribute to satisfy preconditions of an offered operation op_o under consideration

Input: Offered Operation under consideration op_o
Input: Operation from where the backtracking starts op_{first}
Input: variable used to track the satisfied preconditions so far $preCheck$
Input: The Offered Invocation Sequence under consideration $InSeq_o$
Output: A resulting operation sequence $result$ from $InSeq_o$

```
extendToEarlierOps(op_o, op_first, preCheck, InSeq_o)
    Pre = preCheck;     // Step ① op_x = InSeq_o.previousOp(op_first);
    result = null;
    while (! satisfies_pr(Pre , Pre(op_o))) ∧ (op_x != null) do   // Step ②
        Pre = combinePre(Pre, Post(op_x));
        result = op_x + result ;
        op_x = InSeq_o.previousOp(op_x);
    end
    if satisfies_pr(Pre , Pre(op_o)) then                          // Step ③
        matched = true;
        op = result.firstOp();
        for op=result.nextOp(op) do
            if ! satisfies_pr(Pre , Pre(op)) then                  // Step ④
                matched = false;
                Exit;
            end
        end
        if matched then                                            // Step ⑤
            preCheck = Pre;
            return result;
        end
    end
    else return null ;
end
```

resulting sequence *result*. The inclusion of *giveDiscountVoucher()* to *result* does not completely satisfy the preconditions of *bookRoom()* as an object of type `Facility` linked to `RoomPackage` is still not satisfied. Therefore, the $InSeq_o$ is further traversed backwards to include *selectFacility()* to *result*. At this point, the preconditions of *bookRoom()* are completely satisfied and hence the traversal is terminated.

Step ③ At the end of the traversal, if the preconditions of the offered operation under consideration op_o are completely satisfied by Pre, this means that *result* is a sequence of the operations earlier to op_{first} which can participate to satisfy the preconditions of a subsequent offered operation op_o in the potential 1 : n correspondence. However, it has to be checked that every operation in *result* is also invocable in the given circumstances, i.e., its preconditions are satisfied by Pre built so far. *matched* is initiated to notify if some operation in *result* is not invocable. In the given example, *result* contains the offered operation sequence $selectFacility() \rightarrow giveDiscountVoucher()$ so far. As the Pre at this point satisfies the preconditions of *bookRoom()*, now it has to be checked whether the preconditions of *selectFacility()* and *giveDiscountVoucher()* are also satisfied by Pre.

Step ④ *result* is traversed to check if the preconditions of every operation *op* are satisfied by Pre. If this is not the case for an *op*, *matched* is set to `false` and further traversal is stopped. This is because if an *op* in *result* is not invocable then this means that *result* is not an operation sequence that can participate in the 1 : n operation correspondence. For the given example, preconditions of *selectFacility()* and *giveDiscountVoucher()* are satisfied by Pre, which contains the objects `User` and `RoomPackage`.

Step ⑤ If *matched* is `true`, this means that *result* is a valid offered operation sequence as a result of backtracking from op_{first}. Hence, $PreCheck$ is updated with Pre and *result* is returned as a valid result of `extendToEarlierOps()`. Otherwise, a *null* is returned. In the given example, *result* containing $selectFacility() \rightarrow giveDiscountVoucher()$ is returned, which participate to satisfy the preconditions of *bookRoom()* in a potential 1 : n operation correspondence.

In addition to the 1 : 1, n : 1 and 1 : n operation correspondences discussed so far, there can be situations where a n : m operation correspondence may occur between the requested and offered operations. In the next section, we describe such n : m operation correspondences in detail.

6.3.4 n : m Operation Matching

A n : m operation correspondence between a requested and an offered operation sequence can be described as a correspondence where the operations in the offered operation sequence are invocable as their preconditions are satisfied. In turn, these offered operations satisfy the requirements specified in the sequence of the requested operations. Fig. 6.19 shows a diagrammatic representation of such a n : m operation correspondence between the requested operation sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$ and the offered operation $op_o^j \rightarrow \cdots \rightarrow op_o^l$.

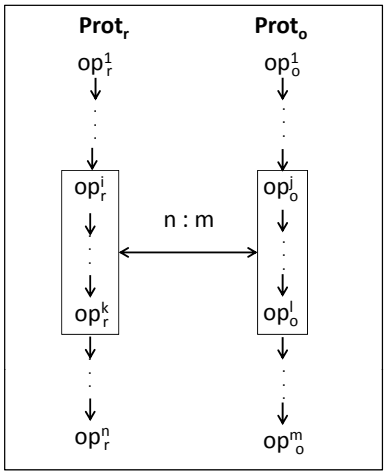

Figure 6.19: A diagrammatic Representation of a n : m Operation correspondence

Similar to 1 : n and n : 1 operation correspondences, there are also different scenarios where a n : m operation correspondence is possible. These different scenarios are discussed in the next section.

Different Scenarios for a n : m Operation Correspondence

Considering different types of scenarios for the n : m operation correspondences, we divide them into two broader categories, which are discussed as follows.

Basic n : m Operation Correspondence: This category includes the scenarios where a n : m operation correspondence results from an extension of the existing 1 : 1, 1 : n, and n : 1 operation correspondences. Different scenarios in this category are shown in Fig. 6.20. Each row in the

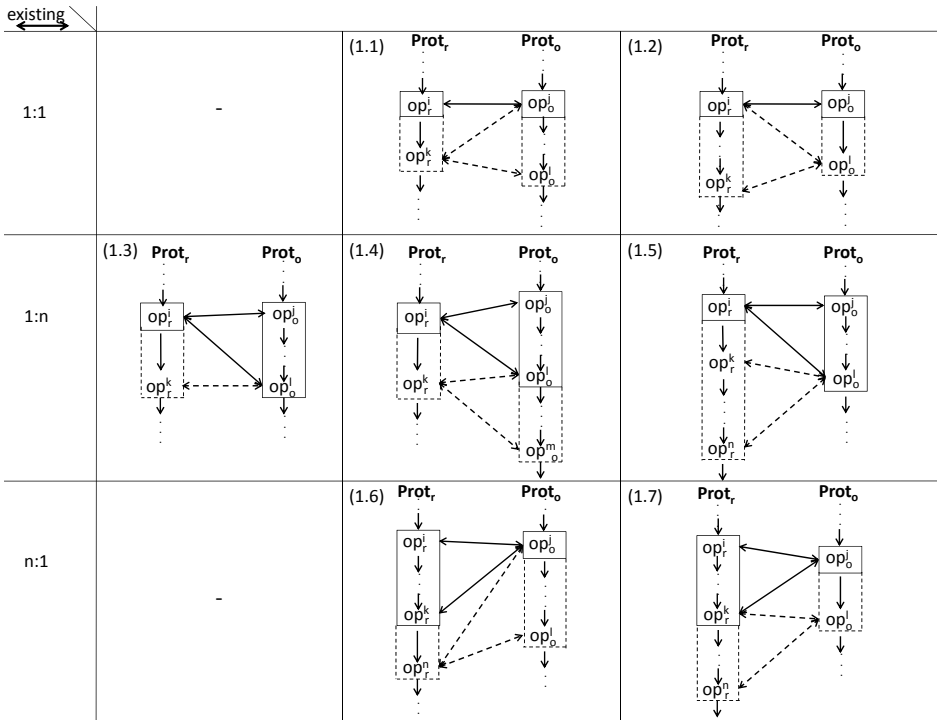

Figure 6.20: Different Scenarios of Basic n : m Operation Correspondences

table shows the different scenarios where an existing 1 : 1 , 1 : n, and n : 1 operation correspondence can be extended to a n : m operation correspondence, respectively. For instance, (1.1) depicts the scenario where initially a 1 : 1 operation correspondence is established between the requested operation op_r^i and the offered operation op_o^j. However, on further investigation it is determined that the postconditions of the next requested operation in the invocation sequence op_r^k are completely satisfied by the offered operation sequence $op_o^j \rightarrow \cdots \rightarrow op_o^l$. op_o^j from the existing correspondence participates in the extension as it may partially satisfy the postconditions of op_r^k or it may participate to satisfy the preconditions of some subsequent offered operation in the sequence. In this case, op_o^j is already invocable as its preconditions are satisfied in the existing 1 : 1 operation correspondence so it does not need to be checked again. For the rest of the offered operation in the sequence $op_o^{j+1} \rightarrow \cdots \rightarrow op_o^l$, there preconditions are satisfied by combining the preconditions of op_r^k with the postconditions of earlier operations in the offered sequence analogous to 1 : n operation matching.

Similarly, (1.4) depicts a situation where a 1 : n operation correspon-

dence is determined between op_r^i and $op_o^j \rightarrow \cdots \rightarrow op_o^l$. On further investigation, it is determined that some offered operations in the existing 1 : n correspondence, e.g., op_o^l in the shown diagram can further participate to satisfy the postconditions of the next requested operation op_r^k in the requested invocation sequence. Hence, the existing 1 : n operation correspondence is extended to a n : m operation correspondence between $op_r^i \rightarrow op_r^k$ and $op_o^j \rightarrow \cdots \rightarrow op_o^m$. In this case, the postconditions of op_r^k are satisfied by the offered operation sequence $op_o^l \rightarrow \cdots \rightarrow op_o^m$. Additionally, the preconditions of op_o^l are already satisfied in the existing 1 : n operation correspondence and the preconditions of the offered operations in the extended sequence $op_o^{l+1} \rightarrow \cdots \rightarrow op_o^m$ are satisfied by combining the preconditions of op_r^k with the postconditions of earlier operations in the sequence.

As an example, we consider the 1 : n operation correspondence shown in Fig. 6.16 between *makeReservation()* of HRS and $validateCredentials() \rightarrow giveDiscountVoucher() \rightarrow makeRoomReservation()$ of *HotelX*. On further investigation, it is determined that the postconditions of the subsequent requested operation *addFeature()* are also *partially satisfied* by *makeRoomReservation()*, i.e., a facility `fp:Facility` is created and linked to the reserved room package `rpp:RoomPackage`. The subsequent offered operation *reserveFacility()* satisfies the rest of the postconditions of *addFeature()*, i.e., the facility `fp:Facility` attached to the room package is also reserved and hence linked to `hbp:HotelBooking`. Consequently, the existing 1 : n operation correspondence can be extended to a n : m operation correspondence shown in Fig. 6.22 where the postconditions of the of the requested operation *addFeature()* are *completely satisfied* by the postconditions of the operations in the sequence $makeRoomReservation() \rightarrow reserveFacility()$. Moreover, the preconditions of $reserveFacility()$ are satisfied by combining the preconditions of *addFeature()* with the postconditions of $makeRoomReservation()$, whose newly created elements `fp:Facility` and its link to `hbp:HotelBooking` are reused in *reserveFacility()*.

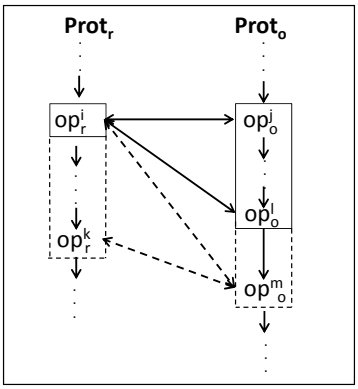

Figure 6.21: A Variant for Scenario 1.5 of basic n : m Operation Correspondence

In the scenario (1.1), (1.3), (1.4), (1.5), and (1.6) the extension of the

existing correspondence to an n : m correspondence arise from the fact that the offered operations in the existing correspondence participate to satisfy the postconditions of the subsequent requested operation. However, it is also possible that the requested operations in the existing correspondence may play a role to extend it to an n : m correspondence. Scenario (1.2), (1.7) and a variant of (1.5) shown in Fig. 6.21 cover such situations. For instance, in (1.2) an existing 1 : 1 operation correspondence is extended to an n : m operation correspondence because the preconditions of op_r^i and the subsequent requested operation sequence $op_r^{i+1} \rightarrow \cdots \rightarrow op_r^k$ can be combined to satisfy the preconditions of the subsequent offered operation op_o^l. In turn, op_o^l satisfies the postconditions of operations in $op_r^{i+1} \rightarrow \cdots \rightarrow op_r^k$. Same situation holds valid for Fig. 6.21, where an earlier 1 : n operation correspondence is extended to a n : m correspondence because the requested operation op_r^i participates to satisfy the preconditions of subsequent offered operation op_o^m, which satisfies the postconditions of subsequent requested operation sequence $op_r^{i+1} \rightarrow \cdots \rightarrow op_r^k$. Similarly, (1.7) depicts a similar case where some of the requested operations that are a part of the an existing n : 1 operation correspondence participate in the same manner.

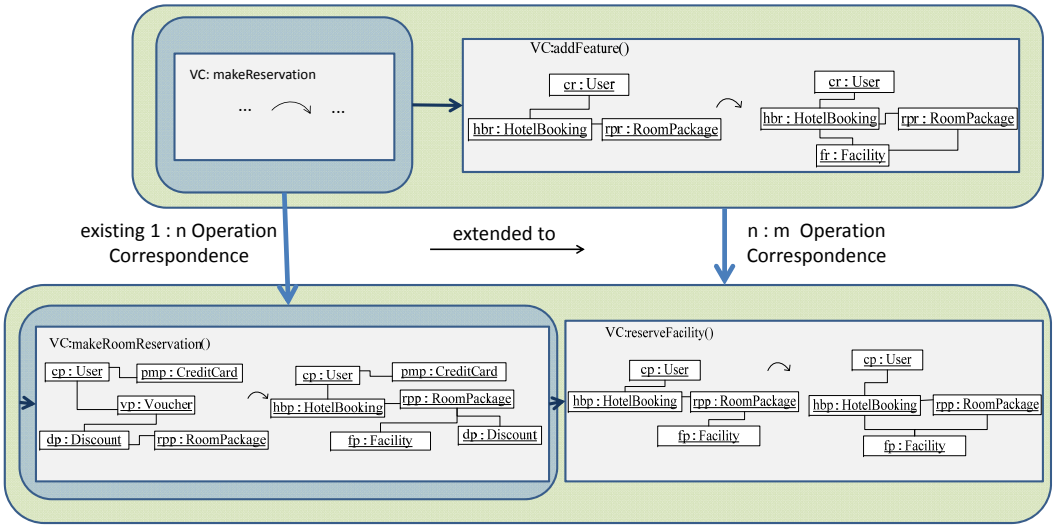

Figure 6.22: An Example for Scenario (1.4) of n : m Operation Correspondence

Fig. 6.23 gives an example for scenario (1.2). In this example, there is a 1 : 1 operation correspondence between the requested operation *makeReservation()* and the offered operation *makeRoomReservation()* as

it reserves a `RoomPackage` for a particular `User`. Additionally, *makeReservation()* also provides a `PaymentMode` for the `User`, which is not required by the offered operation *makeRoomReservation()*. However, next offered operation *sendNotification()* requires a `PaymentMode` for the `User`, which is satisfied by *makeReservation()*. In turn, *sendNotification()* satisfies the postconditions of the next requested operation *sendConfirmation()* by creating a `Notification` for the `HotelBooking` and also deleting the `Voucher` for the `User` as it is already consumed. As a result, a n : m operation correspondence is established.

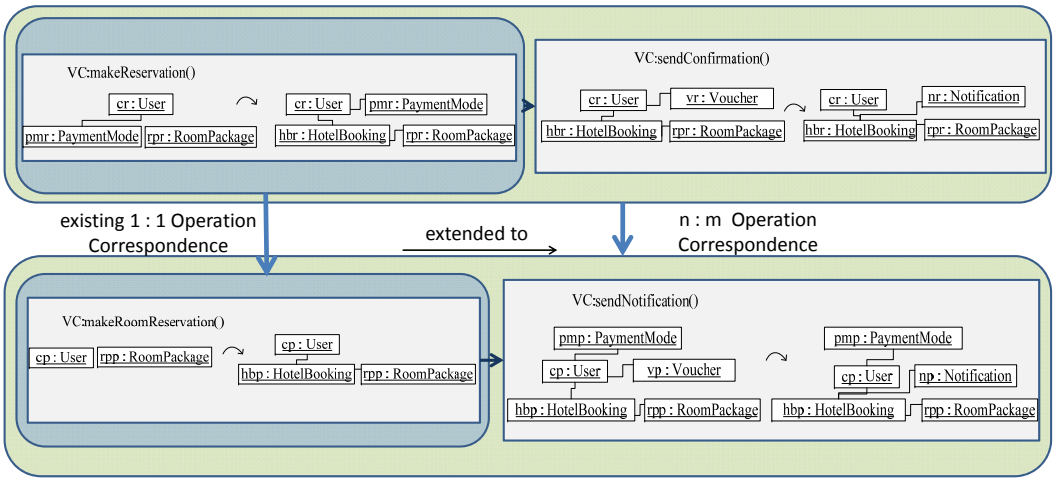

Figure 6.23: An Example for Scenario (1.2) of n : m Operation Correspondence

Apart from these basic scenarios of n : m operation correspondence, there are also some complex scenarios, which are discussed as follows.

Complex n : m Operation Correspondence: There are also certain scenarios for n : m operation correspondence where it does not arise due to an extension of existing operation correspondences. Rather in this case, it is not possible to have 1 : 1, 1 : n or n : 1 correspondences among the requested and offered operation, hence leading to n : m correspondence.

Fig. 6.24 shows different scenarios for such complex n : m correspondence. Scenario (2.1) shows a situation where the postconditions of the operations in the requested operation sequence $op_r^i \rightarrow \cdots \rightarrow op_r^k$ are satisfied by the offered operation op_o^l. However, in order to invoke op_o^l by satisfying its preconditions, some earlier offered operations in the invocation sequence, i.e., $op_o^j \rightarrow \cdots \rightarrow op_o^{l-1}$ have to participate. This leads to an n : m operation

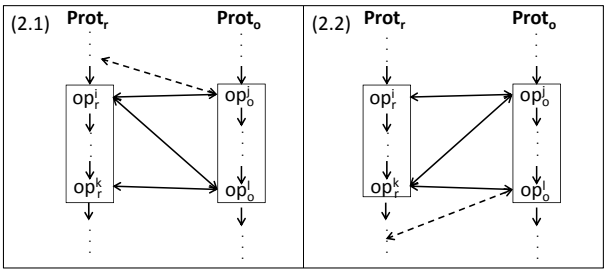

Figure 6.24: Different Scenarios for the complex n : m Operation Correspondences

correspondence between $op_r^i \rightarrow \cdots \rightarrow op_r^k$ and $op_o^j \rightarrow \cdots \rightarrow op_o^l$. The situation can be further complicated if some requested operations prior to op_r^i are included in the correspondence to satisfy preconditions of some operations in the matched offered operation sequence.

Scenario (2.2) shows a situation where the postconditions of a requested operation op_r^k are completely satisfied by the sequence of offered operations $op_o^j \rightarrow \cdots \rightarrow op_o^l$. However, some earlier requested operations in the sequence $op_r^i \rightarrow \cdots \rightarrow op_r^{k-1}$ have to participate in order to satisfy the preconditions of the offered operations in $op_o^j \rightarrow \cdots \rightarrow op_o^l$. This leads to a n : m operation correspondence between $op_r^i \rightarrow \cdots \rightarrow op_r^k$ and $op_o^j \rightarrow \cdots \rightarrow op_o^l$. In this case, the situation is further complicated if the postconditions of some subsequent requested operations are also satisfied by some offered operation in the sequence.

An example for scenario (2.1) is shown Fig. 6.25. The requester's requirements specified in the requested operation sequence *bookFlight()* → *getOnlineTicket()* are satisfied by the offered operation *bookFlight()*. However, this requested operation sequence does not satisfy the preconditions of the offered operation in return, i.e., a `FlightService` linked to `Flight` is not provided in the preconditions of these requested operations, which is required for the invocation of the offered operation *bookFlight()*. For this purpose, an earlier offered operation in the invocation sequence *selectService()* has to be included in the operation correspondence, which creates a `FlightService` and link it to `Flight`. This results in an n : m operation correspondence between *bookFlight()* → *getOnlineTicket()* and *selectService()* → *bookFlight()*.

The different scenarios of basic and complex n : m correspondences discussed in this section specify the primitive cases of n : m operation correspondence. Therefore, the possibility exists that different scenarios can be combined to have further complex scenarios for n : m correspondences. For

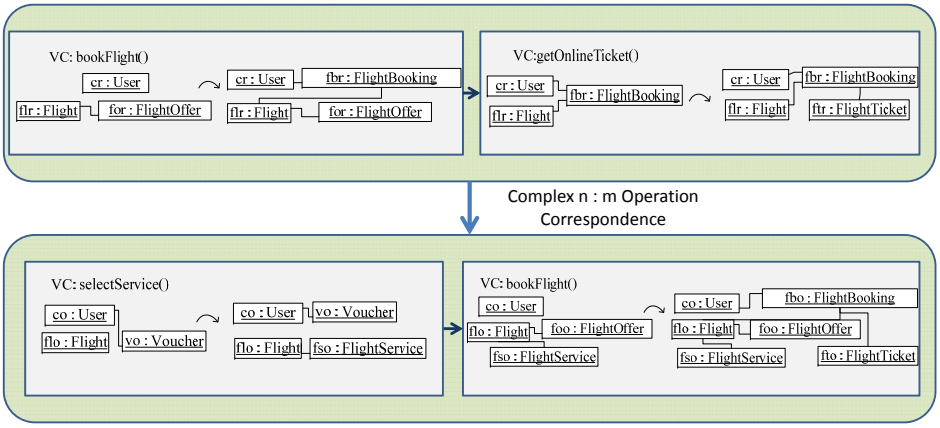

Figure 6.25: Example for a complex n : m Operation Correspondences

instance, some further scenarios can be determined if we consider an extension of an existing n : m operation correspondence in the similar fashion as shown for the basic n : m correspondences.

For the work in this thesis, we do not go into further details of the n : m operation correspondence, such as, the formalized structure definition and an n : m operation matching algorithm that encompasses different scenarios.

However, a closer look at the identified scenarios and examples for n : m operation correspondences make it clear that these correspondences are a combination of different features of the 1 : 1 , 1 : n, and n : 1 operation correspondences. For instance, in scenario (1.6), the n : m operation correspondence is an existing n : 1 correspondence, which is extended with a pattern similar to a 1 : n operation correspondence. Hence, in this case, the basic n : m operation correspondence combines the features of 1 : n and n : 1 operation correspondence.

Therefore, we claim that on the basis of our in-depth analysis of different scenarios for n : m operation correspondences and the proposed strategies for 1 : 1, 1 : n, and n : 1 operation matching, a formalization of the structure and definition of a matching algorithm for n : m operation matching in future must be achievable. For instance, an algorithm for an n : m operation matching catering to scenario (1.6) can result from combining the 1 : n and n : 1 operation matching algorithms.

