Automated Service Discovery and Composition for On-the-Fly SOAs**

Technical Report

tr-ri-13-333

Zille Huma[1], Christian Gerth[1], Gregor Engels[1], and Oliver Juwig[2]

[1] Database and Information Systems Group, Department of Computer Science
University of Paderborn
Zukunftsmeile 1
33102 Paderborn, Germany

{zille.huma,gerth,engels}@upb.de

[2] HRS-Hotel Reservation Service, Germany

Oliver.Juwig@hrs.de

Version 1.0

Paderborn July 1, 2013

** This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre \On-The-Fly Computing" (SFB 901)

Abstract. The true essence of service-oriented computing is the realization of software development based on services that are composed to service-oriented architectures (SOA). In recent years, the number of available software services steadily increased, favored by the rise of cloud computing with its attached delivery models like Software-as-a-Service (SaaS). To fully leverage the opportunities provided by these services for developing highly flexible and aligned SOA, integration of new services as well as the substitution of existing services must be simplified. As a consequence, approaches for automated and accurate service discovery and composition are needed. In this paper, we propose an automatic service composition approach as an extension to our earlier work on automatic service discovery. To ensure accurate results, it matches the service request and available offers based on their structural as well as behavioral aspects. Afterwards, possible service compositions are determined by composing the service protocols through a composition strategy based on labeled transition systems.

1 Introduction

Service-oriented computing (SOC) has emerged as a promising trend to enable the vision of large-scale, heterogeneous and flexible software systems at enterprise level. This vision is realized through the notion of service-oriented architecture (SOA) with services as its basic building blocks. In this context, a SOA developer acts as a service requestor, who wants to discover and compose services available on service markets based on his/her *service request*. As a counterpart, services are developed and published on service markets by service providers in terms of *service offers*. This scenario is sketched in Figure 1.

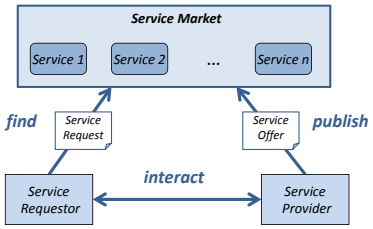

Fig. 1. An Overview of Service Publishing and Discovery

With the advent of cloud computing and its attached delivery models, a steady increase in the number of available services can be observed. This growing plethora of available services provides enormous opportunities for the development of future SOAs towards software systems that are highly flexible and can be aligned more easily to meet constantly changing requirements. Such *On-The-Fly* SOAs could, e.g., maintain their quality autonomously by substituting disused services with improved ones or their functionality could be extended on-demand by integrating new services. To make this vision come true, accurate and automated service discovery and composition mechanisms are needed that have to face several challenges.

First of all, to enable an efficient and precise identification, services must be described in a suitable way by a service specification that comprises structural as well as behavioral aspects of requested and offered services.

Secondly, service discovery mechanisms must deal with the existing multifaceted heterogeneity of the involved service partners. For example, most likely service requestors and providers will work using their own independent ontologies. Thus, service requests and offers may be structurally different but at the same time semantically similar. This semantic similarity must be identified. Additionally, service partners may use different languages/notations to specify their service descriptions and these linguistic differences have to be resolved to match these descriptions. Moreover, based on their independent domain knowledge, service partners may specify service requests/offers at different granularity levels leading to complex n:m correspondences between requested and offered operations. As a consequence, an elaborate mechanism for automatic service discovery and composition has to consider structural as well as behavioral aspects of service partners while overcoming their multifaceted heterogeneity. For this purpose, we proposed an automatic service discovery mechanism for RSDL-based service descriptions [9]. An application scenario for our proposed approach came from our industrial partners Hotel Reservation Service (HRS)[1]. In this scenario, potential new hotel services shall be automatically discovered and connected to provide end users with booking facilities for these hotels.