In the next section, we describe our strategy to determine the set of all possible operation correspondences for a particular requester.

6.3.5 Operation Mapping Generation

On the basis of the proposed strategies for different kinds of operation correspondences, our operation matching mechanism allows to determine the *requester operation mapping* for a particular requester according to Def. 4, i.e., the set of all possible operation correspondences with the set of offers selected after the category matching.

Below, we discuss the proposed algorithm to generate the *operation mapping* for a particular requester in detail.

The algorithm for generating operation mapping is shown in Listing 8. It takes as input the request under consideration req and the set of offers that are selected in the *category matching* phase, i.e., $offers$. As a result, it outputs the set of all possible operation correspondences for the requester called *requester operation mapping* as $OpMap_r$. A detailed description of this algorithm is as follows:

Step ① Initially, $OpMap_r$ is initialized to be empty. $InSeq_r$ is the single invocation sequence comprising the requested protocol in req. In order to build $OpMap_r$ for req, $InSeq_r$ is traversed to match the requested operations in the given sequence with offered operations and hence find their possible operation correspondences.

Step ② Every offer o in $offers$ is considered to determine any possible operation correspondences for the requester. For this purpose, the offered protocol also has to be taken into account in case of 1 : n operation correspondences. Hence, $Prot_o$ represents the offered protocol in o determined through `protocol()`. For instance, in the given example, there are 4 offered services that are selected in the category matching phase, i.e., *HotelX*, *HotelY*, *FlightBooker*, and *PayOnline*. Therefore, in order to determine $OpMap_r$ for *HRS*, the operations of these 4 offered services have to be considered. In every iteration during the traversal of requested invocation sequence $InSeq_r$, the next *traversable* requested operation, which is not traversed and hence for which there is no correspondence determined so far is considered for matching.

Step ③ For any requested operation under consideration, the most preferred choice is to have a 1 : 1 operation correspondence with an offered operation. Therefore, for the current requested operation op_r under consideration, the first option is to search for any possible 1 : 1 operation correspondence from the operations in o determined through `operations()`.

Listing 8: Algorithm to determine Requester Operation Mapping $OpMap_r$ for a requester r with request req with available offers $offers$

```
Input: Request req
Input: Set of offers offers
Output: Requester Operation Mapping OpMap_r

generateOperationMapping(req , offers)
    OpMap_r = null;                                              // Step ①
    InSeq_r = req.InvSeq();
    for o=offers.next() do                                       // Step ②
        Prot_o = o.protocol();
        for op_r = InSeq_r.nextUnmatchedOp() do
            corr_i=oneToOneMatching(op_r , o.operations());      // Step ③
            if corr_i != null then                               // Step ④
                corr_j=nToOneMatching(op_r, InSeq_r, corr_i.offeredOp());
                if corr_j != null then OpMap_r.add(corr_j) ;
                else OpMap_r.add(corr_i) ;                       // Step ⑤
            end
            else
                op_o = null;
                op_o = op_x ∈ o.operations():
                compSatisfies_po(op_x , op_r);                   // Step ⑥
                corr_i=nToOneMatching(op_r, InSeq_r, op_o);
                if corr_i != null then OpMap_r.add(corr_i) ;
                else                                             // Step ⑦
                    corr_j=oneToNMatching(op_r, op_o, Prot_o);
                    if corr_j != null then OpMap_r.add(corr_j) ;
                end
            end
        end
    end
    return OpMap_r                                               // Step ⑧
end
```

Step ④ If a 1 : 1 operation correspondence is found for op_r, it is possible that it can be extended to a n : 1 operation correspondence. As described in scenario 1 for n : 1 operation correspondence, in this case the postconditions of some subsequent operations in the requested invocation sequence are also satisfied by the matched offered operation. For instance, in our running example, the requested operation *checkAvailability()* has a 1 : 1 correspondence to the offered operation *searchRoom()* of *HotelX* offer. On further investigation, this correspondence can be extended to a n : 1 operation correspondence between $checkAvailability() \rightarrow viewDetails()$ and *searchRoom()*. In case where such a n : 1 operation correspondence exists, it is added to the $OpMap_r$.

Step ⑤ If the determined 1 : 1 operation correspondence cannot be extended to a n : 1 operation correspondence, it is added to the $OpMap_r$.

Step ⑥ If a 1 : 1 correspondence is not found for op_r, the first possibility is that there is an offered operation op_o, which *completely satisfies* the postconditions of the requested operation under consideration but the preconditions of op_o are not satisfied by the requested operation op_r. In this case, one option is that some subsequent operations in the requested invocation sequence $InSeq_r$ can participate to satisfy the preconditions of op_o as described in scenario 2 for n : 1 operation correspondence. An example for such a n : 1 operation correspondence is shown in Fig. 6.9. Consequently, the resulting correspondence is added to $OpMap_r$.

Step ⑦ If a n : 1 operation correspondence is not possible in this case, then a 1 : n operation correspondence might be possible. A n : 1 operation is not possible because of two reasons: firstly, the preconditions of op_o are not satisfied by any subsequent operations in the requested invocation sequence. In such a case, a 1 : n operation correspondence according to scenario 1 might be possible where some operations earlier to op_o in the offered protocol $Prot_o$ participate to satisfy preconditions of op_o.

A n : 1 operation correspondence is also not possible, if no op_o exists that completely satisfies the postconditions of op_r. In such a case, there is a possibility of a 1 : n operation correspondence according to its scenario 2 and 3. For instance, in the running example, the postconditions of the requested operation *makeReservation()* are not satisfied by a single operation offered by *HotelX*. Instead, an offered operation

sequence $validateCredentials() \rightarrow giveDiscountVoucher() \rightarrow makeRoomReservation()$ has to be invoked to fulfill the requirements specified in *makeReservation()*. The resulting 1 : n operation correspondence is then added to $OpMap_r$.

1. At the end of this operation matching process, all possible operation correspondences for a requester's request with candidate offers are determined. The set of these correspondences, i.e., *requester operation mapping* $OpMap_r$ is returned as a result.

It is important to mention here that this algorithm can be further extended in future, when the algorithms for n : m operation matching are also defined. In this case, in each case where a 1 : 1, 1 : n and n : 1 operation correspondence is determined, it is checked whether it can be further extended to one of the scenarios of basic n : m operation correspondences. For instance, a determined 1 : n operation correspondence may also be extended to a n : m operation correspondence following scenario 1.3, 1.4, or 1.5 of the basic n : m operation correspondence. For the given example, the 1 : n operation correspondence between *makeReservation()* and $validateCredentials() \rightarrow giveDiscountVoucher() \rightarrow makeRoomReservation()$ can be extended to a n : m operation correspondence following scenario 1.4 as shown in Fig. 6.22.

Similarly, if a 1 : 1, n : 1 or 1 : n operation correspondence is not possible for a particular requested operation under consideration, then a complex n : m operation correspondence might be possible.

For our running example, the operation mapping $OpMap_r$ for HRS is shown in Fig. 6.26. It contains the mapping to 4 offered services that are selected in the category matching phase.

As we discussed earlier, based on the the recommendations and restrictions for RSDL service descriptions, our operation matching mechanism assume that the important parts of operation signatures, i.e., the input/output parameters are automatically matched while matching of their behavioral semantics. This means that the operation correspondences in the requester operation mapping assure that the requested and offered operations in an operation correspondence match on the basis of their structural as well as behavioral aspects.

However as mentioned earlier, there can be rare occasions when it is inevitable for the service partners to specify input/output parameters for the operations with primitive data types, i.e., XML data types as specified in RSDL specification. In such a case, an automatic matching of these primitive parameters is not possible through the proposed operation matching

Providers / Requestor HRS	HotelX	HotelY	FlightBooker	PayOnline
checkAvailability() → viewDetails()	searchRoom() **(n : 1)**	getAvailableRoom() **(n : 1)**	-	-
makeReservation() → addFeature()	validateCredentials() → giveDiscountVoucher() → makeRoomReservation() → reserveFacility() **(n : m)**	reserve() **(n : 1)**	-	-
getFlightOffer()	-	-	searchFlight() → giveOffer **(1 : n)**	-
bookFlight() → getOnlineTicket()	-	-	selectService() → bookFlight() **(n : m)**	-
makePayment()	-	-	-	signIn() → payDues()→ generateReceipt()→ signOut() **(1 : n)**

Figure 6.26: Operation Mapping for HRS Service Request

algorithms as these algorithms only match the objects typed over the non-primitive types from the data model and links in the visual contracts. In this situation, it is required that these primitive parameters in the requested and the offered operation signatures are separately matched. To handle this situation, our approach relies on a rather simple mechanism at the moment.

After the requester operation mapping is generated on the basis of the given operation matching algorithms, our mechanism performs a signature matching for each determined operation correspondence. In this direction, it is checked that for every primitive input parameter of the offered operation(s) in the correspondence, there is a corresponding input parameter of the same type in the comprising requested operation(s). Similarly, for a primitive output parameter of the requested operation(s) in the correspondence, there is a corresponding output parameter of the same type in the offered operation(s). If this does not hold true, then the operation signature matching for the particular operation correspondence is not successful. However, in such a case, our approach does not directly discard the particular operation correspondence because we claim that our elaborate operation matching mechanism based on their behavioral semantics present a more reliable and stronger notion of similarity. Hence, an operation correspondence cannot be discarded directly on the basis of its operation signature

mismatch. The requester in this case is notified about the signature mismatch, who in turn might consider to make some suitable changes to the requested operation signatures accordingly. In other case, despite the operation signature mismatch, the requester may still decide to maintain the operation correspondence and proceed with it to the next stage of service description matching. Later, after negotiation with the particular service provider, there might be a possibility to resolve the signature mismatch.

We claim that a rather elaborate operation signature matching mechanism for the primitive parameters in the requested and offered operations can be introduced in future by doing some extension to the existing operation matching mechanism. Firstly, so far we only match these primitive parameters on the basis of their data types. However, a semantic matching can be enabled by defining the ontological semantics for such primitive parameters. Secondly, the operation matching algorithms that so far only rely on the matching of the objects and links in the requested and offered visual contracts can be extended to also consider the attributes during matching process. In case of such an extension, there is a possibility that the operation parameters with primitive types can be mapped to the attributes in the visual contracts and hence they can also be automatically matched while matching of the requested and offered visual contracts.

After the requester operation mapping is achieved for a particular requester, the service discovery phase is complete and the framework can proceed to the service composition phase where the protocols in the request and selected offers are matched and possible service compositions are determined.

6.4 Summary and Discussion

In this chapter, we presented our multi-level service discovery approach, which takes the normalized service request as input and returns all its possible operation correspondences with the normalized service offers available on the service market. As a result of this phase, a subset of available service offers is determined, which is used to determine any possible service compositions satisfying the service request through protocol matching in the next phase of the proposed approach. The multi-level approach enables a gradual refinement of the resulting set of selected service offers.

Our service discovery approach comprise of two levels, namely, *category matching* and *operation matching* between the request and the offers. As category matching is not the focus of our work, we have briefly describes

how OTF provider develops and maintains a categorization hierarchy to categorize the service request and the published offers in suitable categories. Later, this hierarchy is used to match the service requests and offers on the basis of the assigned categories.

For the operation matching level, we have proposed an operation matching mechanism based on the identified requirements, which allows an elaborate matching of the requested and offered operation in terms of their structural as well as behavioral specification. Additionally, it acknowledges the granularity level heterogeneity, which can lead to complex operation correspondences, such as, 1 : n, n : 1, and n : m correspondences between them. In this direction, we identified different scenarios for each type of correspondence leading to its precise structure definition. Later, we defined a set of operation matching algorithms to determine these different types of correspondences between the requested and offered operations by matching their visual contracts. Later, these algorithms are used to generate the *requester operation mapping* comprising all possible operation correspondences for a particular request with the available offers. As a result, the service discovery phase ends with the requester operation mapping and a subset of available service offers, which have operation correspondences with the service request under consideration.

As the service descriptions are defined manually by the service partners in their independent domains, there are chances that incomplete or incorrect service descriptions can lead to incorrect operation matching results. Therefore, our approach allows the service requester to review the results and make desired modifications accordingly, e.g., some service offers in the result set may be manually discarded by the requester based on their non-functional aspects or to select certain operation correspondences, the requester might make changes to its request (,e.g., n : 1 operation correspondence, scenario 2). In future, we aim to develop an elaborate feedback mechanism that can guide the requester through different causes of operation mismatches and can provide suggestions to improve the results.

As mentioned earlier, the accuracy of the proposed mechanism can be further enhanced in future by also taking other elements of the visual contracts in account, such as, negative and positive application condition, attributes, etc. This can also support a more reliable operation signature matching mechanism if the operation parameters with primitive data types are restricted to be a part of the respective visual contract as attributes. Consequently, the matching of attributes in visual contract will implicitly match those primitive parameters in operation signatures as well, which are not elaborately matched so far. Similarly, for the simplification of the sit-

uation at hand, our operation matching algorithms so far do not deal with multiple objects of a particular data type in a service description. In order to deal with multiple objects of a data type, it has to be investigated that how the notion of element correspondence is affected in this case. Similarly, the cardinality of the multi-objects also have to be considered while matching the visual contracts.

Based on the results of the service discovery phase, the service composition phase can be initiated, which is the topic of the next chapter.

7

Service Composition

After constructing the *requester operation mapping* as a result of the service discovery phase, possible service compositions are determined in the next phase, i.e., *service composition*. As input, it takes the *requester operation mapping*. During the service composition phase, the service protocols of the request and the selected offers are translated to a common semantic domain. These translated service protocols are matched and composed using a composition operator and the resulting composition is analyzed to determine any possible service compositions satisfying the service request. The output of this phase is either a set of candidate service compositions satisfying the service request or a failure notification with feedback about mismatches.

A detailed overview of this phase is given in Fig. 7.1, which will be discussed in detail in the following section.

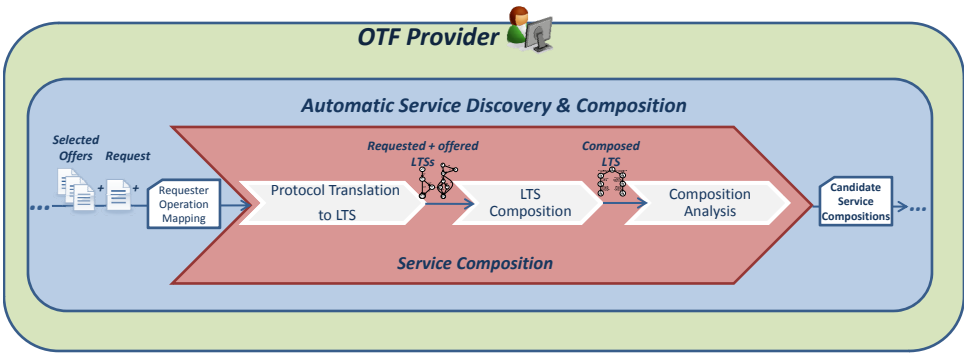

Figure 7.1: An Overview of the Service Composition Phase in the Proposed Approach

7.1 Service Composition Overview

During the service composition phase, our approach relies on the matching of the participating service protocols to determine any possible service compositions.

In this direction, our first concern is that the mechanism should not only allow a 1 : 1 match between the requested and an offered protocol rather it should also enable a service composition through protocol composition leading to 1 : n matching between the requested and offered protocols.

The second important concern is to resolve the linguistic heterogeneity of the requested and the offered protocols. As mentioned in Chap. 1, SOC enables the service partners to function in their independent domains and one of the consequences of this independence is their linguistic heterogeneity. In this context, as our approach deals with RSDL-based service descriptions, the linguistic heterogeneity resolution is particularly important for the protocol composition phase. This is because the requested and offered service protocols are specified using different sub-languages of RSDL and in order to match and compose these protocols, it is important that their linguistic differences are overcome and they are brought to a common representation.

Another important concern in this direction is that our operation matching mechanism introduced in Chap. 6 ensure accurate service discovery by matching the operations on the basis of their different aspects and result in complex 1 : n , n : 1 and n : m operation correspondences. As discussed in Chap. 2, it is important to have a combinatorial approach for the matching of comprehensive service descriptions where the contents and matching results of one aspect contributes to the matching of another aspect. Hence, realizing such a combinatorial approach, the complex operation correspondences determined earlier must be taken into account while matching the protocols in the proposed approach.

From the discussion so far, we derived the requirements that must be fulfilled by an elaborate protocol composition mechanism for RSDL service descriptions. These requirements are as follows:

R1 In addition to 1 : 1 matching between the requested and offered protocols, it must allow a 1 : n protocol matching to determine possible service compositions.

R2 It must be able to resolve the linguistic heterogeneity of the requested and offered service protocols.

R3 It must allow the matching of the involved service protocols based

on complex operation correspondences resulting from the matching of different aspects of the requested and offered operations.

In this direction, first we analyze the state of the art in the protocol matching area to determine whether any existing approach satisfies the requirements mentioned above.

7.1.1 Protocol Matching - State of the Art

Protocol Matching has been a topic of extensive research in CBSE in past decades [166, 2, 139]. Its main focus is to enable communication among components by checking the conformance between their behavioral descriptions specified as protocols. With the vast body of work in this direction, we restrict our discussion to some salient approaches that we consider to be relevant for our work. In this context, approaches like [92, 50, 93, 105, 142, 24, 138] particularly deal with the protocol conformance of UML-based protocols. [92, 93] aim at defining an elaborate methodology for consistency checking among UML-based behavioral models of a system. In this direction, [92] conceptualizes the consistency between a sequence diagram that models the overall system and multiple statemachine diagrams modeling the behavior of each individual component in a real-time systems. For this purpose, it relies on $1:1$ operation correspondences, which are used to match the behavior of every individual object in the sequence diagram to its respective statemachine diagram. Additionally, this approach also conceptualizes the notion of temporal consistency for real-time systems. However, it deals with the problem on a conceptual level where the concrete details, such as, automatic consistency checking, language heterogeneity, etc. are missing. In their later work [50, 93], they further enhance their methodology by translation of sequence diagrams and statemachines to the process algebra CSP [71] to enable their automatic consistency checking. For the resulting processes, consistency checks are defined as CSP assertions which are automatically verified by a model checker. Unlike [50], which checks the consistency of communication patterns between two independent components, Moffet et al. [105] analyze the behavior of a single component in isolation. For this purpose, they define a $1:n$ protocol matching mechanism where the behavior of a component modeled as a single behavioral statemachine (BSM) is matched to the protocol specification of its ports specified as protocol statemachines (PSMs). As all the involved statemachines share the same set of messages, their matching is based on a $1:1$ correspondence between these messages. The approach translates the involved BSM and

PSMs to their formal counterparts, i.e., finite state automata (FSA), which are later composed to be checked for certain safety and liveness properties. [142] deals with the same problem from another perspective where the conformance among a BSM and PSMs is formally defined and a methodology is presented to construct such conformant BSM for existing PSMs. However, the question of how to match the existing BSM to PSMs is not discussed. [24] allows a 1 : n protocol matching among the components in mechatronics systems. In this direction, the behavior of individual components is specified through real-time statecharts (RTSCs) using UML profile for mechatronic systems *MechatronicUML*. It is assumed that a 1 : 1 communication protocol already exists between two components based on the conformance between their RTSCs. Such a 1 : 1 communication between two components is later extended to a 1 : n communication between a single component and multiple components of same type. For this purpose, RTSCs of the involved components are composed automatically using 1 : 1 communication protocol and is verified using UPPAAL[1]. [138] presents an approach similar to [92, 93] where the possible interactions of the components specified as UML collaboration diagrams are matched with the individual behavior of the components specified as statemachines. For this purpose, the statemachines and collaboration diagrams are translated to a PROMELA model and Büchi automata, respectively and Spin model checker is used to verify the model against the automata. All these protocol matching approaches deal with the protocol matching concept in different contexts and allow 1 : n, n : 1 and n : m matching between protocols. Additionally, the problem of linguistic heterogeneity is also indirectly solved by translating the protocols to formal semantic domains. However, most of these approaches assume a closed environment where the components share the same set of messages leading to 1 : 1 operation correspondences. Hence, complex operation correspondences are not considered.

Particularly in the context of SOC, protocol matching has been investigated in detail as well. For instance, approaches like [106, 128, 29, 145, 124, 125, 107, 59, 37, 38] propose mechanisms to match the protocols for different tasks, such as, service discovery, service composition, service replacement, etc. Some of the relevant approaches have already been introduced in Sec. 2.4.

Each of these approaches focus a certain aspect of the problem at hand. For instance, [124, 125] allows service composition on the basis of 1 : n matching among a requested and multiple offered service protocols specified

[1] http://www.uppaal.org/

as labeled transition systems (LTSs). However, this protocol matching is based on 1 : 1 operation correspondences between the requested and offered operations. Similarly, fairly large number of 1 : 1 protocol matching approaches [106, 128, 145, 37, 38] rely on such simple operation correspondences for this purpose. A step further in this direction is taken by approaches like [59, 29] that support a 1 : 1 matching between a requested and an offered protocol on the basis of complex operation correspondences. [59] claims to determine 1 : n and n : 1 operation correspondences based on the matching and consequent splitting/merging of structural elements, such as, input/output parameters. On the other hand, [29] additionally claims to match behavioral semantics specified as source and target states for the transitions in the matched LTSs, however the notion of matching states in not elaborately defined.

As far as the resolution of linguistic heterogeneity for the service protocols is concerned, only [145] is relevant. It considers the protocols specified in different languages and translates them to a common semantic domain before matching.

From this study of the state of the art in the area of protocol matching, it becomes evident that none of the approaches *completely* fulfills our requirements for a suitable protocol matching mechanism. Therefore, in the next section, we introduce our protocol matching mechanism which aims at meeting these requirements.

7.1.2 Our Approach

Contrary to the existing work in the protocol matching and composition area, our service composition approach successfully fulfills the requirements specified earlier. As shown in Fig. 7.1, it takes the *requester operation mapping* generated by the service discovery phase as input. On the basis of this input, a set of service offers that are candidate to participate in the possible service compositions are identified from the available offers in the repository. As mentioned earlier, the main task during this phase is the composition of the service protocols in the service request and the candidate service offers (Satisfying R1). For this purpose, their linguistic heterogeneity needs to be resolved. Hence, in the first step, the requested and the offered service protocols are translated to a common semantic domain, i.e., labeled transition system (LTS) (Satisfying R2). In the next step, the LTSs corresponding to the participating service protocols are matched and composed using our *LTS composition operator*, which utilizes the requester operation mapping from the service discovery phase for this purpose (Satisfying R3). In the last

step, the composed LTS is checked for any possible service compositions that satisfy the service request under consideration.

In the following, we discuss these steps of the our approach in detail.

7.2 Protocol Translation to LTS

Based on the requester operation mappings, our approach selects a subset of the services available on the service market further reducing the result set of the category matching step. For instance, on the basis of the operation mapping for HRS, i.e., $OpMap_{hrs}$, the set of selected service offers comprise of two hotel booking services namely *HotelX* and *HotelY*, a flight booking service *FlightBooker* and an online payment service *PayOnline*. For the rest of this chapter, the service offers of these services are identified by their respective names.

The requested and the offered service protocol for HRS and HotelX is already shown in Fig. 3.8 and Fig. 3.9. The service protocols of the other service offers are shown in Fig. 7.2.

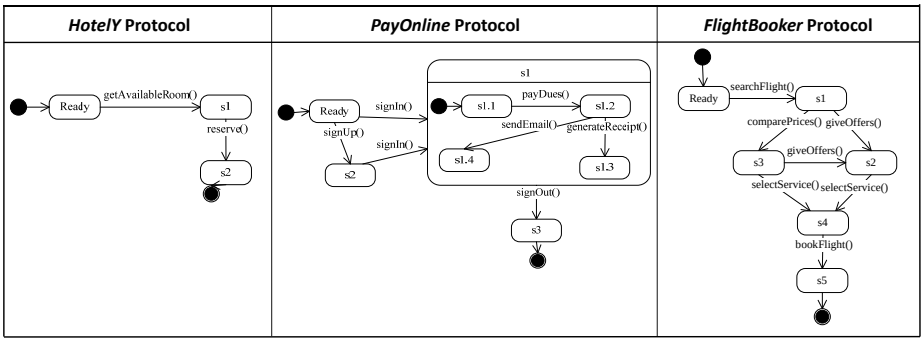

Figure 7.2: The Service Protocols of the selected Service Offers

To resolve the linguistic heterogeneity, our approach translates these service protocols *automatically* to their corresponding labeled transition systems (LTSs). For this purpose, we rely on DMM-based RSDL semantics for service protocols specified in Chap. 4. These semantics are specified in terms of DMM rules (graph transformation rules) that are typed over the runtime metamodel of the UML sequence diagram/UML statemachine diagram and can be applied for automatic translation to the corresponding LTSs.

Here it is important to mention that an automatic application of DMM rules through a graph transformation system results in a very extensive and

relatively big LTS. For instance, in a resulting LTS for UML sequence diagram, between the transitions and corresponding states that represent the invocation of messages on the lifelines, there are other transitions and corresponding states that simulate the processing of the runtime meta model elements like sorter and active message marker, etc. These are meant to track the execution, e.g., determination of the active lifeline, sequencing of the messages, assignment of the marker to the active message, etc. Similarly, for the UML statemachine diagram also, there are transitions and corresponding states in the resulting LTS that deal with the execution details, such as, initialization of the statemachine, the processing of marker tracking the execution of statemachine transitions at runtime, etc. Fig.. 7.3 shows such a blown-up LTSs for the sequence diagram and one of the statemachine diagram in the running example.

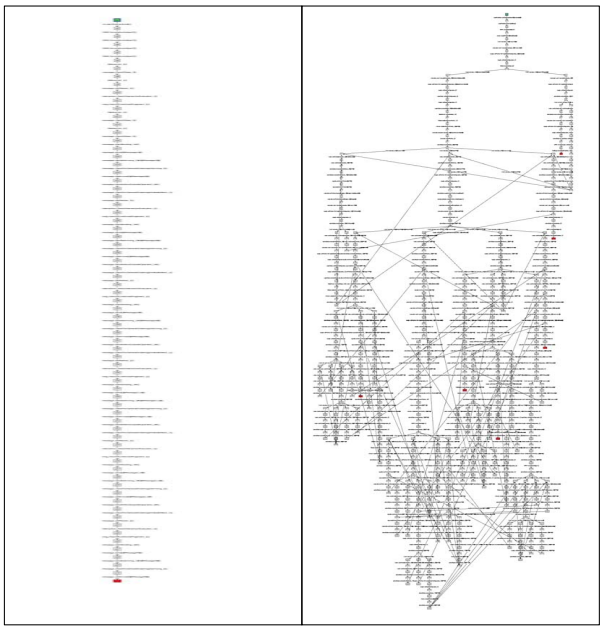

Figure 7.3: The automatically generated LTSs for the running Example using DMM

However, for our LTS composition mechanism, these particular execution details at runtime are not relevant as we are mainly concerned with those parts of LTSs that are concerned with the request/offered operation invocation, i.e., message invocation and transition firing in the sequence diagram and statemachine diagram, respectively. Therefore, a mechanism is

required that can simplify the obtained LTS by removing these irrelevant execution details without changing the actual semantics necessary for our LTS composition mechanism.

For the similar purpose, [19, 54] proposes an algorithm that performs such a simplification of the LTS obtained through the application of DMM rules for UML activity diagram. In this case, the LTS is simplified by removing the elements concerned with the runtime execution details, e.g., initialization of the activity diagram, processing of token flow, etc. and the resulting simplified LTS comprises only of transitions and corresponding states concerned with the execution of actions in the UML activity diagram. We claim that with some modifications, this algorithm can be used to obtain such a simplified LTS for the UML sequence diagram and UML statemachine diagram in our case. Here, we do not indulge in further details of such a mechanism and refer the reader to [19] for a detailed account of different technical aspects of this algorithm, such as, the traversal mechanism, the removal of irrelevant states and transitions, re-labeling of transitions, etc.

After such a simplification, the resulting LTS preserves the initial and final states, the transitions and states representing the branching structure, the invocation of messages and transition firing in the sequence digram and statemachine diagram, respectively. Other details in LTS that are redundant for our LTS composition mechanism, such as, the initialization, marker setting, etc. are removed during this simplification. Such simplified LTSs of the requester, i.e., HRS and the selected service offers in our running example are shown in Fig. 7.4.

A formal definition of such a LTS representing a service protocol is as follows:

Definition 9 (Requested/Offered LTS)**.** *Given a RSDL service description, i.e., a request/offer desc, its corresponding LTS* lts_{desc} *is a 5-tuple* (S, s_0, S_F, A, δ) *where:*

(i) S *is the set of states;*

(ii) $s_0 \in S$ *represents the initial state;*

(iii) $S_F \subseteq S$ *represents the set of final states, i.e., the state(s) corresponding to the state after the last operation invocation or the final state in the requested or offered protocol, respectively.*

(iv) A *is the set of actions corresponding to the requested/offered operations in desc.*

(v) $\delta \subseteq S \times OP \times S$ *is the transition relation and is* deterministic.

For further explanation, we will use the notation $desc.S$, $desc.s_0$, $desc.S_F$, $desc.OP$, *and* $desc.\delta$ *to refer to the elements of* lts_{desc}. □

Def. 9 specifies different elements of the simplified LTS corresponding to the requested or offered service protocol. One thing that is important to mention here is that although in a conventional LTS every state is an accepting state, the final state defined in Def. 9 is a special state, which represents the successful completion of the requested/offered usecase. As the LTSs in our approach correspond to a UML sequence diagram or a UML statemachine diagram with a final state, it is straightforward to determine such a final state in the corresponding LTS. As mentioned in Def. 9 (iii), the final state for a requested LTS represents the state achieved after the last operation invocation specified in the corresponding sequence diagram. Similarly, for the offered LTS, the final state corresponds to the final state of the corresponding statemachine diagram.

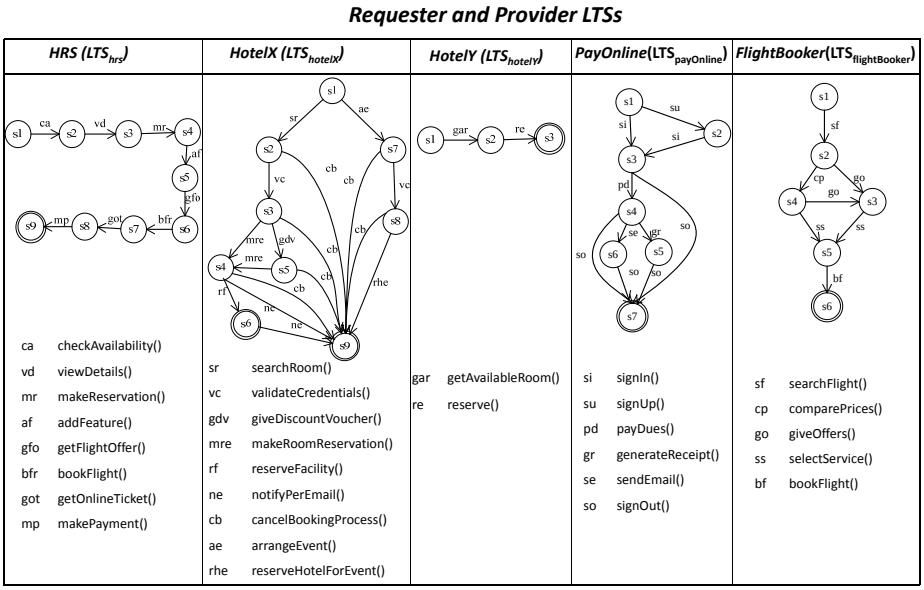

Figure 7.4: The simplified LTSs for the requested and offered Service Protocols in the running Example

LTS as the common semantic domain for the service protocols makes them machine-interpretable and make their automatic analysis possible. This further supports the automation of the composition process carried out in the next step.

7.3 LTS Composition

In order to determine possible service compositions, our approach uses a parallel LTS composition mechanism.

The parallel composition operation represented as $\|$ is already a known concept in process languages like CCS, CSP, LTS, Petri Nets, etc. In a distributed environment, these process languages are used to represent the behavior of the distributed components. Further, an overall system is defined on the basis of the interaction of these distributed components and its behavior is based on the parallel composition of the individual processes of these participating components. The basic idea of parallel composition is that it uses *shared* actions of the participating processes (the actions that are common in their action set) in order to synchronize their execution. This means that a shared action is executed by all the processes at the same time, while the unshared actions are executed in an interleaving manner.

Based on this conventional idea of parallel composition of processes, we have come up with a *specialized* parallel composition operator for the requested and offered service protocols based on the operation correspondences in requester operation mapping. In the following, we formally define our operation mapping-based LTS composition operator and describe it in detail.

7.3.1 Operation Mapping-based LTS Composition

In the conventional parallel composition, the transitions in the composed LTS are a result of parallel and interleaving transitions of the participating LTSs based on shared and unshared actions, respectively. With similar conception, we define an LTS composition mechanism where the parallel transitions are invoked in the participating service protocols on the basis of the operation correspondences between the request and the offers.

In the given scenario, requester operation mapping comprises possible complex operation correspondences. In correlation to the conventional LTS composition, the operation sequences in these correspondences can be understood as the *shared* actions of the participating LTSs and can be invoked in parallel in the respective LTSs resulting into a transition in the composed LTS.

Fig. 7.5 gives a closer insight into our LTS composition mechanism, which performs a *parallel composition* of all the participating LTSs, i.e., lts_r, lts_o^1, ..., lts_o^k on the basis of the requester operation mapping $OpMap_r$. We use the term *overlapping parts* for the parts of the participating LTSs that

correspond on the basis of an operation correspondence. This will be later explained in detail with examples. Consequently, our composition mechanism aims at composing these overlapping parts of the participating LTSs.

In correlation to the conventional LTS composition, those requested and offered operations that are not part of any operation correspondence are termed as the *unshared* actions. In our composition mechanism, such unshared actions are not considered in the composed LTS. This is a salient feature of our composition mechanism called as *selective composition* strategy which only composes selective parts of the participating LTSs where the selection is moderated through the requester LTS. This means that only those parts of the offered LTSs are considered that are relevant for the requester, i.e., that *overlap* with the requested LTS based on the identified operation correspondences. In comparison to a conventionally composed LTS in this case with the unshared actions as well, the selectively composed LTS with the proposed mechanism is smaller in size and hence easier to analyze.

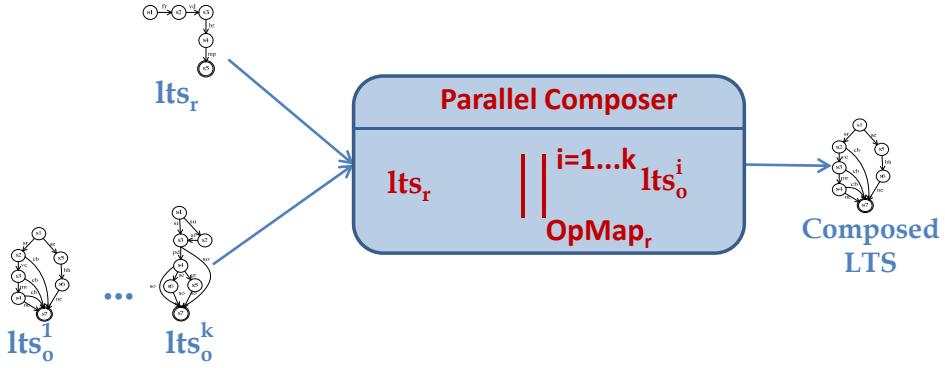

Figure 7.5: An Insight into the LTS Composition Mechanism

The resulting composed LTS can be formally defined as follows:

Definition 10 (Operation Mapping-based Composed LTS)**.** *Given the following:*

- *A request r with* operation mapping *$OpMap_r$;*
- *The requested LTS lts_r according to Def. 9;*
- *The offered LTSs lts_o^1, ..., lts_o^k according to Def. 9 for the selected offers o_1, ..., o_k, respectively.*

The composed LTS $lts_{comp} = lts_r \parallel_{OpMap_r}^{i=1,\ldots,k} lts_o^i$. *It is a 5-tuple* (S, s_0, S_f, A, δ) *where:*

(i) The set of composed states is given as $S = r.S \times o_1.S \times \ldots o_k.S$. *Hence, a composed state* $s_c \in S \wedge s_c = (s_r, s_{o1}, \ldots, s_{ok})$ *where* $s_r \in r.S$, $s_o^1 \in o_1.S$, $\ldots$,$s_o^k \in o_k.S$. *This means that a composed state is represented as a vector of its comprising states from the participating LTSs;*

(ii) The initial state is given as $s_0 \in S \wedge s_0 = (r.s_0, o_1.s_0, \ldots, o_k.s_0)$. *This means that the initial state in the composed LTS is a composition of the initial states of all the participating LTSs;*

(iii) The set of final states is given as $S_f \subseteq S \wedge S_f = r.S_f \times o_1.S \times \ldots o_k.S$. *This means that the set of final states in the composed LTS comprises of the states where the requested LTS* lts_r *is in its final state and the offered LTSs* lts_o^1, $\ldots$, lts_o^k *can be in any state;*

(iv) The set of possible actions is given as $A = seq_r||seq_o : (seq_r, seq_o) \in OpMap_r$. *This means that the set of possible actions for the composed LTS is based on the requester operation mapping* $OpMap_r$. *An action is defined as a parallel invocation of the requested and offered operation sequences comprising an operation correspondence in* $OpMap_r$.

(v) The transition relation is given as $\delta = S \times A \times S$. *With composed states* $s_c^i = (s_r^i, s_{o1}^i, \ldots, s_{ok}^i)$, $s_c^j = (s_r^j, s_{o1}^j, \ldots, s_{ok}^j)$, *and an action* $a = seq_r||seq_o$, *a composed transition* $s_c^i \xrightarrow{a} s_c^j$ *is a composition of the parallel transitions* $\delta_r \subseteq r.\delta$ *and* $\delta_{ox} \subseteq o_x.\delta : o_x \in \{o_1, \ldots, o_k\}$, *where:*

- *Given* $seq_r = op_{r1} \rightarrow \ldots \rightarrow op_{rn}$, δ_r *is the sequence of transitions* $s_r^i \xrightarrow{op_{r1}} s_r^{i+1}, \ldots, s_r^{j-1} \xrightarrow{op_{rn}} s_r^j$ *in* lts_r *resulting from the invocation of* seq_r.
- *Given* $seq_o = op_{o1} \rightarrow \ldots \rightarrow op_{om}$, δ_{ox} *is the sequence of transitions* $s_{ox}^i \xrightarrow{op_{o1}} s_{ox}^{i+1}, \ldots, s_{ox}^{j-1} \xrightarrow{op_{om}} s_{ox}^j$ *in* lts_{ox} *resulting from the invocation of* seq_o.
- $\forall o \in \{o_1, \ldots, o_k\} \setminus \{o_x\} : s_o^i = s_o^j$

Similar to Def. 9, the elements of lts_{comp} *can be referred as* $comp.S$, $comp.s_0$, $comp.S_f$, $compo.A$, *and* $comp.\delta$. □

Def. 10 specifies the composed LTS resulting from the *parallel composition* of the participating LTSs based on the requester operation mapping. We explain this with the help of an example shown in Fig. 7.6(a), which is an excerpt of the composed LTS for our running example. Conforming to (i) in Def. 10, every state in this composed LTS is a composition of the states of the participating LTSs. Similarly, as specified in (ii) in Def. 10, $cs1$ in the composed LTS is the initial state, which is a composition of all the participating initial states. According to (iii) in Def. 10, the composed LTS is in its final state if the requester LTS lts_r is in its final state notifying a possible service composition. This notion of final state and possible service composition will be discussed in more detail in Sec. 7.4.

According to Def. 10(iv), the set of possible actions in the composed LTS comprises of the requested and offered operation sequences in an operation correspondence invoked in parallel. Hence, the parallel invocation of the operation sequences in ($hrs.checkAvailability() \rightarrow hrs.viewDetails()$, $hotelX.searchRoom()$) in their respective LTSs LTS_{hrs} and LTS_{hotelX} forms an action for the composed LTS. The invocation of such an action results in a composed transition in the composed LTS, which is a composition of the parallel transitions in the participating LTSs. For instance, the operation sequences mentioned above can be invoked in the participating LTSs from their respective states in $cs1$. This results in a composed transition from $cs1$ to $cs7$. In this case, $cs7$ represents a composed state, where the participating LTSs, i.e., LTS_{hrs} and LTS_{hotelX} are in their respective states $hrs.s3$ and $hotelX.s2$, respectively after the parallel invocation of the operation sequences. The states of the other participating LTSs remain unchanged.

As a result of the parallel invocation of operation sequences, the invoked transitions of LTS_{hrs} and LTS_{hotelX} are shown in Fig. 7.6(b). We call these the *overlapping* parts of the participating LTSs.

Def. 10 defines the structure of the operation mapping-based composed LTS in the given approach. However, an important question in this direction is that how such a composed LTS can be built from scratch for the participating LTSs. For this purpose, a concrete strategy is required and keeping this requirement is mind, we devised an algorithm that gradually builds such a composed LTS for the participating LTSs on the basis of their operation correspondences. In the next section, we present our algorithm to construct the composed LTS conforming to Def. 10. It also provides a more detailed insight into Def. 10 based on the running example.

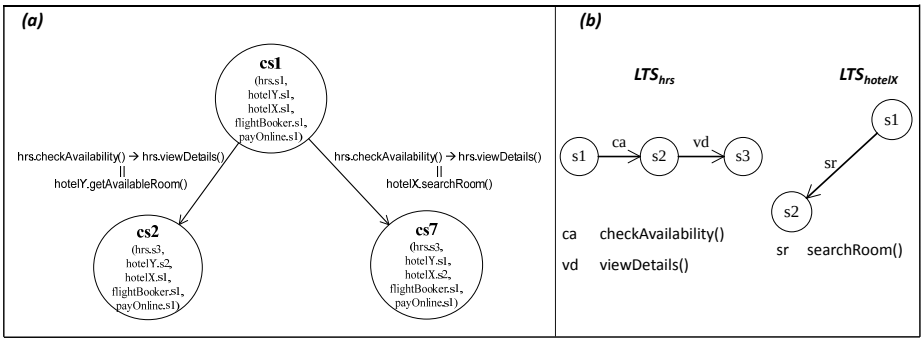

Figure 7.6: (a)Composed LTS after the first Iteration (b) Overlapping Parts of the participating LTSs

7.3.2 LTS Composition Algorithm

We proposed an algoirthm to compose the requested and offered LTSs in a gradual manner and consequently achieve the operation mapping-based composed LTS defined in Def. 10. This algorithm is presented in Listing 9. It takes as input the LTS of a service request lts_r and the set of LTSs of the selected service offers $\{lts_{o_1}, ..., lts_{o_k}\}$. Additionally, it takes the *requester operation mapping*, i.e., $OpMap_r$ as input. As output, it returns a set of determined service compositions, i.e., $Result_{comp}$. If there is no possible service composition, a failure is notified and the requester is provided with suggestions to restructure his/her request based on identified partial compositions.

The algorithm in Listing 9 works as follows:

Step ① In the first step, a composed LTS lts_{comp} is initiated, which is initially empty. Later, a composed state s_{comp}, which is a composition of the initial states of all the participating LTSs is created through `createState()` and added to lts_{comp} through `addState()`. Based on Def. 10(i), $r.s_o$ in s_{comp} represents the initial state in lts_r. Similarly, s_{comp} comprises of the initial states of all the participating LTSs. As shown in Fig. 7.6, $cs1$ shows the initial state of lts_{comp}. In this case all the participating LTSs, i.e., lts_{hrs}, lts_{hotelX}, lts_{hotelY}, $lts_{flightBooker}$, $lts_{payOnline}$ are in their respective initial states.

Step ② s_{cur} is a variable to track the currently traversed composed state during the construction of the composed LTS. In this case, s_{cur} can be represented through the vector of participating states $(r.s_c, o_1.s_c, ...o_k.s_c)$.