In this paper, we extend our approach by a service composition mechanism, which enables the composition of *multiple* services each offering various operations in order to fulfill a service request. Thereby, we address in particular the challenge that in a realistic scenario, a service request typically cannot be satisfied by a single service offer. Our proposed mechanism ensures precise service composition results as it comprehensively covers different elements in service offers and requests, such as operation signatures, operation semantics (pre- and post-conditions), and service protocols to discover and compose potential service offers that satisfy a request.

The remainder of the paper is structured as follows: In Section 2, we introduce our Rich Service Description Language (RSDL), which provides comprehensive notations for service request and offer specification. In Sections 3 and 4, we describe our elaborate approach to service discovery and service composition in detail. Section 5 briefly introduces our tool support. In Section 6, we discuss related work and finally, we conclude the paper and give an outlook on future work in Section 7.

2 Rich Service Description Language

To realize our vision of a comprehensive service specification, we proposed a UML-based rich service description language (RSDL) [10]. Our RSDL provides notations to describe the structure and the behavior of service requests and offers using the following artifacts:

– *Structural description*: We specify operation signatures of a service request/offer using the Web Service Description Language (WSDL) [23].

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

– *Behavioral description* comprises two aspects:
  1. Operation semantics, i.e., pre- and post-conditions for individual operations are specified using UML-based visual contracts (VC) [13]. A VC describes the system state before and after the invocation of an operation in terms of UML object diagrams that are typed over the ontologies of the service partners.
  2. A service protocol specifies required/allowed operation invocation sequences using UML sequence diagrams and UML statechart diagrams for requestors and providers, respectively.

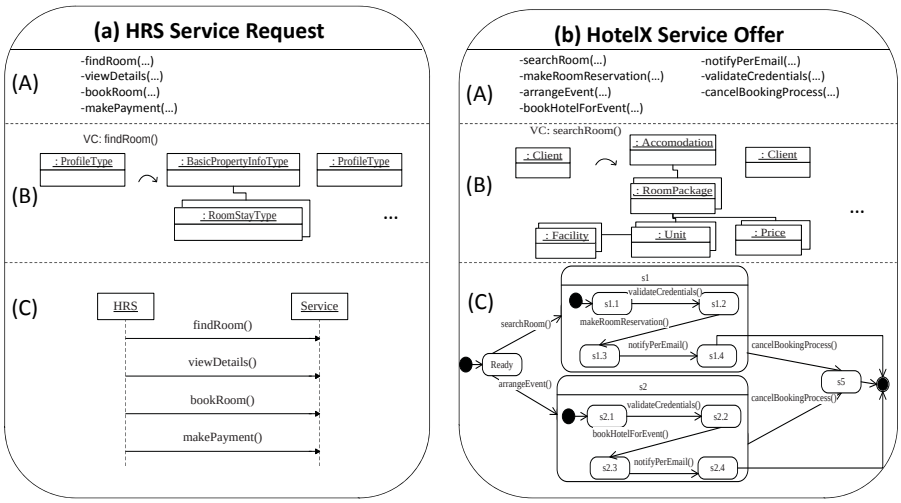

Fig. 2. (a) RSDL-based Service Request of HRS and (b) RSDL-based Service Offer of HotelX

Figure 2(a) shows a RSDL-based *service request* of HRS consisting of *(A)*: operation signatures, i.e., *findRoom()*, *viewDetails()*, *bookRoom()*, *makePayment()*, *(B)*: VCs of these operations typed over the underlying Open Travel Alliance (OTA)[2]-based ontology of HRS, and *(C)*: a requestor protocol as UML sequence diagram. In the context of a service protocol, we assume that a service requestor is interested to specify a *single* invocation sequence of the operations invoked on the offered services. We argue that UML sequence diagrams are a suitable choice for this purpose.

Similarly, Figure 2(b) shows a RSDL-based *service offer* of the hotel service *HotelX*. The specification consists of *(A)*: operation signatures *searchRoom()*, *makeRoomReservation()*, ... , *(B)*: VCs typed over the HarmoNET[3]-based ontology of HotelX, and *(C)*: a provider protocol as UML statechart diagram. In case of a service offer, multiple invocation sequences of provided operations shall be possible, which we specify using a UML statechart diagram.