Listing 9: Algorithm to compose the LTSs of the service partners and to determine valid service compositions

Input: LTS of Service Request lts_r
Input: Set of LTSs of selected offers $\{lts_{o_1}, ..., lts_{o_k}\}$
Input: Set of operation mappings $OpMap_r$ for r
Output: Set of possible service compositions $Result_{comp}$ OR Failure Notification

`findServiceCompositions`($lts_r, \{lts_{o_1}, ..., lts_{o_k}\}, OpMap_r$)

- $lts_{comp} = null$; // Step ①
- `createState`($s_{comp} : (r.s_0, o_1.s_0, ..., o_k.s_o)$);
- lts_{comp}.`addState`(s_{comp});
- $s_{cur} : (r.s_c, o_1.s_c, ...o_k.s_c) = null$; // Step ②
- **while** lts_{comp}.*hasMoreTraversableStates()* **do** // Step ③
  - s_{cur}=lts_{comp}.`nextTraversedState()`;
  - **while** s_{cur}.*hasInvocableCorr*($OpMap_r$) **do** // Step ④
    - $corr$=s_{cur}.`nextInvocableCorr()` where $corr = (seq_r, seq_o) \wedge r.s_c \xrightarrow{seq_r} r.s_i \wedge \exists o_x \in \{o_1, ..., o_k\} : o_x.s_c \xrightarrow{seq_o} o_x.s_j$;
    - `createState`($s_{tar} : (r.s_t, o_1.s_t, ...o_k.s_t)$): $r.s_t = r.s_i \wedge o_x.s_t = o_x.s_j \wedge \forall o \in \{o_1, ..., o_k\} \setminus \{o_x\} : o.s_t = o.s_c$; // Step ⑤
    - lts_{comp}.`addState`(s_{tar});
    - `createTransition`(t_{comp}:$s_{cur} \xrightarrow{seq_r \| seq_o} s_{tar}$); // Step ⑥
    - lts_{comp}.`addTransition`(t_{comp});
  - **end**
- **end**
- **if** lts_{comp}.*hasValidTraces()* **then** // Step ⑦
  - $Result_{comp}$= lts_{comp}.`getValidTraces()`;
  - **return** $Result_{comp}$;
- **end**
- **else return** Failure_Notification;
- ;

end

Step ③ Next, the `while`-loop traverses over the states of the composed LTS lts_{comp} in a breadth-first manner. In every iteration, the next traversed state determined through `nextTraversedState()` is referred by s_{cur}, which is used to further construct the composed LTS if possible. The traversal continues until `hasMoreTraversableStates()` is *false*, i.e., there are no more states to be traversed.

Step ④ For every state s_{cur} that is currently traversed, every *invocable* operation correspondence from $OpMap_r$ is referred through *corr* and is determined by `nextInvocableCorr()`. This is continued until there are no more invocable correspondences for s_{cur}, i.e., `hasInvocableCorr()` is *false*.

corr represented as (seq_r, seq_o) is *invocable* in the state s_{cur} because its requested operation sequence seq_r can be invoked on the requested LTS lts_r from its respective state comprising s_{cur}, i.e., $r.s_c$. As a result of these transitions, lts_r achieves a state $r.s_i$. Similarly, the offered operation sequence seq_o can also be invoked on one of the offered LTS lts_{o_x} from its respective state in s_{cur}, i.e., $o_x.s_c$ and the resulting state for lts_{o_x} is $o_x.s_c$. According to Def. 10, this invocation of the comprising operation sequences is carried out *in parallel* on the participating LTSs. For $cs1$ in Fig. 7.6, there are two invocable correspondences in $OpMap_{hrs}$. First correspondence under consideration is $(hrs.checkAvailability() \rightarrow hrs.viewDetails()\ ,\ hotelX.searchRoom())$, which is invocable in $cs1$ because $hrs.checkAvailability() \rightarrow hrs.viewDetails()$ can be invoked from $hrs.s1$ reaching $hrs.s1$. Additionally, $hotelX.searchRoom()$ can be invoked from one of the offered LTSs, i.e., lts_{hotelX} from its comprising state $hotelX.s1$. As a result of the transition, $hotelX.s2$ is reached.

Step ⑤ For every *invocable* operation correspondence *corr* of s_{cur}, a new composed state s_{tar} represented as $(r.s_t, o_1.s_t, ...o_k.s_t)$ is created and added to lts_{comp}. In s_{tar}, the requester LTS lts_r and the particular offered LTS lts_{o_x} are in their respective new states reached after the invocation of *corr*, i.e., $r.s_t = r.s_i$ and $o_x.s_t = o_x.s_j$, respectively. All the other offered LTSs are in their same respective states comprising s_{cur}. For instance, for the considered invocable operation correspondence for $cs1$, $cs7$ in Fig. 7.6 is created, where LTS_{hrs} and LTS_{hotelX} are in $hrs.s3$ and $hotelX.s2$, respectively after the invocation of the operation sequences in *corr*. All the other offered LTSs maintain their state from $cs1$ in $cs7$.

Step ⑥ Later, a composed transition t_{comp} with the action $seq_r \parallel seq_o$ is created between s_{cur} and the newly created s_{tar} through `createTransition()` and added to lts_{comp} through `addTransition()`. According to Def. 10(v), t_{comp} represents the composition of the transitions in participating LTSs resulting from the parallel invocation of the operation sequences in $corr$. For example, the composed transition between $cs1$ and $cs7$ in Fig. 7.6 (a) represents the composition of the parallel transitions resulting from $hrs.checkAvailability() \rightarrow hrs.viewDetails()$ and $hotelX.searchRoom()$in LTS_{hrs} and LTS_{hotelX}, respectively. As a result, a composite state and a corresponding composition transition is created for every invocable correspondence of s_{cur}. Conceptually, this can be understood as a composition of the overlapping parts of the participating LTSs.

Step ⑦ Analogously, the composed states $cs2$ and $cs7$ are traversed in the next iteration and as a result, lts_{comp} is further constructed. It is completely constructed if there are no more states to be traversed. For our running example, the composed LTS is shown in Fig. 7.7. After the complete construction, lts_{comp} has to be examined to determine its valid traces, which represent the valid service compositions. This is done through `hasValidTraces()`. In case it holds *true*, these are added to $Result_{comp}$ through `getValidTraces()` and returned. Otherwise, a failure notification is returned. In the next section, we discuss the notion of valid traces in detail and how the composed LTS can be examined to determine such valid traces.

As mentioned earlier, a strength of the proposed approach is its *selective* LTS composition strategy, where only those parts of the offered LTSs are considered that are relevant to the requester and hence *overlap* with the requested LTS on the basis of the determined operation correspondences. Consequently, the composed LTS comprises of only parallel transitions resulting from the parallel invocation of the requested and offered operations in the operation correspondences. In most cases, this strategy results in a smaller composed LTS as compared to a conventionally composed LTS and hence makes its examination to determine potential service compositions easier. For instance, apart from the parallel transitions in the participating LTSs, a conventionally composed LTS can also comprise the interleaving transitions in the participating LTSs. These interleaving transitions result from the operation invocations, which does not participate in any operation correspondence, e.g., from $cs1$, we have so far considered the composed

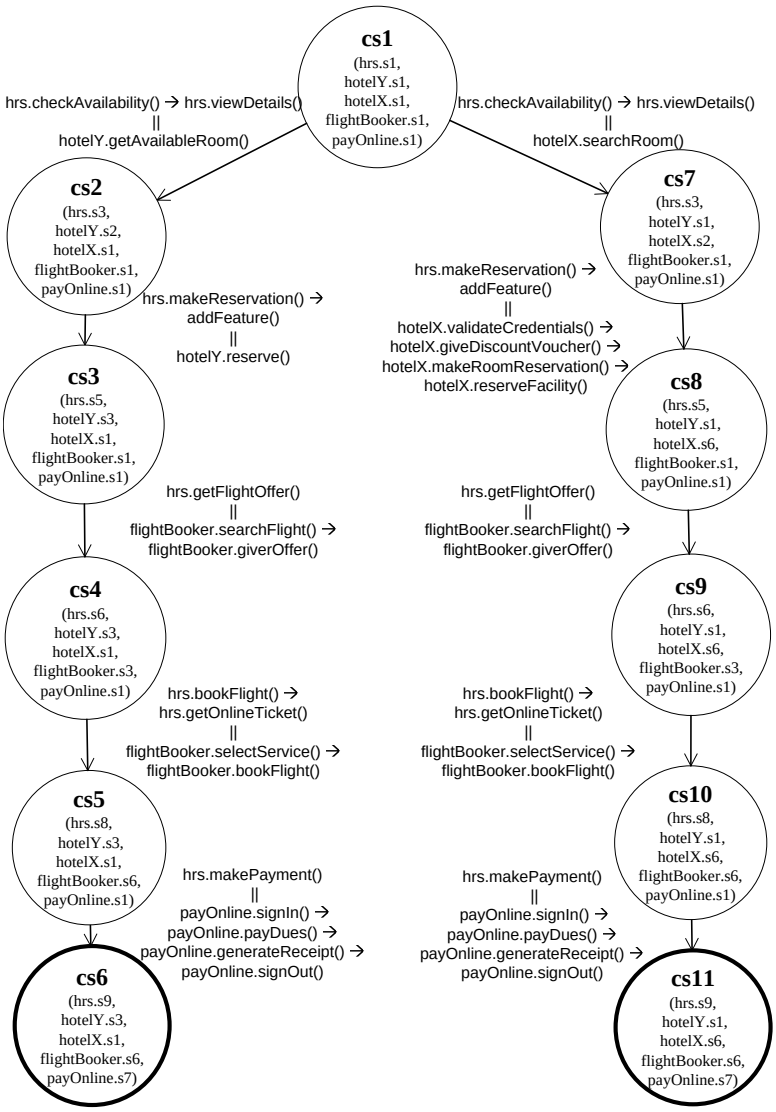

Figure 7.7: Composed LTS for our running Example

transitions that result from the parallel transitions of the participating LTSs. However in a conventional setting, further composed transitions are also possible resulting from the invocation of operations $hotelX.arrangeEvent()$, $payOnline.signIn()$, etc. in their respective LTSs. But in our selective composition strategy, these transitions are not relevant since they are not part of any invocable operation correspondences from the composed state

under consideration and hence are not included to lts_{comp}.

In the next section, we explain the mechanism that is used to examine the composed LTS and determine the valid service compositions. In case of valid service composition, there are some further important observations that we also discuss in the next section.

7.4 Determination and further Examination of valid Service Compositions

After the construction of the composed LTS, our approach examines it in order to determine any valid service compositions satisfying the service request under consideration. As a result, the requester is notified about a *success* or *failure* in service composition phase. These two different scenarios are discussed in detail in the following sections.

7.4.1 Successful Service Composition Phase

The service composition phase is considered to be *successful*, if it can determine at least one valid service composition satisfying the service request. A valid service composition in the composed LTS is defined as follows:

Definition 11 (Valid Service Composition)**.** *Given the composed LTS lts_{comp} for the request r and the selected offers o_1, ..., o_k, a trace $trace_{comp}$ in lts_{comp} represents a valid service composition, if it reaches a final state of the lts_{comp}.* □

As specified in Def. 10, the final state of the composed LTS is reached when the requested protocol is in its final state. This means that the request is completely satisfied by the composition of certain offers selected in the service discovery phase. In our given example, $cs1 \rightarrow cs2 \rightarrow cs3 \rightarrow cs4 \rightarrow cs5 \rightarrow cs6$ and $cs1 \rightarrow cs7 \rightarrow cs8 \rightarrow cs9 \rightarrow cs10 \rightarrow cs11$ are the two traces in the composed LTS, which represent the valid service compositions of the service offers of *HotelX*, *HotelY*, *FlightBooker*, and *PayOnline* satisfying the request of *HRS*. Based on these results, the service requester, i.e., HRS may select a particular composition to satisfy its request.

Like service discovery phase, our approach enables the service requester to examine the results of service composition phase and make suitable choices accordingly. For instance, our approach performs automatic service discovery and composition mainly focusing the functional aspects of the requested/offered service. Therefore, once the possible service compositions

are determined, the service requester may manually match and analyze the non-functional aspects, such as, cost, performance, or reliability, etc. and then negotiate with the service providers to make a final selection.

Similarly, there is another situation where the service requester must negotiate with the service providers manually to decide for a certain valid service composition. This happens when a service composition is *unsafe*, unlike the valid service compositions in Fig. 7.7, which we term as *safe service composition*. A safe service composition is defined as follows:

Definition 12 (Safe Service Composition)**.** *A valid service composition* $trace_{comp}$ *for the request* r *and the selected offers* o_1, ..., o_k *is also termed as a safe service composition if:*

1. *The interaction between the* r *and the selected offers* o_1, ..., o_k *is* sequential*, i.e., another service offer* o_j *is invoked only after the end-to-end invocation of the offered use case of an already invoked service* o_i *is complete.*

2. *None of the involved offered services is in its intermediate state in the final state of* $trace_{comp}$*.* □

The valid service compositions in Fig. 7.7 are termed as safe because the service requester interacts with each offered service in an uninterrupted manner and completes the offered use case, which guaranteeing a successful completion and validity of the results achieved at the end. The importance of such a safe interaction can be better explained with the help of a few scenarios for unsafe service composition. The examples also clarify why it is important to communicate with the concerned service providers before selecting an unsafe service composition.

The first scenario of such a unsafe service composition occurs when the requester does not have a sequential interaction with the offered services rather it switches back and forth during their invocation. In this case, the requester invokes the offered services in an interleaving manner. This can be understood with the help of an example shown in Fig. 7.8. This is a variant of our running example, where the sequence of the functionality required by HRS is slightly different. In this case, it first searches for hotel room and suitable flights with *checkAvailability()* and *getFlightOffer()*, respectively and later proceeds with the booking activity. In this case, one of the valid service composition is shown in Fig. 7.8. The service composition proceeds by first invoking the *HotelY* service to search the hotel rooms. As a result, LTS_{hotelY} is in an intermediate state $hotelY.s2$ after the first

composed transition. Next, it switches to *FlightBooker* to search for suitable flights. The corresponding composed transition results in a transition in $LTS_{flightBooker}$ leading to its intermediate state $flightBooker.s3$. Later, it switches back to *HotelY* to book a hotel room from the search results achieved earlier. In this case, the requester starts interaction with other services while the allowed sequence of an already invoked service *HotelY* is not complete. Consider the case where *HotelY* has certain time constraints and the search results are discarded after a certain period of time. In such a situation, if the requester's interaction with *FlightBooker* takes more time, the search results would be discarded and room booking activity cannot be carried out later successfully with *HotelY*. Hence, it is important for the requester to communicate with these service providers and make sure that his required functionality is indeed fulfilled and such erroneous situations do not arise due to the shift of control from one service to the other in the service composition.

Another scenario where an offered invocation sequence is *partially* invoked arises from the fact that a service request may be completely fulfilled after invoking a *part* of the functionality offered by the service. According to Def. 10, the composed LTS reaches its final state when the request is completely fulfilled, i.e., the requester LTS is in its final state. This means that it is not necessary that all the participating service offers reach their final state in a valid service composition. According to RSDL-specification in Chap. 3, final state of an offered service protocol represents the successful completion of an offered use case and it has to be reached in order to guarantee the validity of the achieved results. This means that a valid service composition where some offered LTSs are in some intermediate state and have not reached their final state depicts a situation where the offered use case is not completed and hence the validity of results is not guaranteed. An example for such a scenario will be discussed shortly.

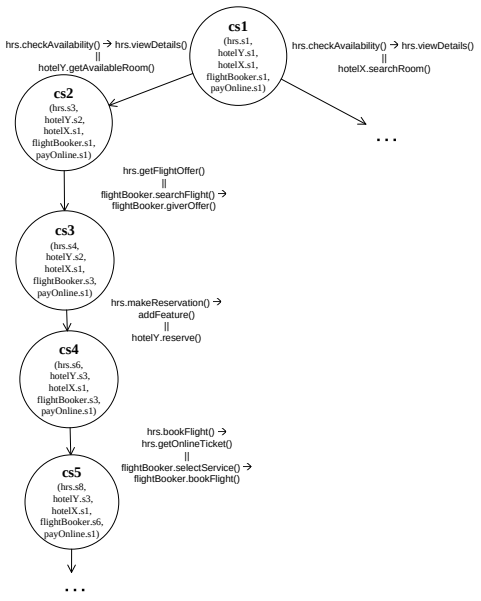

Figure 7.8: A possible Service Composition with interleaving offered LTSs

Consider the offered service protocol of *HotelY* that notifies that the offered use case is complete once the room reservation is done through the operation *reserve()*. Consider two different variants of this offered protocol: in the first variant, the service offer allows the user to recommend the hotel to further users with an operation *recommendToOthers()* after *reserve()*. In the second variant, the service offer allows the user to print its booking confirmation with *printConfirmation()* after *reserve()*. If the composed LTS in Fig. 7.7 is constructed with these variants, LTS_{hotelY} is invoked partially as the required functionality of HRS is complete after *reserve()* and LTS_{hotelY} does not reach its final state in both the valid service compositions. Hence, HRS has to communicate with the service provider to make a final decision. In case of the first variant, the service provider notifies that such a partial invocation is not possible as it implicitly obligate the user to perform recommendation activity or the reservation is canceled otherwise. This means that the offered service does not allow a successful fulfillment of the requester's requirements and hence HRS cannot select this service composition. In case of the second variant, the service provider notifies that the reservation is still valid if the confirmation is not printed and allows HRS to partially invoke the offered use case. Therefore in this case, the HRS can select the service composition with HotelY service for hotel booking activities. In such a case, it is also possible that the service provider revises its published service offer and update it with a new use case that successfully completes after the reservation is done.

As a result of the successful service composition phase, the service requester has some traces of the composed LTS representing valid service compositions. However, these traces of the composed LTS can be termed as the *blueprints* that only represent possible service compositions on a conceptual level. These blueprints are not directly invocable on the actual services and have to be translated to an executable form, e.g., a composed service or an executable BPEL process. At the moment, we assume that the service requester may use these blueprints as guidelines to *manually* define a concrete invocable service composition. Additionally, we argue that an automatic mechanism, i.e., an orchestrator can be developed in future that can extensively use these blueprints and other information produced during different phases of our approach to *automatically* define a concrete invocable service composition.

7.4.2 Failed Service Composition Phase

If the composition analysis cannot find any valid service composition in the composed LTS lts_{comp}, it notifies a failure to the requester. In this case, the requester is provided with feedback in terms of the composed LTS, that does not contain any valid service compositions and additional information about possible reasons of failure. On the basis of this feedback, the requester may consider to restructure his/her service request.

For example, assuming that HRS request a further operation *paymentInfoPerEmail()* (referred as *ppe*) that is invoked after *makePayment()*. Fig. 7.9(a) shows the LTS of HRS for this scenario. Also consider that during operation matching, *paymentInfoPerEmail()* is mapped to the operation *sendEmail()* of the *PayOnline* service (c.f. Fig. 7.4).

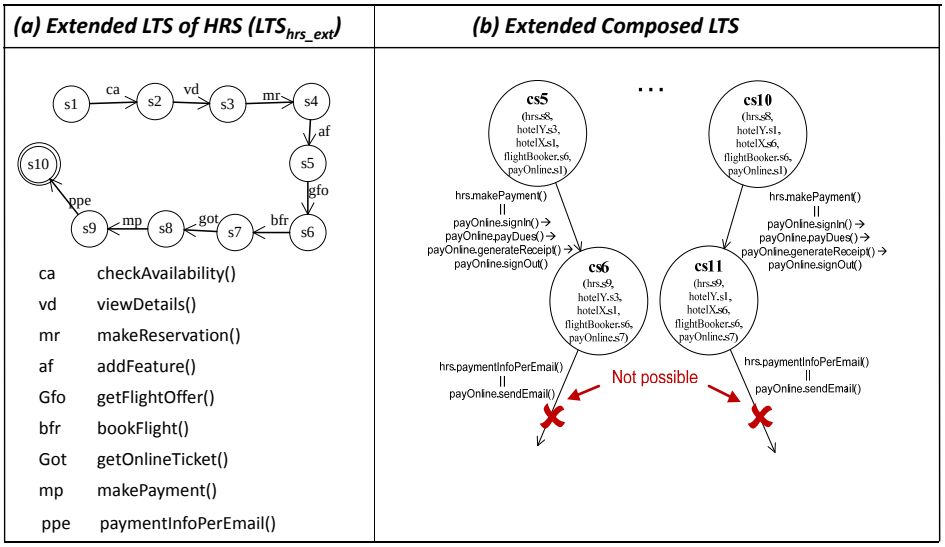

Figure 7.9: Extended LTS of HRS and Composed LTS in the Failure Scenario

Fig. 7.9(b) shows the modified scenario of our previously computed composed LTS (c.f. Fig. 7.7), which was complete after the traversal of *cs6* and *cs11*. In case of Fig. 7.7, *cs6* and *cs11* were the final states of the composed LTS according to Def. 10 hence notifying a successful service composition.

In the modified scenario in Fig. 7.9(b), the composed LTS is complete after the traversal of *cs6* and *cs11*, because there are no further invocable correspondences from either *cs6* or *cs11* and hence the composed LTS cannot be constructed further. However, a final state is not reached in any of

its traces as another transition ppe needs to be invoked on the requested LTS representing the operation *paymentInfoPerEmail()* to reach its final state. Although the operation correspondence (*hrs.paymentInfoPerEmail()*, *payOnline.sendEmail()*) specifies that the *PayOnline* service fulfills this requirement but this correspondence is not *invocable* from either *cs6* or *cs11*. This is for the reason that although *paymentInfoPerEmail()* can be invoked from the respective state $hrs.s6$ of LTS_{hrs} but *sendEmail()* cannot be invoked from the respective state *payOnline*.$s7$ of $LTS_{payOnline}$. As a consequence, none of the traces in the composed LTS reach a final state and hence there is no possible service composition satisfying the HRS request.

In this case, the requester is provided with the composed LTS and the operation correspondence that is not invocable in the composed states *cs6* and *cs11* because the comprising offered operation sequence is not invocable on the participating offered LTS, i.e., $LTS_{payOnline}$ from its respective state. Based on this information, the requester may choose to restructure the request, e.g., it may decide to not include the operation *paymentInfoPerEmail()* in the service request or it may negotiate with the service providers to modify their service offer if possible.

Hence in this case, our proposed service composition mechanism fails to determine any possible service composition based the selected service offers to satisfy the service request.

7.5 Discussion

As evident from the running example, we restrict the service protocols specified in the service requests and offers to be without any loops for the course of this thesis. Therefore, our overall approach in general and the proposed LTS composition mechanism in particular does not consider service protocols with loops so far.

However, considering the importance of loops in the service protocols, we have done some initial investigation in this direction and expect that our findings can be used to extend the proposed approach to also consider loops in the service protocols. It is worth mentioning that with the inclusion of loops in the service protocols, the earlier phases of the proposed approach, i.e., service description normalization and service discovery mostly remain unaffected. In this context, mainly the LTS composition mechanism in the service composition phase will require an extension.

Fig. 7.10 shows a matrix of different cases of loop occurrence in the requested and offered service protocols with varying degrees of complexity.

Requester \ Providers	No Loops	Loops with iteration #	Simple Loops	Nested Loops	Unstructured Loops
No Loops	✓	✓	✓	✓	✓
Loops with iteration #	✓	✓	✓	✓	✓
Simple Loops	?	?	?	×	×
Nested Loops	×	×	×	×	×

Figure 7.10: Different Cases of Loop Occurrences in the Participating LTSs

Additionally, it also describes the capability of our LTS composition mechanism in this regard and possible extension points to handle different cases. In this figure, a ✓ symbol depicts that the proposed mechanism has the capability to handle this case, a ? symbol shows that we have analyzed our mechanism for this case with some examples and our findings can be used as a basis for an extension to handle this case. A × symbol represent the cases that have not been investigated by us so far.

The complexity of the loop occurrence in the requested and offered protocols can vary from *no loops* to *nested ones* and we do not take unstructured loops into account. The first row of this matrix shows that our LTS composition mechanism suffices in the case where the requested service protocol does not have any loops. As described in Sec. 7.3, our LTS composition mechanism is *primarily* moderated through the requested protocol and only those offered sequences are considered for LTS that are *relevant* to the requester. Therefore, in case of single sequence in the requested protocol, the complex loop occurrences in the offered protocols are not relevant for the composition process. Similarly, if the requested protocol has loops with fixed iterations, it can be flattened on the basis of the number of iterations and as a result the proposed composition mechanism is applicable despite loop occurrences in the offered protocols.

On the contrary, this is not true in case of simple loops in the requested protocol. In this case, we have deduced that our proposed mechanism is not completely applicable in its current form. For instance, Fig. 7.11(a) shows the slightly modified variants of two of the LTSs in our running example,

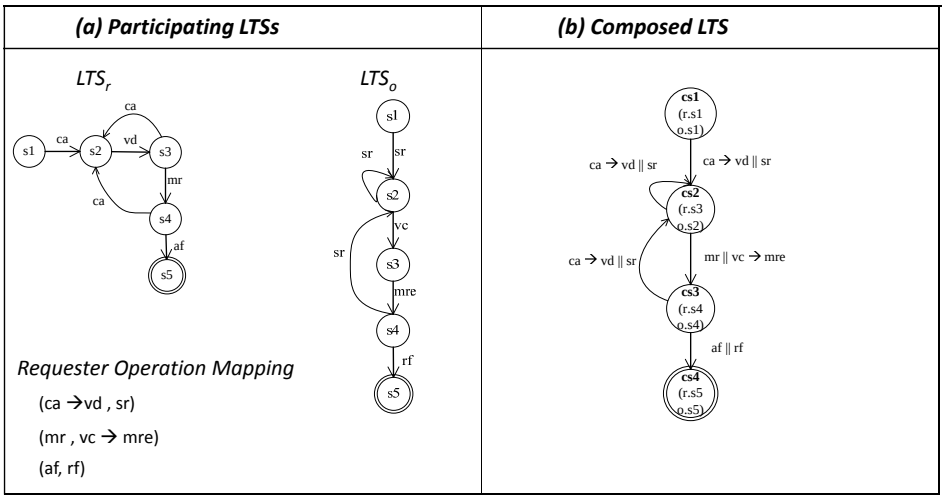

Figure 7.11: An Example for correct Composition of LTSs with Loops

i.e., HRS as lts_r and HotelX as lts_o. In these variants, both the requested and the offered LTS have simple loops. In this scenario, the composed LTS resulting after the LTS composition step of the proposed mechanism is shown in Fig. 7.11(b). During the determination of valid service compositions, our mechanism determines that there is a valid service composition as a trace in the composed LTS reaches a final state. With a closer look on these results, it can be determined that this valid service composition indeed fulfills the requested functionality as the loop pattern required in the requested protocol is also correctly preserved in this service composition. Hence, the results of the proposed mechanism on the basis of the defined criteria can be termed as *correct*. However, this is also important to mention that the criteria for valid service composition does not explicitly check the preservation of required loops and this can lead to incorrect result.

This can be understood with the help of an example shown in Fig. 7.12. In this example, our mechanism will determine a valid service composition, which is represented through the trace that reaches the final state. However, a closer insight reveals that this is not actually true as the service composition represented through this trace does not completely fulfill the requested functionality. The requested protocol specifies the requirement that the operations b, c, and d are invoked in a loop but this requirement is not fulfilled in the composed LTS. Therefore in such a case, the mechanism must be able to detect that the required loop is not preserved in the composed LTS and hence must be able to notify a failed service composition.

On the basis of these findings, we conclude that in case of simple loops in the service protocols, the proposed mechanism needs to be extended.

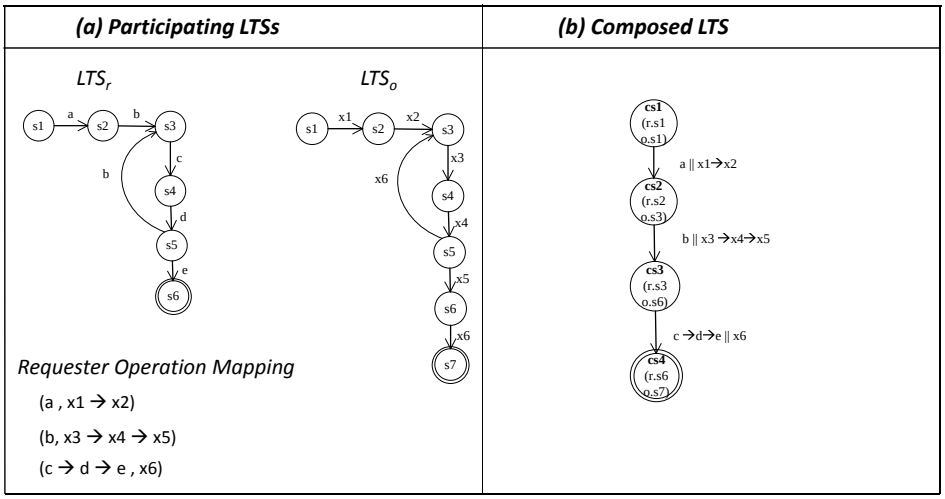

Figure 7.12: An Example for incorrect Composition of LTSs with Loops

So far, our observations for simple loop occurrences in the service protocols through different examples have not been extended to more complex cases of nested loops. In future, further examples can be developed to analyze the behavior of our proposed mechanism in these complex cases and determine possible extension points for the proposed approach.

7.6 Summary

In this chapter, we introduced our service composition approach based on our mechanism for parallel composition of LTSs. For this purpose, first the linguistic heterogeneity of the RSDL service protocols is resolved by translating them to a common semantic domain and hence enable their composition. Based on their DMM-based semantic specification, the requested and offered service protocols are translated to their respective LTSs. Later, a specialized LTS composition operator is used to compose these requested and offered service protocols on the basis of the operation mapping determined earlier. The resulting composed LTS is examined to determine valid service compositions that satisfy the service request. The resulting service composition can act as a blueprint, which can be used to implement actual service composition on the basis of the selected services. At the end of

this phase, the service discovery and composition process is complete for a particular request in the OTF computing environment.

8 Tool Support

In this chapter, we present the tool support for automatic service discovery and composition. For this purpose, a significant part of our service discovery and composition approach introduced in previous chapters is implemented as a prototype, which particularly deals with RSDL-based service descriptions.

The remainder of this Chapter is structured as follows: In the next section, we layout the requirements in terms of use cases that need to be implemented by the workbench. In Sec. 8.2, we present a component architecture for our service discovery and composition workbench. The implemented workbench is explained through a detailed insight into its technical and conceptual aspects in Sec. 8.3. Lastly, Sec. 8.4 presents our evaluation results for the workbench.

8.1 Requirements for the Workbench

In order to enable automatic service discovery and composition, our approach comprises different use cases, which are visualized in Fig. 8.1. These use cases serve as the requirements for a workbench that realizes the proposed approach. In this direction, there are three actors that interact with the workbench namely the service provider, the service requester and the OTF provider.

There are two basic use cases, which are mainly initiated by the service partners, i.e., *Publish Service Offer* and *Perform Service Discovery and Composition*. *Publish Service Offer* allows the service provider to publish his service on the service market. *Perform Service Discovery and Composition* allows the service requester to search for a single or composed service offer to fulfill his request. Both these use cases start with specifying RSDL-based service description for the respective service partner realized through *Specify Service Description* use case. In order to publish the specified service offer or initiate the service discovery for the specified service request, the

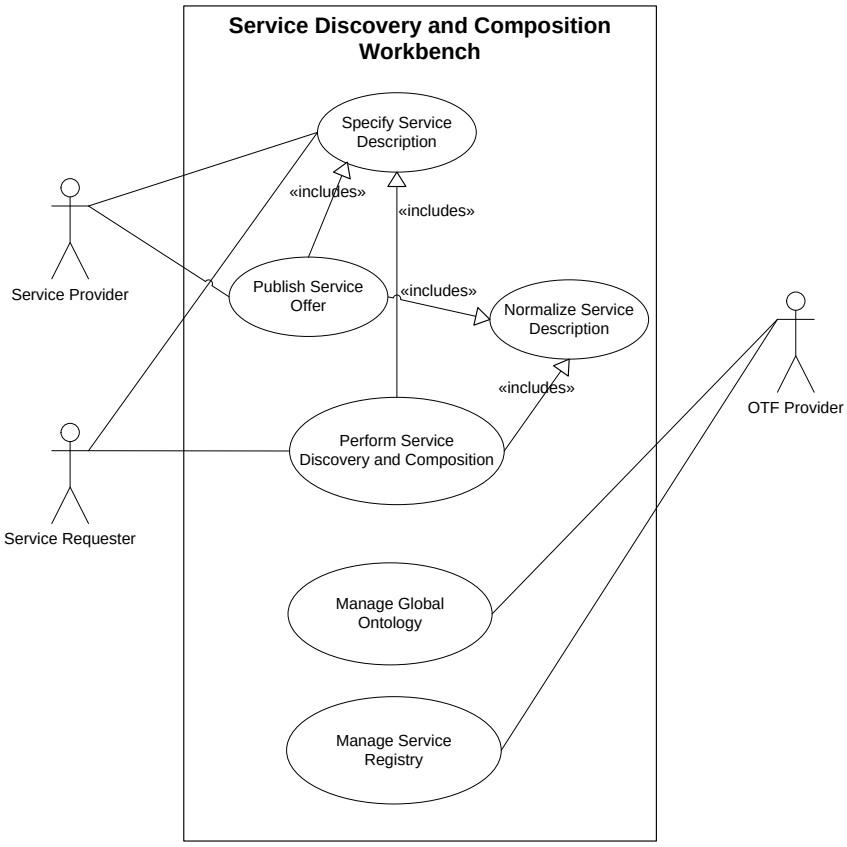

Figure 8.1: Use Cases for our Service Discovery and Composition Workbench

service description has to be normalized to a global representation based on the local-global data model matching. This is realized through the *Normalize Service Description* use case. The OTF provider interacts with the system mainly through two use cases: *Manage Global Ontology* enables the OTF provider to define and maintain the global ontology and its conforming data model, whereas *Manage Service Registry* provides the interface to the service registry where the service offers are maintained according to the categorization hierarchy specified by the OTF provider.

A significant part of these use cases are implemented in our prototype of Service Discovery and Composition Workbench. In the next section, we give an architectural overview of our workbench through its components.

8.2 Workbench Architecture

To give an overview of our *Service Discovery and Composition Workbench*, its component architecture is presented in the Fig. 8.2. Technically, these components are realized through multiple plug-ins based on the plug-in mechanism of the underlying Eclipse development platform.

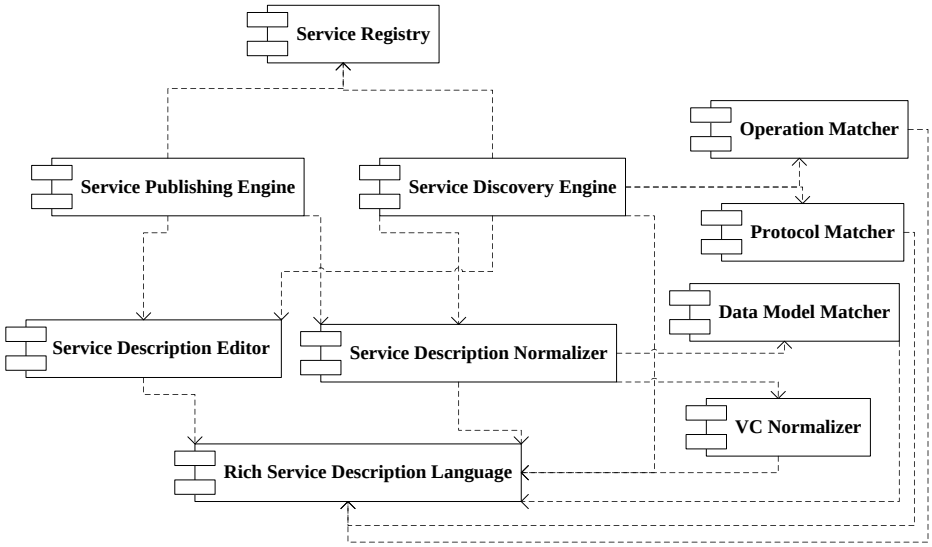

Figure 8.2: Architecture of our Service Discovery and Composition Workbench

In this architecture, the *Rich Service Description Language* defines the language described in Chap. 3 and provides basic operations to specify and access different elements of the comprehensive service descriptions of the service partners.

On the basis of this language specification, *Service Description Editor* provides an elaborate editor that allows the service partners to specify their RSDL-based service requests/offers. For this purpose, it uses the Papyrus and Henshin editors. Similarly, the normalization is carried out through the *Service Description Normalizer* component.

An RSDL-based service description is normalized to a common representation as explained in Chap. 5. For this purpose, *Service Description Normalizer* takes the service description as input and normalizes it using *Data Model Matcher* and *VC Normalizer* components. In this direction, *Data Model Matcher* component is responsible to match the local data model of

the service partner to the global data model of the OTF provider based on the matching strategy defined in Chap. 5. For this purpose, the local data model is annotated with concepts from the global ontology and based on these ontological semantics, the data models are matched. *VC Normalizer* takes the local-global data model mappings from *Data Model Matcher* and is focused on the translation of the visual contracts typed over the local data model to their counterparts typed over the global data model of the OTF provider.

The *Service Publishing Engine* allows the service provider to publish his service offer to the service registry. In this context, an RSDL-based service offer is categorized on the basis of the categorization hierarchy maintained by the OTF provider. On the basis of selected categories, the service offer is published accordingly to the service registry.

For this purpose, the *Service Registry* component is implemented. It maintains a database of the published service offers, which are stored according to their categorization. This service registry is implemented through technologies, such as, Jersey framework[1] for JAX-RS, Apache Tomcat Server[2], MySQL[3], etc.

The *Service Discovery Engine* allows the service requester to search for any published service offers that fulfill his requirements specified in his service request. On the basis of the normalized requests and offers, it enables a multi-level service discovery matching different aspects of the involved service descriptions. In this direction, the *Operation Matcher* realizes the operation matching mechanism defined in Chap. 6. The *Protocol Matcher* enables a $1 : 1$ matching between the requested and the offered protocols, which is a simpler version of the mechanism defined in Chap. 7. This aspect will be discussed shortly in more detail.

In the next section, we explain the implementation details of our service discovery and composition workbench.

8.3 Workbench Implementation

In this section, we introduce the prototypic workbench that implements salient features of our service discovery and composition approach in detail through its technical and conceptual details.

[1] http://jersey.java.net/
[2] http://tomcate.apache.org
[3] http://www.mysql.com/

8.3.1 Tools and Technologies

In this section, we introduce the tools and technologies that serve as the implementation platform for our workbench.

Eclipse

For software development, Eclipse[4] is the most commonly used open-source integrated development environment (IDE). Although majorly aimed to support Java-based software initially, it evolved into a customizable and extendable IDE leveraged through its extensive plug-in system. Our service discovery and composition workbench is also realized through this plug-in mechanism. In this context, we rely on different eclipse-based tools and technologies, which we introduce briefly before going into the details of the workbench.

Eclipse Modeling Framework (EMF)

Eclipse Modeling Framework (EMF)[5] is a modeling framework that allows development of model-based applications. EMF comprises of development tools and offers runtime support for different aspects of such a model-based development, such as, model specification, automatic code generation facility for the developed model, viewing and editing of the model through a visual editor, etc.

The core of the EMF framework is based on a defined metamodel known as *Ecore* and the models defined through EMF are typed over this underlying Ecore metamodel. Ecore model is aligned to Essential MOF (EMOF) model, which is a subset of OMG's MetaObject Facility (MOF) model[6]. Other fundamental parts of EMF are EMF.Edit and EMF.Codegen, which enable editor development for EMF models through facilities like generic reusable classes for editors and code generation for EMF model, respectively. For our workbench, EMF is at the core to specify our service description language RSDL. This in turn serves as the basis for the editor and the matcher for RSDL-based service descriptions.

[4] https://eclipse.org/
[5] http://www.eclipse.org/modeling/emf/
[6] http://www.omg.org/mof/

Papyrus

Papyrus[7] is a graphical editing tool that aims at providing an integrated environment for the editing of EMF models. It comes up with diagram editors for EMF-based modeling languages and also provides the glue to integrate these editors with other model-based development (MBD) tools. In this direction, papyrus provides diagram editors for different modeling languages, such as, UML, SysML and MARTE.

In the context of UML, papyrus provides extensive support by providing a range of editors for different types of UML models conforming to the OMG's UML specification [115]. These UML models created through papyrus are typed over the UML metamodel, which in turn is defined as EMF Ecore model[8]. Papyrus allows detailed extension and customization of different elements, such as, model explorer, diagram editors, property editors, etc. hence supporting the implementation of UML profiles.

Our workbench uses papyrus to implement the editor for the specification of RSDL-based service descriptions.

Henshin

Henshin[9] is a model transformation tool for EMF models. It comprises of an Ecore-based model-to-model transformation language, a graphical editor to create transformations and an execution engine to execute the created transformations.

Our workbench mainly relies on the Henshin editor to specify the visual contracts in an RSDL-based service specification as henshin-based graph transformations. These visual contracts serve as the basis for the operation matching strategy explained in Chap. 6.

EMF Compare

EMF Compare[10] is a facility that allows comparison and merging of EMF models. This facility comes up with a basic infrastructure for model comparison, which can be extended and reused. Additionally, it comes up with a tool integrated in the eclipse IDE that enables a visualization of the model similarities and differences and merging of the models on the basis of this information. Similarly, it facilitates the collaborative team work on models

[7] https://eclipse.org/papyrus/
[8] http://eclipse.org/modeling/mdt/?project=uml2
[9] https://www.eclipse.org/henshin/
[10] https://www.eclipse.org/emf/compare/

through a generic comparison engine and an export of the differences as a model patch.

The model matching process of EMF Compare consists of two phases, namely the matching and the differencing phase. Depending on the individual comparison strategies, EMF Compare allows a customization of the two phases accordingly. Our Data Model Matcher component reuses this basic infrastructure of the EMF Compare and extend it to implement the data model matching strategy explained in Chap. 5.

Query-View-Transformation Relations(QVTr)

Query-View-Transformation (QVT) [117] is a standard for model transformations, which covers the querying and views of a model as specialized types of model transformations. QVT Relations (QVTr) is a specialized language for QVT, which is mainly based on relation definition for bidirectional model transformations.

In the context of our approach, This concept of relations is directly comparable to the data model mappings achieved after data model matching process. These mappings can be translated to QVTr relations and the service description normalization can be realized as the model transformations based on these relations.

Support for QVTr is provided in Eclipse through technologies like QVT declarative[11] and mediniQVT, etc. Our workbench use these technologies to automatically define and execute QVTr-based model transformations and realize the service description normalization on the basis of the data model mappings.

8.3.2 Implemented Features

In this section, we give an overview of the implemented features in the context of the required use cases specified in Sec. 8.1.

8.3.3 Specify Service Description

Whenever a service partner accesses the our workbench to publish his service offer or initiate the service discovery process, it is based on his service request/offer. The *Specify Service Description* use case enables the service partners to specify their RSDL-based service requests and offers in terms of EMF models. Based on our choice of UML as the underlying language for

[11] http://projects.eclipse.org/projects/modeling.mmt.qvtd

our RSDL, Papyrus[12] is selected as the foundation for the service description editor. In this direction, the papyrus editor is customized according to the RSDL specification. Screen shots in the Fig. 8.3 show that the user can particularly select the RSDL palette for papyrus and access RSDL notations to specify his service description.

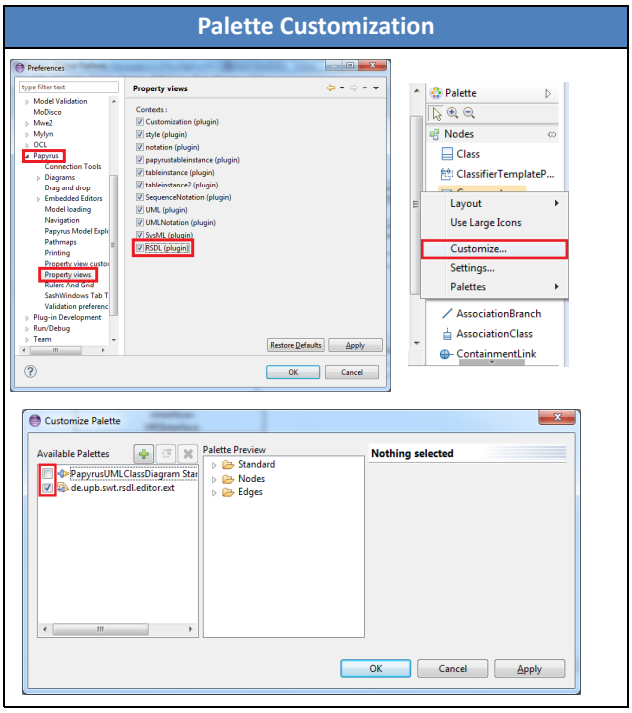

Figure 8.3: Palette Customization Wizard

Additionally, During the specification of his RSDL-based service description, the user is guided through a wizard as shown in Fig. 8.4.

As shown in Fig. 8.5 with different aspects of HRS request, our workbench allows the service partners to comprehensively specify different aspects of his RSDL-based service request/offer through the service description editor. This specification consists of two main tasks: First, service partners can specify their underlying data model as a UML class diagram using a Papyrus-based editor (see Fig. 8.5 (a)).