[2] http://www.opentravel.org
[3] http://www.harmonet.org

We have certain assumptions for service protocols of service partners. Firstly, the requestor protocol comprises a single invocation sequence without any loops. Similarly, the provider protocols also does not contain any loops. Additionally, the provider protocol must have a final state representing the particular point where the service provider expects an invocation sequence to end.

Based on *rich* service request and offers, an automated service discovery and composition can be achieved. In the next section, we describe our service discovery and composition mechanism for such RSDL-based service requests and offers.

3 Automated Service Discovery

In the following, we describe an automated mechanism for service discovery and composition. In this section, an overview of our approach and a description of the service discovery is given. In Section 4, we then address the composition of services.

3.1 Overview of the Proposed Approach

Figure 3 gives an overview of the proposed multi-step approach. It is initiated, when a requestor accesses the service market to discover and compose potential offers to satisfy his/her request. In the *request normalization* phase (Step 1 & 2), the request is normalized to make it comparable to the available service offers. For this purpose, the requestor's local ontology is manually mapped to a global ontology agreed upon by all service partners. Similarly, a service provider manually maps his/her local ontology to this global ontology while publishing a service on the service market. Such a global ontology is a necessity to enable automatic service discovery and we assume that it is a part of the service market and is maintained by the market provider. Ontology mapping is an extensively-researched complex problem, which is not the focus of our work and we rely on existing work for this purpose. In Step 2, the operation semantics described in terms of visual contracts (VC) in the service request are automatically normalized to a public representation using the local-global ontology mappings. Here it is important to mention that the first three steps are part of our earlier work and the details of these steps along with examples are provided in [9].

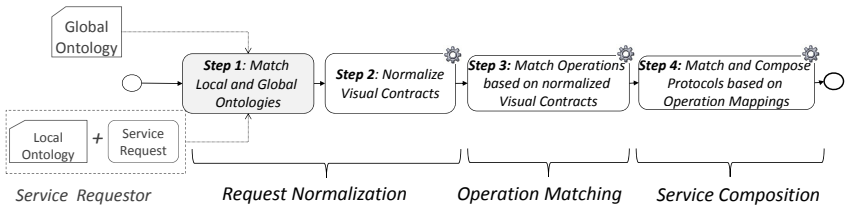

Fig. 3. Overview of the Automated Service Discovery and Composition

Based on a normalized service request, a multi-step service matching is carried out to identify and compose offered service(s). In the *operation matching* phase (Step 3), a subset of service offers is selected through matching of operations in service request and available service offers on the basis of their *normalized* VCs. Finally, in the *service composition* phase (Step 4), the protocols in service request and selected service offers are matched and composed to determine possible service compositions. In the next subsection, we will give an overview of *operation matching* phase, which is important for a better understanding of *service composition* phase.

3.2 Operation Matching

We have proposed a set of operation matching algorithms for RSDL-based service requests and offers, which are explained in detail in [9]. These algorithms compute $1:1$, $1:n$, $n:1$ and (partially) $n:m$ mappings between operations in available service offers and operations in service requests[4]. Operation matching determines whether the requested behavior is fulfilled by the offered behavior based on the visual contracts (VC). The result is a set of mappings between requested operations in a service request and offered operations of available service offers, which is defined as follows:

Definition 1 (Operation Mappings) *For a service request r, the set of operation mappings $OpMap_r$ is given by $OpMap_r = \{(m_r, m_o)\}$ where $m_r = op_i^r \rightarrow ... \rightarrow op_n^r$ is a sequence of requested operations in r that can be invoked on the requested service protocol in the given order. This sequence is mapped to $m_o = op_j^o \rightarrow ... \rightarrow op_m^o$, i.e., a sequence of one or more offered operations in a service offer o, which can be invoked on the offered service protocol in the given order.*

As an example, the operation mappings obtained by matching the service request of *HRS* and the service offer of *HotelX* (introduced in Figure 2) and two further service offers of *HotelY* and *PayOnline* are shown in Figure 4. One mapping contained in $OpMap_{hrs}$ is the $n:1$ operation mapping ($findRoom() \rightarrow viewDetails()$, $searchRoom()$) that maps the sequence of requested operations to an offered operation of *HotelX*. Based on the operation mappings, we compose the operations of the service offers to satisfy the service request in the next section.