In addition, service partners can specify their requested/offered functionality in terms of its different aspects, such as, operation signatures, their respective visual contracts and the requested/offered service protocol typed

[12]http://www.eclipse.org/papyrus/

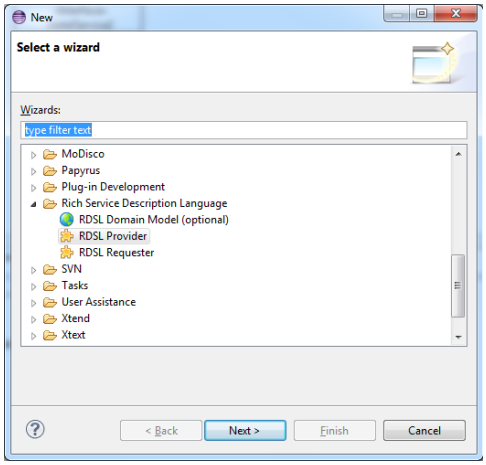

Figure 8.4: Wizard for specifying RSDL-based Service Description

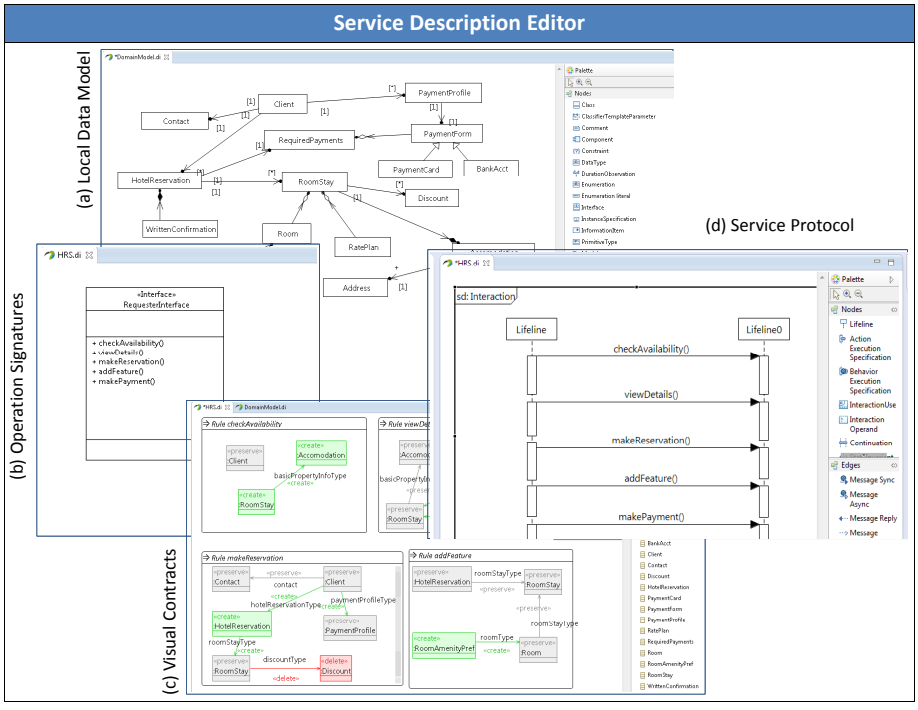

Figure 8.5: Screenshots of the Service Description Editor in our Workbench

over their data models (see Fig. 8.5 (b, c, d)). The operation signatures and service protocols are specified using the Papyrus editor for UML Interface

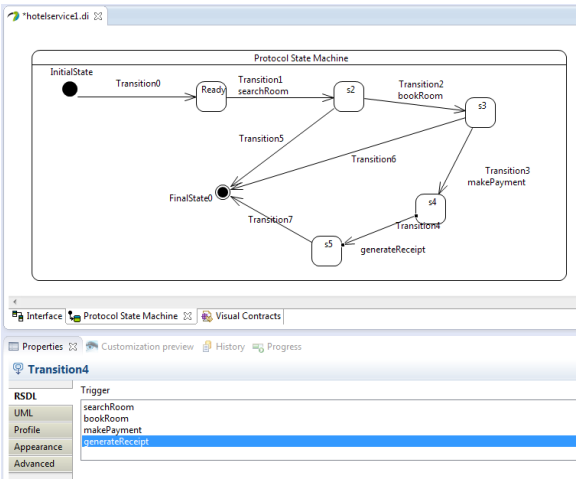

Figure 8.6: Offered Protocol Specification as UML Statemachine Diagram

and UML sequence diagrams / UML statemachine diagrams, respectively. Similarly, for a service offer, Fig. 8.6 shows the specification of the protocol as UML statemachine diagram for a hotel service.

The service description editor relies on Henshin editor[13] to specify the visual contracts (VCs). The Henshin editor supports the specification of VCs as graph transformation rules that are typed over an underlying Ecore model. A detailed mechanism is defined to navigate between the Papyrus and Henshin editors. This mechanism automatically translates the data model specified as UML class diagram in Papyrus editor to an Ecore model for VC specification in Henshin editor. Additionally, this mechanism also provides the glue to propagate the modifications in the Papyrus-based data model to the corresponding Ecore model in Henshin. Overall, the specification of service requests/offers is enabled through the coherent environment provided by the service description editor, which is based on a seamless customization and integration of the Papyrus and Henshin editors.

8.3.4 Normalize Service Description

The service descriptions of the service partners have to be normalized in order to enable their matching. This normalization is carried out in *Normalize Service Description* use case. In this use case, the workbench enables the automatic translation of the service description to a common representa-

[13] http://www.eclipse.org/modeling/emft/henshin/

tion conforming to the global ontology and the respective global data model maintained by the OTF provider.

This normalization is carried out in two steps: In the first step, the local data model is matched to the global data model of the OTF provider. For this purpose, our workbench relies on the EMF Compare Infrastructure. It reimplements the provided interfaces according to the proposed data model matching algorithm in Chap. 5 and also reuses the EMF Compare GUI to display the data model matching results.

As discussed in Chap. 5, the matching of the local and global data models mainly depends on their ontological semantics defined through the global ontology of the OTF provider. As mentioned in Chap. 5, the global ontology comprises multiple domain-specific ontologies (DSOs). An automatic access to these ontologies in order to define and later use the ontological semantics is enabled through use of different technologies, e.g., Jena framework[14] is used to access the ontologies defined in commonly-used languages, such as, OWL, RDFS, etc. Similarly, technologies like extJWNL library[15] and SPARQL web interface[16] are used to access WordNet and DBpedia, respectively. For the example under consideration, the global ontology comprises of an example tourism ontology, which is mainly based on the concepts from HarmoNET ontology[17] with some extensions.

Based on the defined ontological semantics, the local-global data model matching is performed, which results in the data model mappings as shown in Fig. 8.7. As seen here, the local data model of HRS (shown in the left pane) is matched to the global data model of the OTF provider (shown in the right pane). For the local class `Accommodation`, the global class `Hotel` is found as a match and it further shows the matching between the attributes of these two classes. During this matching, our mechanism does not consider the associations of the classes so far.

The second step of the the service description normalization process is the normalization of visual contracts based on the data model mappings achieved in the first step. This is realized by our workbench as QVTr-based model-to-model transformation based on an automatically generated QVTr transformation script. This automatic generation is driven by the data model mappings where each class mapping corresponds to a relation in the transformation script. The automatic generation of the transformation script is carried out by the eclipse plugin for QVTr, i.e., *QVTd*.

[14] http://jena.apache.org/
[15] http://extjwnl.sourceforge.net/
[16] http://dbpedia.org/sparql
[17] http://euromuse.harmonet.org

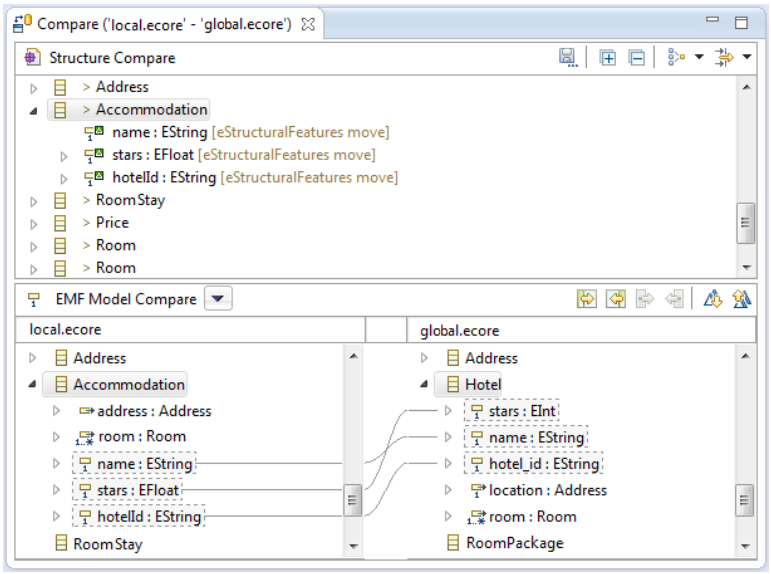

Figure 8.7: Data Model Matcher based on EMF Compare

Fig. 8.8 shows such an automatically generated QVTr script for the given example. As it shows, the script comprises of the relations generated through the data model mappings achieved earlier. For instance, the relation `Hotel_Accomodation` defines how an `Accommodation` object can be transformed to a `Hotel` object and vice versa.

Taking the generated transformation script as input, the transformation execution engine of *mediniQVT* is used to run the transformation. This results in a transformation of the VCs typed over the local data model to their global counterparts. Consequently, the *normalized* service description is achieved.

8.3.5 Publish Service Offer

The service provider interacts with the service discovery and composition workbench mainly through the *Publish Service Offer* use case. In order to publish his service on the OTF market, the service provider defines his service offer realized through the *Specify Service Description* use case discussed earlier. Later, he normalizes his service offer according to the mechanism described in *Normalize Service Description* use case.

After the definition and normalization of his service offer, he can publish his service offer in the service registry using service publishing engine.

```
top relation Hotel_Accommodation{
      var_stars : ecore::EInt;
      var_name : String; ...
      enforce domain global dom_hotel : provider::Hotel {
                  stars = var_stars,
                  name = var_name,
                  room = var_room : global::Room {} ...
      };
      enforce domain local dom_accom : local::Accommodation {
                  stars = var_stars,
                  name = var_name,
                  address = var_address : local::Address {}...
      };
      when {

            Room_Room(var_room, local::Room.allInstances()->any(true)); ...
      }
}
top relation Address_Coordinates_Address{
      var_city : String;
      …
      enforce domain global ag : global::Address {
            coordinates = c,
            city = var_city,
            …
      };
      enforce domain global c : global::Coordinates {
            …
      };
      enforce domain local al : local::Address {
            town = var_city,
            …
      };
}
```

Figure 8.8: Automatically-generated QVTr Script through local-global Mappings

For this purpose, he categorizes his offer on the basis of the categorization hierarchy maintained by the OTF provider. Consequently, his service offer is stored under the selected categories.

Fig. 8.9 shows a screenshot of the service publishing engine. In this screenshot, the normalized RSDL-based service offer of a hotel service is categorized according to the categorization hierarchy maintained by the OTF provider and stored in the service registry on the OTF market.

8.3.6 Perform Service Discovery and Composition

This use case provides the main access point for the service requester to interact with the workbench. Before initializing the service discovery and composition in the OTF market, the service requester defines and normalizes his service request as described earlier.

The service discovery engine in our workbench allows to automatically and accurately match the service request and the available offers through the multi-level service discovery approach explained in Chap. 6. Fig. 8.10 shows the screenshots of the service discovery engine.

As shown in Fig. 8.10(a), a subset of the available hotel service offers

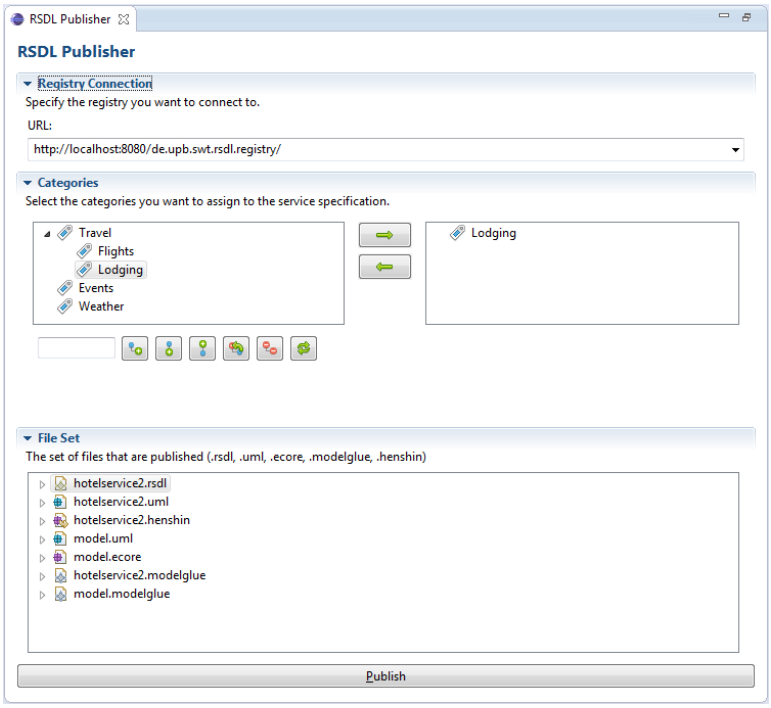

Figure 8.9: Screenshot for the Service Publishing Engine in our Workbench

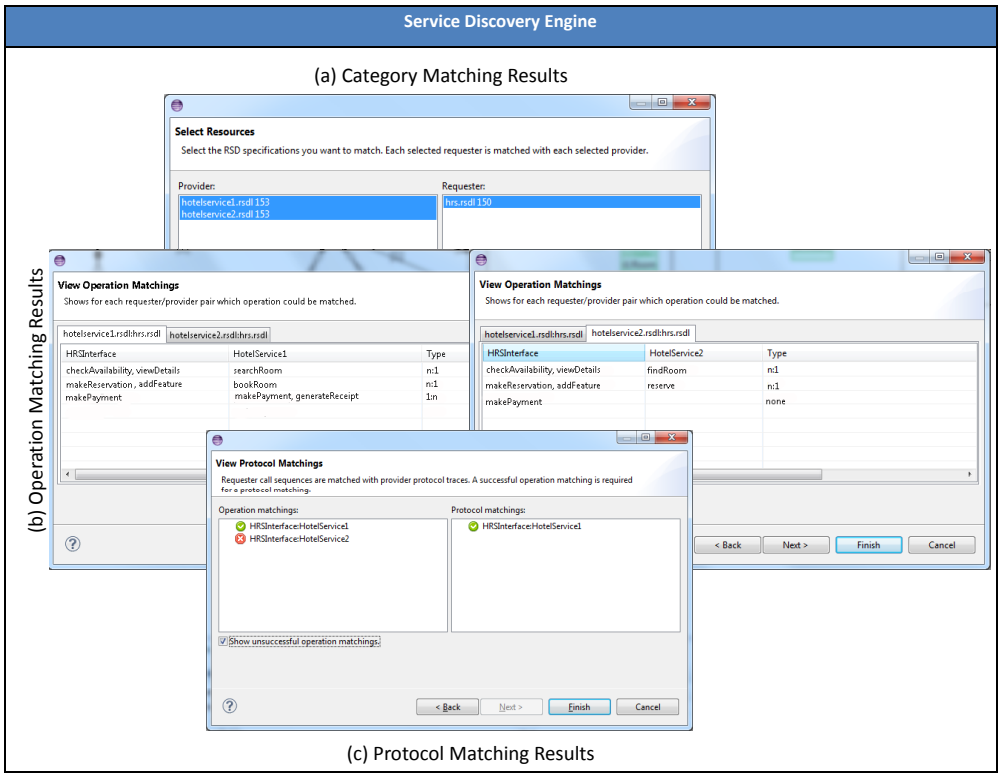

Figure 8.10: Screenshots of the Service Discovery Engine in our Workbench

are selected on the basis of the category matching between the HRS request and the available offers. On the next level, an operation matching is performed between the HRS request and the selected hotel service offers on the basis of the set of operation matching strategies presented in Chap. 6. Fig. 8.10(b) shows the resulting operation correspondences between the requested operations of HRS and the offered operations of the available hotel services.

In the next step, a protocol matching is performed between the request and the selected offers, which is based on the operation correspondences achieved after the operation matching. So far, our workbench implements a relatively straightforward $1:1$ matching between the requested and the offered protocols. Fig. 8.10(c) shows the results of such a protocol matching where the required invocation sequence in HRS protocol matches with the offered invocation sequence of hotel1 shown in Fig. 8.6. Due to time constraints, the implemented workbench does not realize the $1:n$ proto-

col matching and composition mechanism for service composition defined in Chap. 7. However, we believe that a future extension of the workbench which implements the proposed service composition mechanism can be greatly supported through the reuse of existing DMM tool support.

8.3.7 Manage Global Ontology

Apart from the service partners, the OTF provider interacts with our workbench to set up and maintain the OTF computing environment. In this direction, he initiates *Manage Global Ontology* use case to build the global ontology comprising the domain-specific ontologies (DSOs) for the domains that it caters to. This also includes the definition of conforming global data models for the respective domains. Additionally, if the OTF provider later decides to provide services to a new domain , then the global ontology has to be extended through this use case.

In this direction, so far our workbench does not provide an ontology editor for the OTF provider and we assume that a an existing ontology development tool or an existing commercial ontology is used for this purpose. Additionally, papyrus editor is used to define a conforming global data model.

8.3.8 Manage Service Registry

Through this use case, the OTF provider has the option to maintain the service offers published in the service registry. The first task in this direction is the maintenance of the categorization hierarchy. This is achieved through a MySQL database in our workbench. Fig. 8.11 shows a screenshot of the interface available to the OTF provider to manage this categorization hierarchy. This interface can be used to add new categories, remove or modify the existing ones, etc.

Additionally, the service registry is also maintained as a mySQL database and an interface similar to the one shown in Fig. 8.11 is provided to have an overview of the published service offers.

In the next section, we present our evaluation results for our workbench.

8.4 Evaluation

For our work, our basic objective was to automate the service discovery and composition process while ensuring accurate results. Consequently, the evaluation of our approach comprises two aspects: firstly, the degree of

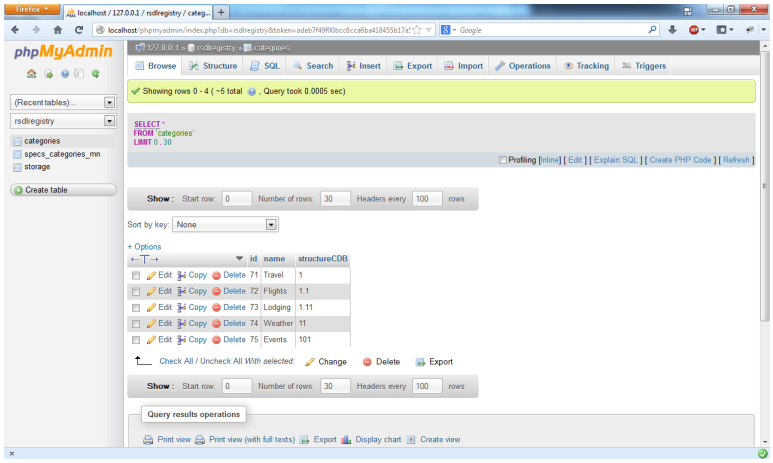

Figure 8.11: A Screenshot of the Categorization Hierarchy Management for the Service Registry

automation that can achieved and secondly, the accuracy of the matching results achieved through our approach.

For the first aspect, a significant part our approach has been automated through the implemented workbench discussed in Sec. 8.3 in detail. Hence, we conclude that the the significant degree of automation of the service discovery and composition process can be achieved.

For the second aspect, there are two types of automatic matching results that need to be evaluated for their accuracy: firstly, the local-global data model matching results, which serve as the basis for the normalization of the requested/offered service description to a common representation. Secondly, the service description matching results between the service request and the available offers on the service market.

We comply to Ontology Alignment Evaluation Initiative [18] (OAEI) to assess the accuracy of results for our data model matching approach. OAEI is an initiative that aims for a standardized performance evaluation of matching systems for ontologies/data models and identify their strengths and weaknesses. In order to test different matching systems, OAEI executes these systems on the test data sets using Semantic Evaluation at Large Scale[19] (SEALS) platform.

The evaluation of our data model matching system is carried out us-

[18] http://oaei.ontologymatching.org/
[19] http://www.seals-project.eu/

ing the 2012 evaluation campaign of OAEI. This evaluation allowed an assessment of the matching results of our system and classified it among 23 matching systems participating for this evaluation campaign. It comprises 5 different test data sets each comprising sufficient number of test cases based on 4 different data models maintained by the OAEI. These models belong to different information domains and also vary in their sizes. In each individual test case, a data model is matched to a modified version of itself. To assess the quality of matching results, they are compared to a reference mappings defined by OAEI for the model pair in question.

The evaluation was conducted on a machine with dual core 32-but processor with 2.4 GHz and 3 GB RAM. In terms of time consumption, our matcher took on average 183 seconds per test, which we consider to be satisfactory in comparison to the average of 557 seconds and 6 seconds per test for the slowest and the fastest matcher, respectively.

The accuracy of the matching results are based on three well-established metrics in the area of information retrieval [7]: Precision, Recall and F-measure, which are defined as follows:

Definition 13 (Precision)**.** *Let C be the set of the reference mappings for the data models o_1, o_2. Further let T be the set of mappings retrieved while matching o_1, o_2 with a matching system Sys. Then precision of Sys is defined as:*

$$P = \frac{|C \cap T|}{|T|}$$

Definition 14 (Recall)**.** *Let C be the set of the reference mappings and T be the set of retrieved mappings of a matching system Sys. Then recall of Sys is defined as:*

$$R = \frac{|C \cap T|}{|C|}$$

Definition 15. ***F-measure.*** *With the definitions of R and P, the F-measure F is defined as:*

$$F = 2\,\frac{P\,R}{P + R}$$

For a particular matcher under consideration, the *precision* represents the ratio of the correct mappings determined by the matcher to the total number of the mappings that it defined. Similarly, recall represents the ratio of the correct mappings determined by the matcher to the total number of possible correct mappings. Both these metrics can be misleading if considered in isolation, e.g., a matcher that determines a relatively small number

of mappings that are mostly true will have a high precision and thus can be termed as highly accurate if precision is considered in isolation as an evaluation metric. However, an additional consideration of recall value, which will be low in this case will enable a correct evaluation of the matcher's accuracy. Similarly, using high recall value for a matcher as an indication of higher accuracy can also be incorrect. In this case, the matcher can have a higher ratio of false positives along with true positives leading to lower precision and hence cannot be termed as accurate. Therefore, the third metric F-measure is relatively more meaningful, which is a harmonic mean of precision and recall.