4 Automated Service Composition

In Step 4 of our approach (see Fig. 3), the protocol of the service request and the protocols of the service offers that contain mapped operations are compared. Based on this comparison, the operations contained in the service offers are composed on the basis of the computed operation mappings to determine valid

[4] Please note that before operations in service offers and requests are matched, suitable service offers may be preselected based on a domain-specific categorization provided by the service market. However, this preselection is out of scope in our approach.

Providers / Requestor: HRS	HotelX	HotelY	PayOnline
findRoom() → viewDetails	searchRoom() **(n:1)**	getAvailableRoom() **(n:1)**	-
bookRoom()	validateCredentials() → makeRoomReservation() → notifyPerEmail() **(1:n)**	reserve() **(1:1)**	-
makePayment()	-	-	signIn() → payDues()→ generateReceipt()→ signOut() **(1:n)**

Fig. 4. Operation Mappings between the HRS Request and three available Service Offers

service compositions satisfying the request. In other words, we evaluate whether the matched operations of the offered service can be invoked in the desired order resulting in valid service compositions. To achieve this goal, *service composition* phase consists of multiple tasks:

(i) Translation of the protocols in service request and offer to a common semantic domain in terms of Labeled Transition System (LTS).
(ii) Composition of the LTS of requested service protocols and the LTSs of offered service protocols based on the operation mappings.
(iii) Analysis of the composed LTS to determine valid service compositions.

In the following sections, we discuss each of these tasks in detail.

4.1 Protocol Translation to LTS

RSDL service requests and offers contain linguistic heterogeneity by using two different notations to specify service protocols, i.e., UML sequence diagrams and UML statecharts, respectively. To enable automated consistency checks and determine valid service compositions, we translate these two notations to a common semantic domain. For that purpose, we first determine the behavior of these two notations in our approach by specifying their semantics formally.

There are already several semantic specification approaches [8, 4, 12] that aim at translating UML artifacts to a semantic domain, such as, labeled transition system (LTS), Petri net, CSP process, etc. In our approach, we use Dynamic Meta Modeling (DMM) [8] for specifying the behavior of the service protocol formally and unambiguously and to enable the compute of a LTS for a given service protocol. We argue that DMM is most suitable for our approach because it is particularly targeted at languages that have an abstract syntax defined in terms of a *metamodel*, such as the UML. Apart from that, DMM is intuitively understandable due to its visual, object-oriented syntax.

A formal definition of a LTS representing a service protocol is as follows:

Definition 2 (Requested/Offered LTS) *An LTS lts_{desc} representing the service protocol in a request/offer desc is a 5-tuple $(S, s_0, S_F, OP, \delta)$ where:*

(i) S *is the set of states, where* $s_0 \in S$ *represents the initial state and* $S_F \subseteq S$ *represents the set of final states.*

(ii) OP *is the set of requested/offered operations.*

(iii) $\delta \subseteq S \times OP \times S$ *is the transition relation and is* deterministic.

In this paper, we will use the notation $desc.S$, $desc.s_0$, $desc.S_F$, $desc.OP$, *and* $desc.\delta$ *to refer to the the elements of* lts_{desc}.

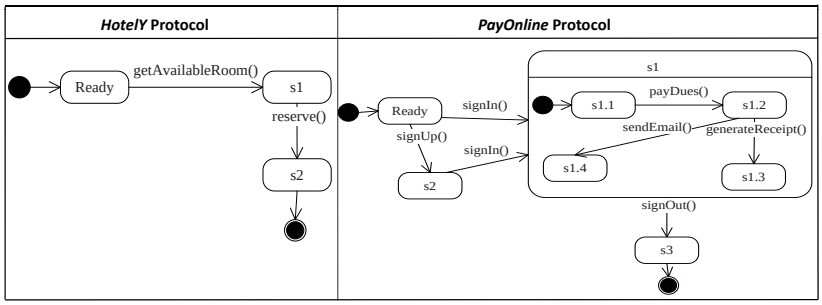

Fig. 5. Service Protocols for Payment and Hotel Service

The protocols of the service request and service offers of our running examples are shown in Figure 2 and Figure 5. The corresponding LTSs for these protocols computed using DMM are shown in Figure 6.

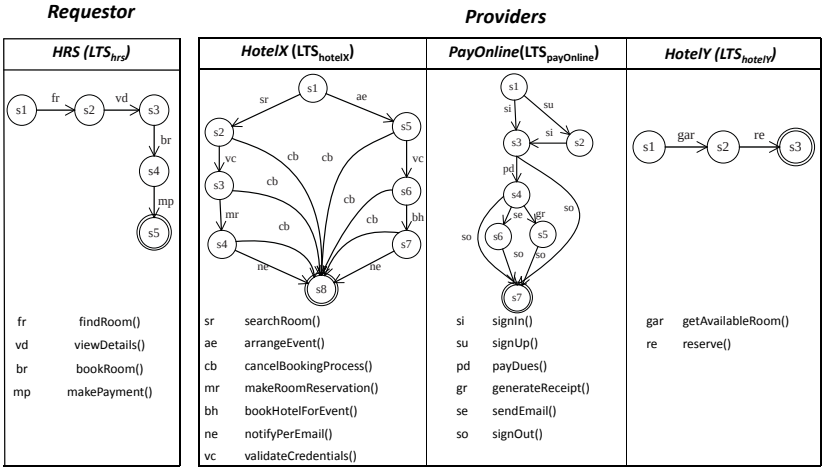

Fig. 6. LTSs for the Service Partners in our Running Example

In the next section, we identify possible service compositions by comparing the LTS of the service protocols.

4.2 LTS Composition

In this section, we present our LTS composition algorithm and explain it in the context of our running example. In order to automatically detect possible service compositions that satisfy a service request, we compose the LTSs of a service request and service offers by overlapping them on the basis of the operation mappings determined earlier.

Listing 1: Algorithm to compose and analyze LTSs of the service partners to find 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`($lts_r, \{lts_{o_1}, ..., lts_{o_k}\}$, $OpMap_r$)
  define s_{comp}:$(r.s_0, o_1.s_0, ..., o_k.s_o)$; // ①
  add s_{comp} as initial state to the composed LTS lts_{comp};
  while $lts_{comp}.hasMoreStates()$ **do** // ②
    s_{cur}=lts_{comp}.nextState(), where s_{cur}:$(r.s_c, o_1.s_c, ...o_k.s_c)$;
    while $s_{cur}.hasInvocableMappings()$ **do** // ③
      map=$s_{cur}.nextInvocableMapping()$, where $map : (m_r, m_o)$ AND $o \in \{o_1, ..., o_k\}$; // ④
      add s_{tar}:$(r.s_t, o_1.s_t, ...o_k.s_t)$ to lts_{comp}, where $r.s_c \xrightarrow{m_r} r.s_t$ AND $o.s_c \xrightarrow{m_o} o.s_t$; // ⑤
      add t_{comp} to lts_{comp}, where t_{comp}=$s_{cur} \xrightarrow{m_r \| m_o} s_{tar}$; // ⑥
    end
  end
  if $lts_{comp}.hasCompleteTraces()$ **then** // ⑦
    $Result_{comp}$= $lts_{comp}.completeTraces()$ **return** $Result_{comp}$
  end
  else return Failure_Notification
end

An algorithm for LTS composition is given in Listing 1. It takes as input the LTS of a service request and the LTSs of offered services. Additionally, it takes the prior computed operation mappings between requested and provided operations as input. As output, it returns a set of possible service compositions. If there is no possible service composition, a failure is notified and the requestor is provided with suggestions to restructure his/her request based on identified partial compositions.