Based on these metrics, Fig. 8.12 shows the matching results of our data model matcher in comparison with some selected matchers that participated for the evaluation campaign. In these matchers, MapSSS and ASE could be considered the two top-most accurate matchers on the basis of their F-measure value. Edna is a simple string-based matcher, which is used as a baseline in this case. The detailed results of this evaluation are presented in [1].

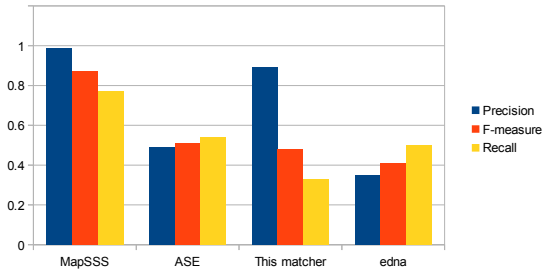

Figure 8.12: Comparison of the Matchers in OAEI Evaluation Campaign

Compared to the top accurate matchers, we consider the performance of our matcher satisfactory and the accuracy of our data model matching results can be termed as reasonably high. Here it is important to mention that our data model matcher in particular had incorrect matching results in the cases where the data model elements did not have meaningful names. This is because the semantic matching of the elements mainly depends on their ontological semantics in the domain ontology defined on the basis of their names. This aspect needs to be further investigated in future that what other information in the data model can be used for the definition of ontological semantics in case of ambiguous names.

In order to evaluate the accuracy of the service description matching results, a particular service request needs to be matched to the service offers published on the service market. For this purpose, a service market with sufficiently large number of diverse service offers is required. This is partially

realized as the service registry component in our workbench but it still lacks a comprehensive collection of service offers. As part of CRC 901 "On-The-Fly (OTF) Computing", our outlook for future work also includes the development of such a comprehensive OTF service market with adequate service offers to extensively evaluate the accuracy of our service description matching results.

8.5 Summary and Discussion

In this chapter, we have presented our prototypic workbench for automatic service discovery and composition, which is fundamentally based on the plug-in mechanism provided by Eclipse. For its implementation, we have used different eclipse-based tools and technologies, such as, EMF, EMF Compare, Papyrus, Henshin, etc. In this workbench, we have implemented several components realizing multiple phases of our approach, such as, service description normalization, service publishing and service discovery, etc. Additionally, we also presented our evaluation results for different parts of this workbench.

In future, we aim to extend it with features that have not been implemented so far, e.g., the service composition mechanism, etc. and carry out an extensive evaluation on the CRC platform with case studies from different domains.

9 Conclusion and Future Work

In this thesis, we have presented our framework for automatic service discovery and composition, which is one of the major milestones aimed by the service-oriented computing (SOC) paradigm. In this concluding chapter, we summarize the work presented in this thesis.

In the next section, we first give a brief summary of the presented framework and an overview of the contributions made. Next, we present an outlook for the important future work to further extend and enhance this work. Finally, we conclude with some final remarks and the lessons learnt during the course of this work.

9.1 Summary and Contributions Overview

The main aim of this thesis was to propose an approach that allows to achieve one of the major goals of SOC, i.e., enabling automatic service discovery and composition while ensuring accurate results. For that purpose, a mechanism for automatic service matching is required, which ensures the accuracy of results by comprehensively matching the service requests and offers in terms of their structural as well as behavioral aspects. In this direction, the current standards of the service description, such as, WSDL [162] allow a limited specification of the requested/offered functionality. Therefore, there is a need for languages that allow comprehensive specification of service requests and offers in terms of their different aspects. Additionally, conforming to the essence of SOC, which allows the service partners to function in their independent domains, the service description matching mechanism must overcome their underlying multi-faceted heterogeneity while matching their heterogeneous service descriptions.

To achieve the goals of this thesis, we presented a framework for automatic service discovery and composition, which ensures accurate results by matching comprehensive service descriptions while overcoming the underly-

ing multi-faceted heterogeneity of the service partners. In the following, we give an overview of the main components of our framework together with the important contributions.

1. **Rich Service Description Language:** In order to enable the service partners to specify their service requests and offers comprehensively, we presented *Rich Service Description Language (RSDL)*. In this direction, our focus was to come up with elaborate notations to specify different aspects of the service descriptions comprising structural and behavioral details of the requested/offered service. RSDL comprises multiple artifacts that cover these aspects of the service descriptions in detail. Additionally, it ensures a wider acceptance by conforming to the de facto industry standard UML, whose visual notations make it easier to use. Similarly, complying to the particular scenario of OTF computing under consideration, RSDL provides specialized notations for the specification of service requests and offers according to their particular features.

   We defined the syntax of RSDL in terms of a meta-model, which conforms to the syntax of the underlying UML constructs [115]. For the formal semantics, some relevant existing approaches [96, 64, 137, 110] are reused. These approaches are mainly based on an important concept from the area of graph theory, i.e., typed graph transformation rules [34]. Such formal semantics for RSDL allows an automatic processing and verification of its different artifacts during service discovery and composition process.

2. **Local-Global Data Model Matching and the Service Description Normalization:** One of the strengths of our framework is its elaborate mechanism to overcome the multi-faceted heterogeneity of service partners while matching their service descriptions. The first facet of this is the data model heterogeneity of the service partners. To overcome this heterogeneity, our framework introduces a technique that automatically normalizes the service descriptions from their local to a common representation in the public domain. Such a normalization to a common representation enables the automatic service description matching despite their heterogeneous data models.

   A local-global data model matching mechanism serves as the basis for this service description normalization, which extensively relies on the concept of standardized domain information as global ontology. Our framework provides guidelines to the OTF provider to define such a

global ontology and a conforming global data model. The data model matching mechanism allows a semantic matching between the local and the global data models based on their ontological semantics in the global ontology. This matching exploits a variety of semantic matching techniques [52] and results in local-global data model mappings. The service description normalization technique uses these data model mappings to translate the local service description to their global counterparts automatically.

3. **Multi-level Service Discovery:** As a first step of the service description matching process, our framework presented a multi-level service discovery mechanism that selects and gradually filters the potential service offers that may satisfy the requester's requirements. The core functionality of this service discovery mechanism is based on an operation matching between the request and available offers.

   Realizing the importance of a comprehensive matching between service descriptions, our operation matching approach matches the requested and offered operations on the basis of their structural and behavioral aspects. Additionally, it also considers the granularity level heterogeneity between the service partners and does not restrict itself to 1 : 1 correspondence between the requested and the offered operations. Rather, it aims to identify 1 : 1 , 1 : n, n : 1, and n : m correspondences between these operations.

   In this direction, we carried out an in-depth analysis and precisely identified different cases of such complex correspondences and formalized the structure of 1 : 1 , 1 : n, n : 1 correspondences on the basis of these identified cases. Further, we proposed a set of matching algorithms to identify such operation correspondences between the request and the available offers. As a result of operation matching, a set of possible operation correspondences for the service request is identified and on this basis, a subset of offers is selected to determine any valid service compositions in the next phase of the approach.

4. **Service Composition through Protocol Matching:** After the service discovery phase, which results in a subset of candidate offers, our approach allows to determine possible service compositions based on a protocol matching mechanism. In this direction, an important concern is to resolve the linguistic heterogeneity of the requested and offered protocols specified in RSDL. For this purpose, our mechanism allows an automatic translation of the heterogeneous requested and

offered protocols to a common semantic domain, i.e., labeled transition system (LTSs) through their semantic specification as DMM rules.

Based on the resulting LTSs, a 1 : n protocol matching between the requested protocol and the offered protocols of the candidate offers is performed. This is enabled through a specialized parallel LTS composition operator that allows a composition of the participating LTSs on the basis of the complex operation correspondences resulting from service discovery phase. As a result of this LTS composition, possible service compositions are determined that completely fulfill the requester's requirements. Additionally, our mechanism provides additional information by differentiating among different types of service compositions, such as, safe and unsafe compositions, which further supports the service requester to make a final selection.

5. **Tool Support and Evaluation:** As a proof-of-concept for our service discovery and composition approach, we implemented a prototypic workbench, which realizes a significant part of our approach. Based on the Eclipse plug-in mechanism, this workbench implements some major usecases, such as, specifying and normalizing of the service descriptions, performing service discovery and composition, managing service registry, etc. For this purpose, different eclipse-based tools and technologies, like EMFCompare, Papyrus, Henshin, etc. are used.

   Apart from the automation of the proposed approach through the implemented prototype, we evaluated the accuracy of our data model matching mechanism, which is implemented as a part of the service description normalization usecase. For this evaluation, we comply to Ontology Alignment Evalaution Initiative (OAEI) that is a public forum to evaluate the performance of matching systems. In comparison to the top accurate matchers submitted to OAEI, the accuracy of our data model matcher is reasonably high. We claim that other parts of the workbench will also be evaluated extensively in future on the basis of a comprehensive service repository and different case studies developed as a part of CRC 901 - OTF Computing.

9.2 Outlook on Future Work

Apart from its current features and contributions, there are certain aspects that are so far not covered by the proposed framework. Similarly, there are certain directions in which this framework can be extended to increase its

scope and make it more effective. We give this outlook on future work as follows.

Our Rich Service Description Language (RSDL) aims at comprehensively specifying service descriptions and hence enable their accurate matching. However, currently its notations are limited to the specification of functional aspects of the requested/offered service. Recently, the researchers in SOC have also shifted their focus towards devising approaches for the specification and matching of non-functional aspects of the service description [91]. A promising extension for RSDL can be the specification of such non-functional aspects in service requests and offers. In this direction, the OMG's UML profile for modeling quality of service and fault tolerance (QFTP) [114] can serve as a foundation. Later, the proposed service description matching mechanism has to be extended as well to automatically match these non-functional elements of service descriptions.

In the context of the proposed local-global data model matching mechanism, an extension of global ontology definition and maintenance mechanism is required, which enables the alignment of DSOs in the global ontology to support service descriptions spanning multiple domains and allows a more flexible modification of the global ontology. The proposed local-global data model matching mechanism so far does not consider independent local ontologies of service partners. It is worth investigating that how the existence of such local ontologies can support the data model matching mechnaism, e.g., the information in the local ontology might be used to improve the annotation of the local data model in the global ontology. Similarly, calculation of similarity value between data elements can be further enhanced with other types of similarity coefficients. In this direction, [52] defines a variety of similarity coefficients that can be calculated between data elements for their matching. For service description normalization, certain aspects need further investigation, such as, attribute normalization, information loss, etc.

In case of the proposed operation matching mechanism, the accuracy of matching results can be enhanced by matching further elements of visual contracts, which have not been considered so far. This means an extension of the matching mechanism with the attributes, negative/positive application condition and multi-objects. Additionally, on the basis of the detailed analysis about different possible scenarios, the definition of a formal structure and matching algorithm for different kinds of n : m operation correspondence is also a potential area for future work in this context.

For the proposed service composition mechanism, a possible extension is to consider requested and and offered service protocols with complex notations. For instance, a requested protocol with branching, multiple invocation

sequences, and loops, etc. can be considered in future. Similarly, the offered protocol can also be extended to include parallelism, loops, etc. In Sec. 7.5, we already pointed out different scenarios that can occur in case of loops in the requested and offered protocols. On the basis of these initial observations, it can be further investigated to see how the LTS composition mechanism and the notion of valid service composition has to be extended in such cases. Another possible future extension of the framework is the implementation of a concrete service composition based on the blueprint resulting from service composition phase. In this direction, the RSDL should also be extended to contain concrete binding details to the actual underlying services for the service offers.

Another important aspect that can further enhance the relevance of the proposed framework is the development of a feedback mechanism in future. In case a valid service composition fulfilling the request cannot be determined by the proposed framework, the requester can be assisted by providing him feedback about the cause of failure and how to resolve the failure. For this purpose, the feedback mechanism has to gather and analyze the information produced during different phases of the service matching process. Based on this analysis, it formulates suggestions for the requester for any possible modifications in his request that can improve the chances of achieving a valid service composition with the proposed framework.

Further, the proposed framework only consider an *exact* match between the service requests and offers where the requested functionality has to be completely fulfilled by the determined service composition. However, in case an exact match cannot be found, an *approximate* match where the determined service composition partially satisfies the requested functionality can also be relevant for the requester. For such a future extension, one possibility is to consider *plugin* and *subsume* type similarity between VC elements during operation matching resulting in an approximate matching between operations. An existing approach for approximate matching developed in the context of OTF computing [126] can provide the basis to extend the proposed framework in this direction.

Finally, another interesting area that needs further exploration in future is the adaptation of the proposed framework for a dynamic environment. The proposed framework allows the requester to carry out service discovery and composition process at design time. It does not support the scenario where an on-the-fly adaptation of an existing service composition is required to support new functionality, incorporate new services, and replace failed services, etc. It can support to fully realize the vision of OTF computing with service discovery and composition in such a dynamic and on-the-fly

environment.

9.3 Final Remarks

The contributions made in this dissertation enable OTF computing to achieve a major goal of the service-oriented computing (SOC), i.e., enabling seamless interaction among heterogeneous service partners.

We are convinced that the full potential of SOC can only be utilized if the service partners are allowed to have a seamless collaboration along with complete independence to function in their independent domains. This can be ensured through an automation of service discovery and composition process while overcoming the underlying heterogeneity of the service partners. Consequently, we came up with an elaborate service discovery and composition framework that effectively meets these challenges.

We believe that despite certain unsolved issues, the proposed approach is a promising endeavor towards meeting the currently faced challenges in SOC. We are convinced that with focused efforts to extend the approach according to the recommendations in Sec. 9.2 it can evolve to completely fulfill the vision of OTF computing.

Bibliography

[1] Aguirre, J., Eckert, K., Euzenat, J., Ferrara, A., van Hage, W. R., Hollink, L., Meilicke, C., Nikolov, A., Ritze, D., Scharffe, F., Shvaiko, P., Sváb-Zamazal, O., dos Santos, C. T., Jiménez-Ruiz, E., Grau, B. C., and Zapilko, B. (2012). Results of the ontology alignment evaluation initiative 2012. In *Proceedings of the 7th International Workshop on Ontology Matching*, volume 946 of *CEUR Workshop Proceedings*, Boston, MA, USA. CEUR-WS.org.

[2] Allen, R. and Garlan, D. (1997). A Formal Basis for Architectural Connection. *ACM Transaction on Software Engineering and Methodology*, 6(3):213–249.

[3] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, Berlin Heidelberg.

[4] Arabshian, K., Danielsen, P. J., and Afroz, S. (2012). Lexont: A semi-automatic ontology creation tool for programmable web. In *AAAI Spring Symposium: Intelligent Web Services Meet Social Computing*, volume SS-12-04 of *AAAI Technical Report*. AAAI.

[5] Atkinson, C., Bayer, J., and Muthig, D. (2000). Component-based product line development: The kobra approach. In *Software Product Lines*, volume 576 of *The Springer International Series in Engineering and Computer Science*, pages 289–309. Springer US.

[6] Avison, D. and Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools (3rd edition)*. McGraw Hill.

[7] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] Bansal, A., Kona, S., Simon, L., Mallya, A., Gupta, G., and Hite, T. (2005). A universal service-semantics description language. In *Proceedings of Third IEEE European Conference on Web Services (ECOWS'05)*, pages 214–225, Orlando, FL, USA. IEEE Computer Society.

[9] Bartalos, P. and Bieliková, M. (2011). Automatic Dynamic Web Service Composition: A Survey and Problem Formalization. *Computing and Informatics*, 30(4):793–827.

[10] Bastide, R. and Barboni, E. (2004). Component-based behavioural modelling with high-level petri nets. In *Proceedings of Third Workshop on Modelling of Objects, Components and Agents (MOCA'04)*, pages 37–46. DAIMI.

[11] Becker, S., Brogi, A., Gorton, I., Overhage, S., Romanovsky, A., and Tivoli, M. (2006). Towards an engineering approach to component adaptation. In *Architecting Systems with Trustworthy Components*, volume 3938 of *LNCS*, pages 193–215. Springer, Berlin Heidelberg.

[12] Becker, S., Koziolek, H., and Reussner, R. (2009). The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22.

[13] Bellur, U. and Mande, T. (2009). Automated web service composition using semantic descriptions. In *Proceedings of 4th IEEE Asia-Pacific Services Computing Conference (APSCC'09)*, pages 377–384, Singapore. IEEE Computer Society.

[14] Bellur, U. and Vadodaria, H. (2008). On extending semantic matchmaking to include preconditions and effects. In *Proceedings of IEEE International Conference on Web Services (ICWS'08)*, pages 120–128, Beijing, China. IEEE Computer Society.

[15] Bener, A. B., Ozadali, V., and Ilhan, E. S. (2009). Semantic matchmaker with precondition and effect matching using SWRL. *Expert Systems with Applications*, 36(5):9371–9377.

[16] Bin, X., Yan, W., Po, Z., and Juanzi, L. (2005). Web services searching based on domain ontology. In *Proceedings of IEEE Seventh International Workshop on Service-Oriented System Engineering (SOSE'05)*, pages 51–55, Beijing, China. IEEE Computer Society.

[17] Bloomer, J. (1992). *Power Programming with RPC*. O'Reilly & Associates, Inc.

[18] Bolton, F. and Walshe, E. (2001). *Pure CORBA: A Code-Intensive Premium Reference*. SAMS Publishing, Indianapolis, IN, USA.

[19] Bondarenko, A., Dau, D., Feldkord, S., Gellermann, M., Gerth, C., Hornkamp, M., Mhlenfeld, B., and Qiu, H. (2006). Abschlussbericht:Patternbasierte Geschftsprozess-Modellierung (PaGeMo). Technical report, University of Paderborn, Germany.

[20] Borger, E. and Stark, R. F. (2003). *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, Berlin Heidelberg.

[21] Box, D. (1997). *Essential COM*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

[22] Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition.

[23] Bran, S. and Gérard, S. (2014). *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE*. Elsevier, Amsterdam.

[24] Bröggelwirth, J. (2014). Design of Real-Time Coordination Protocols for Multilateral Communication. Master's thesis, University of Paderborn.

[25] Brogi, A., Corfini, S., Aldana, J. F., and Navas, I. (2006). Automated discovery of compositions of services described with separate ontologies. In *Proceedings of 4th International Conference on Service-Oriented Computing (ICSOC'06)*, volume 4294 of *LNCS*, pages 509–514, Chicago, IL, USA. Springer-Verlag Berlin Heidelberg.

[26] Brogi, A., Corfini, S., and Popescu, R. (2008). Semantics-based Composition-oriented Discovery of Web Services. *ACM Transactions on Internet Technology*, 8(4):19:1–19:39.

[27] Brown, A. W. and Wdlnau, K. C. (1996). Engineering of component-based systems. In *Proceedings of 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'96)*, pages 414–422, Montreal, Canada. IEEE Computer Society.

[28] Bures, T., Hnetynka, P., and Plasil, F. (2006). Sofa 2.0: Balancing advanced features in a hierarchical component model. In *Proceedings of Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, pages 40–48, Prague, Czech Republic. Springer-Verlag Berlin Heidelberg.

[29] Cavallaro, L., Nitto, E., and Pradella, M. (2009). An Automatic Approach to Enable Replacement of Conversational Services. In *Proceedings of 7th International Joint Conference ICSOC-ServiceWave '09*, volume 5900 of *LNCS*, pages 159–174, Budva, Montenegro. Springer-Verlag Berlin Heidelberg.

[30] Chabeb, Y. and Tata, S. (2008). Yet another semantic annotation for WSDL. In *Proceedings of the IADIS International Conference WWW/Internet (ICWI'08)*, pages 462–467, Freiburg, Germany. IADIS.

[31] Chabeb, Y., Tata, S., and Ozanne, A. (2010). Yasa-m: A semantic web service matchmaker. In *Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications (AINA'10)*, pages 966–973, Perth, Australia. IEEE Computer Society.

[32] Cicalese, C. D. T. and Rotenstreich, S. (1999). Behavioral specification of distributed software component interfaces. *IEEE Computer*, 32(7):46–53.

[33] Corradini, A., Montanari, U., and Rossi, F. (1996). Graph processes. *Fundamenta Informaticae*, 26(3):241–265.

[34] Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., and Löwe, M. (1997). Algebraic approaches to graph transformation, part i: Basic concepts and double pushout approach. In *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, pages 163–245. World Scientific.

[35] Corrales, J. C., Grigori, D., Bouzeghoub, M., and Burbano, J. E. (2008). Bematch: A platform for matchmaking service behavior models. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'08)*, pages 695–699, Nantes, France. ACM, New York, USA.

[36] Crnković, I., Sentilles, S., Vulgarakis, A., and Chaudron, M. (2011). A classification framework for software component models. *IEEE Transactions on Software Engineering*, 37(5):593–615.

[37] Cubo, J., Canal, C., and Pimentel, E. (2010). Context-aware service discovery and adaptation based on semantic matchmaking. In *Proceedings of Fifth International Conference on Internet and Web Applications and Services (ICIW'10)*, pages 554–561, Barcelona, Spain. IEEE Computer Society.

[38] Cuzzocrea, A. and Fisichella, M. (2011). Discovering semantic web services via advanced graph-based matching. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC'11)*, pages 608–615, Anchorage, Alaska, USA. IEEE Computer Society.

[39] de Bruijn, J., Lausen, H., Polleres, A., and Fensel, D. (2006). The Web Service Modeling Language WSML: An Overview. In *Proceedings of 3rd European Semantic Web Conference (ESWC'06)*, volume 4011 of *LNCS*, pages 590–604. Springer-Verlag Berlin Heidelberg.

[40] DeMarco, T. (1978). *Structured Analysis and System Specification*. Yourdon Press, New York, USA.

[41] Digital Enterprise Research Institute (2005). OnTour - The Tourism Ontology. http://e-tourism.deri.at/ont/.

[42] Dobing, B. and Parsons, J. (2005). Current practices in the use of uml. In *Perspectives in Conceptual Modeling*, volume 3770 of *LNCS*, pages 2–11. Springer-Verlag, Berlin Heidelberg.

[43] Domínguez, E., Lloret, J., Pérez, B., Rodríguez, Á., Rubio, Á. L., and Zapata, M. A. (2007). A survey of uml models to xml schemas transformations. In *Proceedings of 8th International Conference on Web Information Systems Engineering (WISE'07)*, volume 4831 of *LNCS*, pages 184–195, Nancy, France. Springer-Verlag Berlin Heidelberg.