Construction of the composed LTS: The algorithm in Listing 1 works as follows:

1. A composed state s_{comp}, which is a composition of the initial states of all the participating LTSs is created and added to the composed LTS lts_{comp}. It is represented with a vector of the participating states. Figure 7 shows a partially composed lts_{comp} for our running example with $cs1$ as its initial state.
2. Next, the `while`-loop traverses over the states of the composed LTS lts_{comp} and constructs it further until there are no more states to be traversed.
3. For every state s_{cur} that is currently traversed, the *invocable* operation mappings from $OpMap_r$ are determined. An operation mapping is *invocable* in a composed state s, if its comprising operation sequences can be directly invoked in s. For $cs1$ in Figure 7, there are two invocable mappings in $OpMap_{hrs}$. One Mapping is ($hrs.findRoom() \rightarrow hrs.viewDetails()$, $hotelX.searchRoom()$), which is invocable in $cs1$ because $hrs.findRoom() \rightarrow hrs.viewDetails()$ can be invoked from $hrs.s1$ and $hotelX.searchRoom()$ can be invoked from $hotelX.s1$. The other invocable mapping for $cs1$ is ($hrs.findRoom() \rightarrow hrs.viewDetails()$, $hotelY.getAvailableRoom()$).
4. For every *invocable* operation mapping *map* of s_{cur}, lts_{comp} is further constructed by composing the parts of the participating LTSs that overlap on the basis of *map*. For example, for one of the invocable mapping for $cs1$, the overlapping parts of LTS_{hrs} and LTS_{hotelX} are shown in the right-hand side of Figure 7.
5. For every *map*, a new composed state s_{tar} is created and added to lts_{comp}. s_{tar} is a composition of the states that *overlap* through the invocation of operation sequences in *map*. Figure 7 shows the newly created composed state $cs5$, where LTS_{hrs} and LTS_{hotelX} are in $hrs.s3$ and $hotelX.s2$, respectively after the invocation of the operation sequences in *map* under consideration.
6. Similarly, a composed transition t_{comp} is added between s_{cur} and s_{tar}, which represents the parallel invocation of the overlapping transitions. For example, the composed transition between $cs1$ and $cs5$ in Figure 7 represents the parallel invocation of $hrs.findRoom() \rightarrow hrs.viewDetails()$ and $hotelX.searchRoom()$in LTS_{hrs} and LTS_{hotelX}, respectively.
7. Analogously, the composed states $cs2$ and $cs5$ are traversed and as a result, lts_{comp} is further constructed. It is completely constructed if there are no more states to be traversed. After the complete construction, lts_{comp} is analyzed to determine any possible service compositions, which will be discussed in detail in next subsection.

A salient feature of the proposed algorithm is its *selective composition* strategy where the LTS composition is moderated through the LTS of the requested service protocol. That means, only those parts of the LTSs of offered service protocols are considered that overlap with the LTS of requested protocol and hence are *relevant* for the requestor based on the identified operation mappings. As a result of selective composition, the composed LTS is smaller in size and easier to analyze as compared to a conventionally composed LTS. For example, a conventional LTS composition mechanism may include some other transitions in

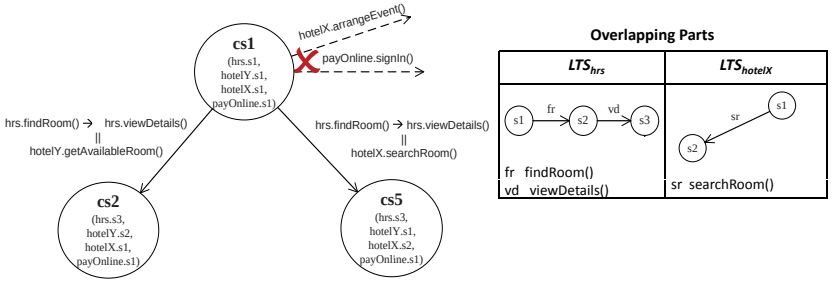

Fig. 7. Composed LTS after the first Iteration

the composed LTS, e.g., from $cs1$, some other possible transitions are $hotelX.arrangeEvent()$, $payOnline.signIn()$, etc. However, in our selective composition strategy, these transitions are not relevant for the requestor since they are not part of any invocable operation mappings from the composed state under consideration and hence are not included to lts_{comp}.

Analysis of the composed LTS: Our analysis mechanism determines all the *complete* traces, which represent the possible service compositions in the composed LTS. A trace of lts_{comp} is *complete* if it ends in a composed state where the LTS of requested service protocol is in its final state. In our given example, $cs1 \rightarrow cs2 \rightarrow cs3 \rightarrow cs4$ and $cs1 \rightarrow cs5 \rightarrow cs6 \rightarrow cs7$ represent two possible service compositions. Based on these results, a service requestor (e.g. HRS) may decide for a particular composition based on quality attributes of the provided services, such as, costs, performance, or reliability. Quality attributes are currently not considered in our approach but can be easily added by extending our rich service specification language.

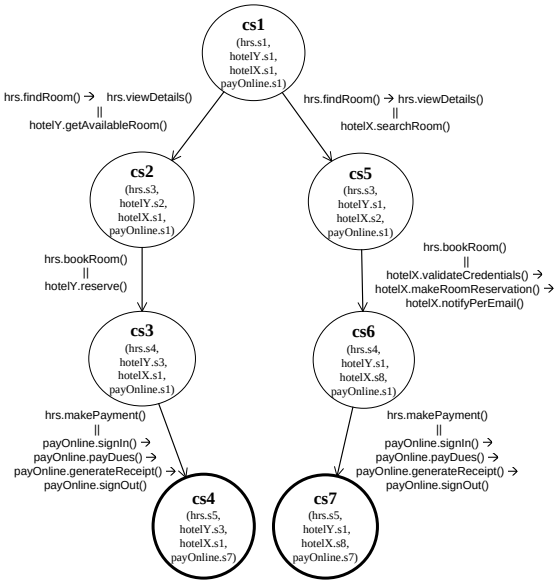

Fig. 8. Completely Composed LTS of our running Example

The algorithm notifies a failure finding a valid service composition, if lts_{comp} does not have any complete trace. In this case, the service requestor is provided

with feedback in terms of partially composed LTS and the particular points where a composition failed. On the basis of this feedback, the requestor may restructure his/her service request. For example, assuming that HRS request a further operation *paymentInfoPerEmail()* (referred as *ppe*) that is invoked after *makePayment()*. Figure 9(a) shows the LTS of HRS for this scenario. During operation matching, *paymentInfoPerEmail()* may be mapped to the operation *sendEmail()* of the PayOnline service (compare Fig. 6). Figure 9(b) shows the