[44] D'Souza, D. F. and Wills, A. C. (1998). *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Professional.

[45] E. Bruneton, T. Coupaye, J. S. (2004). The Fractal Component Model Specification. http://fractal.ow2.org/specification/.

[46] Ehrig, H., Engels, G., and Rozenberg, G., editors (1999). *Handbook of Graph Grammars and Computing by Graph Transformation: applications, languages and tools. Vol. 2*. World Scientific.

[47] EJB Expert Group (2006). JSR 220: Enterprise JavaBeans 3.0 (Final Release). https://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

[48] Engels, G., Gogolla, M., Hohenstein, U., Hülsmann, K., Löhr-Richter, P., Saake, G., and Ehrich, H. (1992). Conceptual modelling of database applications using extended ER model. *Data Knowledge Engineering*, 9:157–204.

[49] Engels, G., Güldali, B., Soltenborn, C., and Wehrheim, H. (2007). Assuring Consistency of Business Process Models and Web Services Using

Visual Contracts. In *Proceedings of Applications of Graph Transformations with Industrial Relevance (AGTIVE'07)*, volume 5088 of *LNCS*, pages 17–31, Kassel, Germany. Springer Berlin Heidelberg.

[50] Engels, G., Küster, J. M., Heckel, R., and Groenewegen, L. (2001). A methodology for specifying and analyzing consistency of object-oriented behavioral models. *SIGSOFT Software Engineering Notes*, 26(5):186–195.

[51] ESSI WSMO Working Group (2005). Web Service Modelling Ontology. http://www.wsmo.org/.

[52] Euzenat, J. and Shvaiko, P. (2007). *Ontology Matching*, volume 18. Springer-Verlag, Berlin Heidelberg.

[53] Faber, P., Mairal, R., and Magana, P. (2011). Linking a domain-specific ontology to a general ontology. In *Proceedings of the Twenty-Fourth International Florida Artificial Intelligence Research Society (FLAIRS'11)*, Palm Beach, Florida, USA. AAAI Press.

[54] Förster, A. (2009). *Pattern based business process design and verification*. PhD thesis, University of Paderborn.

[55] Georgakopoulos, D. and Papazoglou, M. P. (2008). *Service-Oriented Computing*. The MIT Press.

[56] Gerth, C. (2013). *Business Process Models - Change Management*. PhD thesis, University of Paderborn, Springer, Berlin Heidelberg, LNCS, vol. 7849.

[57] Gil, A. F., Cong, Z., and Baltá, A. (2012). Bridging the gap between service description models in service matchmaking. *Multiagent and Grid Systems*, 8(1):83–103.

[58] Girault, C. and Valk, R. (2001). *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag Berlin Heidelberg.

[59] Grigori, D., Corrales, J., Bouzeghoub, M., and Gater, A. (2010). Ranking bpel processes for service discovery. *IEEE Transactions on Services Computing*, 3(3):178–192.

[60] Grigori, D., Corrales, J. C., and Bouzeghoub, M. (2008). Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems*, 33(7–8):681–698.

[61] Haller, A., Cimpian, E., Mocan, A., Oren, E., and Bussler, C. (2005). WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of IEEE International Conference on Web Services (ICWS'05)*, pages 321–328, Orlando, FL,USA. IEEE Computer Society.

[62] Hanninen, K., Maki-Turja, J., Nolin, M., Lindberg, M., Lundback, J., and Lundback, K.-L. (2008). The rubus component model for resource constrained real-time systems. In *Proceedings of International Symposium on Industrial Embedded Systems (SIES'08)*, pages 177–183, Montpellier / La Grande Motte, France. IEEE Computer Society.

[63] Hatcliff, J., Leavens, G. T., Leino, K. R. M., Müller, P., and Parkinson, M. (2012). Behavioral interface specification languages. *ACM Computing Surveys*, 44(3):16:1–16:58.

[64] Hausmann, J. H. (2005). *Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages*. PhD thesis, University of Paderborn.

[65] Hausmann, J. H., Heckel, R., and Lohmann, M. (2005). Model-based Development of Web Service Descriptions Enabling a Precise Matching Concept. *International Journal of Web Services Research*, 2(2):67–85.

[66] Heckel, R., Corradini, A., Ehrig, H., and Löwe, M. (1996). Horizontal and vertical structuring of typed graph transformation systems. *Mathematical Structures in Computer Science*, 6(6):613–648.

[67] Heckel, R., Ehrig, H., Wolter, U., and Corradini, A. (2001). Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. *Applied Categorical Structures*, 9(1):83–110.

[68] Hepp, M. (2008). Goodrelations: An ontology for describing products and services offers on the web. In Gangemi, A. and Euzenat, J., editors, *Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 329–346. Springer Berlin Heidelberg.

[69] Hewlett-Packard Company (2002). Web Services Conversation Language (WSCL) 1.0. http://www.w3.org/TR/wscl10/.

[70] Hissam, S. A., Ivers, J., Plakosh, D., and Wallnau, K. C. (2005). Pin component technology (v1.0) and its c interface. Technical report, Carnegie Mellon University, Software Engineering Institute.

[71] Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677.

[72] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). Semantic Web Rule Language (SWRL). http://www.w3.org/Submission/SWRL/.

[73] Huma, Z., Gerth, C., and Engels, G. (2014). On-the-fly computing: automatic service discovery and composition in heterogeneous domains. *Computer Science - Research and Development*, pages 1–29.

[74] Huma, Z., Gerth, C., Engels, G., and Juwig, O. (2012a). Towards an Automatic Service Discovery for UML-based Rich Service Descriptions. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS'12)*, volume 7590 of *LNCS*, pages 709–725, Innsbruck, Austria. Springer-Verlag, Berlin Heidelberg.

[75] Huma, Z., Gerth, C., Engels, G., and Juwig, O. (2012b). UML-based Rich Service Description and Discovery in Heterogeneous Domains. In *Proceedings of the Forum at the CAiSE'12 Conference on Advanced Information Systems Engineering*, volume 855 of *CEUR Workshop Proceedings*, pages 90–97, Danzig, Poland. CEUR-WS.org.

[76] Huma, Z., Gerth, C., Engels, G., and Juwig, O. (2013). Automated service composition for on-the-fly soas. In *Proceedings of the 11th International Conference on Service Oriented Computing (ICSOC'13)*, volume 8274 of *LNCS*, pages 524–532, Berlin, Germany. Springer-Verlag, Berlin Heidelberg.

[77] International Business Machines Corporation (IBM) (2007). Business Process Execution Language for Web Services. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

[78] International Business Machines Corporation (IBM) (2009). Rational Software Architect RealTime Edition. http://www-947.ibm.com/support/entry/portal/product/rational/rational_software_architect_realtime_edition?productContext=443691142.

[79] Ke, C. and Huang, Z. (2012). Self-adaptive semantic web service matching method. *Knowledge-Based Systems*, 35:41–48.

[80] Kifer, M. and Lausen, G. (1989). F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. *SIGMOD Records*, 18(2):134–146.

[81] Klusch, M., Fries, B., and Sycara, K. (2009a). Owls-mx: A hybrid semantic web service matchmaker for owl-s services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):121–133.

[82] Klusch, M. and Kapahnke, P. (2012a). Adaptive signature-based semantic selection of services with owls-mx3. *Multiagent Grid Systems*, 8(1):69–82.

[83] Klusch, M. and Kapahnke, P. (2012b). The isem matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(0):1–14.

[84] Klusch, M. and Kapahnke, P. (2012c). The isem matchmaker: A flexible approach for adaptive hybrid semantic service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, 15(3).

[85] Klusch, M., Kapahnke, P., and Zinnikus, I. (2009b). Hybrid adaptive web service selection with sawsdl-mx and wsdl-analyzer. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications (ESWC'09)*, ESWC 2009 Heraklion, pages 550–564, Heraklion, Crete, Greece. Springer-Verlag, Berlin, Heidelberg.

[86] Klusch, M. and Kaufer, F. (2009). Wsmo-mx: A hybrid semantic web service matchmaker. *Web Intelligence and Agent Systems*, 7(1):23–42.

[87] Kokash, N. (2006). A Comparison of Web Service Interface Similarity Measures. In *Proceedings of the Third Starting AI Researchers' Symposium (STAIRS'06)*, volume 142 of *Frontiers in Artificial Intelligence and Applications*, pages 220–231, Trentino, Italy. IOS Press.

[88] Kona, S., Bansal, A., Blake, M. B., and Gupta, G. (2008). Generalized Semantics-Based Service Composition. In *Proceedings of IEEE International Conference on Web Services (ICWS'08)*, pages 219–227, Beijing, China. IEEE Computer Society.

[89] Kopecký, J., Moran, M., Vitvar, T., Roman, D., and Mocan, A. (2007). WSMO Grounding. http://www.wsmo.org/TR/d24/d24.2/v0.1/.

[90] Kotonya, G., Sommerville, I., and Hall, S. (2003). Towards a classification model for component-based software engineering research. In *Proceedings of 29th Euromicro Conference*, pages 43–52, Belek-Antalya, Turkey. IEEE Computer Society.

[91] Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benrernou, S., Brandic, I., Kertész, A., Parkin, M., and Carro, M. (2013). A survey on service quality description. *ACM Computing Surveys*, 46(1):1:1–1:58.

[92] Küster, J. and Stroop, J. (2001). Consistent Design of Embedded Real-time Systems with UML-RT. In *Proceedings of Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 31–40, Magdeburg, Germany. IEEE Computer Society.

[93] Küster, J. M. and Stehr, J. (2003). Towards Explicit Behavioral Consistency Concepts in the UML. In *Proceedings of the 2nd International Workshop on Scenarios and State Machines: Models, Algorithms and Tools*, Portland, USA. IEEE Computer Society.

[94] Leymann, F. (2005). The (Service) Bus: Services Penetrate Everyday Life. In *Proceedings of International Conference on Service-Oriented Computing (ICSOC'05)*, volume 3826 of *LNCS*, pages 12–20. Springer-Verlag, Berlin Heidelberg, Berlin, Heidelberg.

[95] Liu, F., Shi, Y., Yu, J., Wang, T., and Wu, J. (2010). Measuring Similarity of Web Services Based on WSDL. In *Proceedings of IEEE International Conference on Web Services (ICWS'10)*, pages 155–162, Miami, Florida, USA. IEEE Computer Society.

[96] Lohmann, M. (2006). *Kontraktbasierte Modellierung, Implementierung und Suche von Komponenten in serviceorientierten Architekturen*. PhD thesis, University of Paderborn.

[97] LSDIS Lab (2004a). METEOR-S. http://lsdis.cs.uga.edu/projects/meteor-s/.

[98] LSDIS Lab (2004b). Web Service Semantics. http://lsdis.cs.uga.edu/projects/WSDL-S/wsdl-s.pdf.

[99] Manickam, P., Sangeetha, S., and Subrahmanya, S. V. (2013). *Component- Oriented Development and Assembly: Paradigm, Principles, and Practice Using Java*. Auerbach Publications, Boston, MA, USA.

[100] Masuch, N., Hirsch, B., Burkhardt, M., Hessler, A., and Albayrak, S. (2012). SeMa2:A Hybrid Semantic Service Matching Approach. In *Semantic Web Services*, pages 35–47. Springer Berlin Heidelberg.

[101] Mätzel, K.-U. and Schnorf, P. (1997). Dynamic component adaptation. Technical report, Ubilab Technical Report 97.6.

[102] Mens, T. (2002). A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 28(5):449–462.

[103] Microsoft (2000). Microsoft UDDI registry. http://uddi.microsoft.com/.

[104] Miller, G. A. (1995). Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41.

[105] Moffett, Y., Beaulieu, A., and Dingel, J. (2011). Verifying UML-RT Protocol Conformance Using Model Checking. In *Proceedings of 14th International Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, volume 6981 of *LNCS*, pages 410–424, Wellington, New Zealand. Springer-Verlag, Berlin Heidelberg.

[106] Motahari Nezhad, H. R., Benatallah, B., Martens, A., Curbera, F., and Casati, F. (2007). Semi-automated Adaptation of Service Interactions. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*, pages 993–1002, Banff, Alberta, Canada. ACM.

[107] Motahari Nezhad, H. R., Xu, G. Y., and Benatallah, B. (2010). Protocol-aware Matching of Web Service Interfaces for Adapter Development. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*, pages 731–740, Raleigh, North Carolina, USA. ACM.

[108] Naeem, M., Heckel, R., Orejas, F., and Hermann, F. (2010). Incremental Service Composition based on Partial Matching of Visual Contracts. In *Proceedings of Fundamental Approaches to Software Engineering (FASE'10)*, volume 6013 of *LNCS*, pages 123–138, Paphos, Cyprus. Springer-Verlag, Berlin Heidelberg.

[109] Navas, I., Sanz, I., Aldana, J., and Berlanga, R. (2005). Automatic generation of semantic fields for resource discovery in the semantic web. In Andersen, K., Debenham, J., and Wagner, R., editors, *Database and Expert Systems Applications*, volume 3588 of *LNCS*, pages 706–715. Springer-Verlag, Berlin Heidelberg.

[110] Nesterow, V. (2010). Eine formale, graphbasierte Semantik für UML Statemachines. Master's thesis, University of Paderborn.

[111] OASIS (2005). OASIS SOA Reference Model Technical Committee. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[112] OASIS (2007). Web Services Business Process Execution Language 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf.

[113] Object Management Group (OMG) (2006). Component Object Request Broker Architecture (CORBA) Component Model. http://www.omg.org/spec/CCM/4.0/.

[114] Object Management Group (OMG) (2008). UML Profile for Modeling Quality of Service and Fault Tolerance (QFTP) – 1.1. http://www.omg.org/spec/QFTP/1.1/.

[115] Object Management Group (OMG) (2009). Unified Modeling Language (UML) – Superstructure, Version 2.3. http://www.omg.org/spec/UML/2.3/Infrastructure.

[116] Object Management Group (OMG) (2011a). Business Process Model and Notation (BPMN) 2.0. http://www.omg.org/spec/BPMN/2.0/.

[117] Object Management Group (OMG) (2011b). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1. http://www.omg.org/spec/QVT/1.1/PDF/.

[118] OpenTravel Alliance (2014). OTA XML Schema. http://www.opentravel.org/Specifications/OnlineXmlSchema.aspx.

[119] OSGi Alliance (2007). OSGi Service Platform Core Specification. http://www.osgi.org/Specifications/HomePage.

[120] O'Sullivan, J. J. (2006). *Towards a precise understanding of service properties*. PhD thesis, Queensland University of Technology.

[121] OWL-S Coalition (2006). OWL-based Web Service Ontology. http://www.ai.sri.com/daml/services/owl-s/1.2/.

[122] Paliwal, A., Shafiq, B., Vaidya, J., Xiong, H., and Adam, N. (2012). Semantics-based automated service discovery. *IEEE Transactions on Services Computing*, 5(2):260–275.

[123] Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic Matching of Web Services Capabilities. In *Proceedings of the First International Semantic Web Conference (ISWC'02)*, pages 333–347, Sardinia, Italy. Springer-Verlag, Berlin Heidelberg.

[124] Pathak, J., Basu, S., and Honavar, V. (2006a). Modeling Web Service Composition using Symbolic Transition Systems. In *Proceedings of AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing*, Boston, Massachusetts, USA. AAAI Press.

[125] Pathak, J., Basu, S., and Honavar, V. (2006b). Modeling Web Services by Iterative Reformulation of Functional and Non-functional Requirements. In *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC'06)*, volume 4294 of *LNCS*, pages 314–326, Chicago, IL, USA. Springer-Verlag, Berlin Heidelberg.

[126] Platenius, M. C. (2013). Fuzzy service matching in on-the-fly computing. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13)*, pages 715–718, Saint Petersburg, Russian Federation. ACM.

[127] Plebani, P. and Pernici, B. (2009). Urbe: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642.

[128] Poizat, R. M. P. and Salaün, G. (2008). Adaptation of Service Protocols Using Process Algebra and On-the-Fly Reduction Techniques. In Bouguettaya, A., Krüger, I., and Margaria, T., editors, *Proceedings of International Conference on Service Oriented Computings (ICSOC'08)*, volume 5364 of *LNCS*, pages 84–99, Sydney, Australia. Springer-Verlag, Berlin Heidelberg.

[129] PrgrammableWeb (2005). PrgrammableWeb: Online Web Services Search Engine. http://www.programmableweb.com/.

[130] Princeton University (2010). WordNet: A Lexical Database for English. http://wordnet.princeton.edu/.

[131] Rao, J. and Su, X. (2004). A Survey of Automated Web Service Composition Methods. In *Proceedings of the First International Conference on Semantic Web Services and Web Process Composition (SWSWPC'04)*,

volume 3387, pages 43–54, San Diego, CA, USA. Springer-Verlag, Berlin Heidelberg.

[132] Reussner, R. H., Becker, S., and Firus, V. (2004). Component Composition with Parametric Contracts. In *Tagungsband der Net.ObjectDays 2004*, pages 155–169.

[133] Roshandel, R. and Medvidovic, N. (2003). Modeling multiple aspects of software components. In *Proceedings of 3rd Workshop on Specification and Verification of Component-based Systems(SAVCBS'03)*, pages 88–91, Helsinki, Finland. Addison-Wesley Longman Publishing.

[134] Rozenberg, G., editor (1997). *Handbook of Graph Grammars and Computing by Graph Transformations: Foundations. Vol. 1.* World Scientific.

[135] Sabou, M., Richards, D., and van Splunter, S. (2003). An experience report on using DAML-S. In *Proceedings of Workshop on E-Services and the Semantic Web (ESSW '03)*, Budapest, Hungary. ACM Press.

[136] SAWSDL Working Group (2007). Semantic Annotations for WSDL and XML Schema (SAWSDL). http://www.w3.org/TR/2007/REC-sawsdl-20070828/.

[137] Schäfer, J. (2010). Eine formale, graphbasierte Semantik für UML Interactions. Master's thesis, University of Paderborn.

[138] Schäfer, T., Knapp, A., and Merz, S. (2001). Model Checking UML State Machines and Collaborations. *Electronic Notes in Theoretical Computer Science*, 55(3):357–369.

[139] Schmidt, H. W. and Reussner, R. H. (2002). Generating adapters for concurrent component protocol synchronisation. In *Proceedings of the IFIP TC6/WG6.1 5th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'02)*, FMOODS'02, pages 213–229. Kluwer, B.V.

[140] Schwichtenberg, S. (2013). Ontology-based Normalization and Matching of Rich Service Descriptions. Master's thesis, University of Paderborn.

[141] Schwichtenberg, S., Gerth, C., Huma, Z., and Engels, G. (2014). Normalizing Heterogeneous Service Description Models with Generated QVT

Transformations. In *Proceedings of 10th European Conference on Modelling Foundations and Applications (ECMFA'14)*, volume 8569 of *LNCS*, pages 180–195, York, UK. Springer-Verlag, Berlin Heidelberg.

[142] Shigo, O., Okawa, A., and Kato, D. (2006). Constructing Behavioral State Machine using Interface Protocol Specification. In *Proceedings of the XIII Asia Pacific Software Engineering Conference (APSEC '06)*, pages 191–198. IEEE Computer Society.

[143] Smullyan, R. (1995). *First-order Logic*. Dover Books on Mathematics Series. Dover Publications.

[144] Snell, J., Tidwell, D., and Kulchenko, P. (2002). *Programming Web services with SOAP*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

[145] Spanoudaki, G. and Zisman, A. (2010). Discovering Services during Service-Based System Design Using UML. *IEEE Transactions on Software Engineering*, 36(3):371–389.

[146] Spivey, J. M. (1988). *Understanding Z: A Specification Language and Its Formal Semantics*. Cambridge University Press, New York, NY, USA.

[147] Stroulia, E. and Wang, Y. (2005). Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14(4):407–438.

[148] Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data and Knowledge Engineering*, 25(1-2):161–197.

[149] Sun Microsystems (1997). Javabeans Specification. `java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html`.

[150] Syeda-Mahmood, T., Shah, G., Akkiraju, R., Ivan, A.-A., and Goodwin, R. (2005). Searching service repositories by combining semantic and ontological matching. In *Proceedings of IEEE International Conference on Web Services (ICWS'05)*, pages 13–20 vol.1, Orlando, FL, USA. IEEE Computer Society.

[151] Szlenk, M. (2006). Formal semantics and reasoning about uml class diagram. In *Proceedings of International Conference on Dependability of Computer Systems (DepCos-RELCOMEX'06)*, pages 51–59, Szklarska Poreba. IEEE Computer Society.

[152] Tarjan, R. E. (1983). *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[153] Tran, V. X., Puntheeranurak, S., and Tsuji, H. (2009). A new service matching definition and algorithm with SAWSDL. In *Proceedings of 3rd IEEE International Conference on Digital Ecosystems and Technologies (DEST'09)*, pages 371–376, Istanbul, Turkey. IEEE Computer Society.

[154] Tsui, F. and Karam, O. (2010). *Essentials of Software Engineering*. Jones & Bartlett Learning.

[155] United States Census Bureau (2012). North American Industry Classification System(NAICS). https://www.census.gov/eos/www/naics/2012NAICS/2012_Definition_File.pdf.

[156] University, K. S. (2006). Knowledge Interchange Format (KIF). http://www-ksl.stanford.edu/knowledge-sharing/kif/#manual.

[157] Vaculin, R., Neruda, R., and Sycara, K. (2009). The process mediation framework for semantic web services. *International Journal on Agent-Oriented Software Engineering*, 3(1):27–58.

[158] W3C (2004a). OWL Web Ontology Language. http://www.w3.org/TR/owl-features/.

[159] W3C (2004b). RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/.

[160] W3C (2004c). Web Service Architecture. http://www.w3.org/TR/ws-arch/.

[161] W3C (2005). Web Services Choreography Description Language 1.0. http://www.w3.org/TR/ws-cdl-10/.

[162] W3C (2007). Web Service Description Language(WSDL). http://www.w3.org/TR/wsdl20/.

[163] Wache, H., Voegele, T., Visser, T., Stuckenschmidt, H., Schuster, H., Neumann, G., and Huebner, S. (2001). Ontology-based integration of information - a survey of existing approaches. In *Proceedings of Workshop on Ontologies and Information at the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 108–117, Seattle, USA. Morgan Kaufmann.

[164] Warmer, J. and Kleppe, A. (1999). *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, Reading, MA.

[165] Webber, J. (2010). *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly, Beijing, China.

[166] Yellin, D. M. and Strom, R. E. (1997). Protocol Specifications and Component Adaptors. *ACM Transaction on Programming Languages and Systems*, 19(2):292–333.

List of Figures