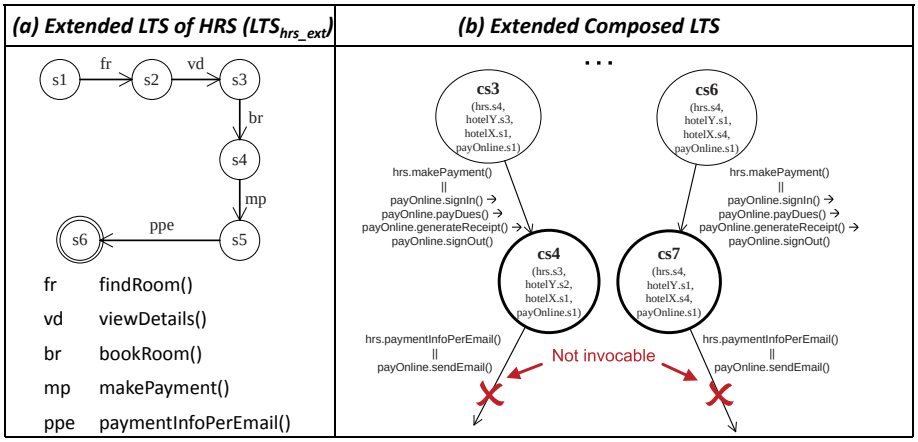

Fig. 9. Extended LTS of HRS and Composed LTS in the Failure Scenario

lower part of our previously computed composed LTS (cf. Fig. 8), which was complete after the traversal of *cs4* and *cs7*. However, in this slightly modified scenario, the composed LTS is no longer complete since also the operation *paymentInfoPerEmail()* needs to be fulfilled. Considering the operation mapping (*hrs.paymentInfoPerEmail()*, *payOnline.sendEmail()*) the operation *sendEmail()* of the PayOnline service would be suitable but cannot be invoked in any of the states *cs4* or *cs7*. As a consequence, none of the traces in the composed LTS is *complete* and hence there is no possible service composition satisfying the HRS request.

At the end of this phase, a set of possible service compositions is achieved that satisfy the service request or a failure notification with feedback for a requestor. These service composition results can be used by the service requestor to define a concrete service composition. To summarize, our service composition mechanism further strengthens the vision of On-The-Fly SOAs through elaborate and automatic matching and composition of service descriptions while resolving their heterogeneity issues.

5 Tool Support

We have implemented a tool, called RSDL Workbench as a part of the service computing platform developed at the Collaborative Research Center (CRC 901)

"'On-The-Fly Computing"'[5]. It provides an editor for service partners to specify their RSDL-based service descriptions. The operation semantics normalization and operation matching is fully automated based on given local/global ontology mappings.

The RSDL Workbench has been realized as an Eclipse plugin and is implemented using EMF, GMF, and Henshin[6]. Figure 10 shows two screenshots of the service specification editor contained in the RSDL Workbench. Currently, we extend the workbench to also support the automated service composition. In this context, we integrate the existing DMM tool support for the translation of service protocols into LTSs. Furthermore, an extensive evaluation of the effectiveness of the proposed approach through a variety of case studies in the CRC environment is also in progress.

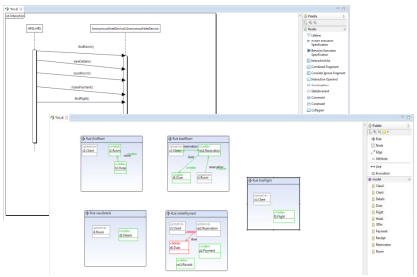

Fig. 10. Screenshots for RSDL Workbench

6 Related Work

For this section, our particular focus is on the *workflow-based* approaches for automatic service composition where a service composition is defined as a workflow with multiple services interacting on the basis of control/data flow [20].

Automation of service composition process is an important concern of the SOC research community [3] and is further complicated by the advent of *comprehensive* service description languages [16, 6]. Below, we discuss some of the salient approaches [2, 11, 15, 1, 5, 21, 18, 7] on the basis of multiple factors, such as, the matched elements of request/offers, heterogeneity resolution, etc. and compare them with our approach.

Concerning matched elements in service request and offer, some [2, 11, 15] examine operation signatures and operation semantics with the assumption that the requested/offered service consist of a single operation and hence, do not consider service protocols. [2] also matches quality attributes of request and offer but the calculation of a service's QoS value is not clearly defined. Similar to our approach, [15] specifies the behavior of the requested/offered services in terms of visual contracts. However, our approach additionally considers service protocols and comes with a detailed heterogeneity resolution mechanism. Other approaches [1, 5, 21, 17, 18] consider service protocols while matching as well. For instance, METEOR-S [1, 19] enables service discovery by matching of WSDL-S-based [14] service descriptions considering operation semantics and QoS values of services. Additionally, service composition is enabled by considering requested service protocol defined in a BPEL-like notation. however, offered services are assumed to

[5] http://sfb901.uni-paderborn.de
[6] http://www.eclipse.org/modeling/emft/henshin/

have single operation, hence offered service protocols are not considered. [5, 21, 17, 18] propose service composition mechanisms for service descriptions specified in de facto languages, such as, OWL-S[16] and UML. These mechanisms discover and compose services offers based on the operation signatures and service protocols but do not consider operation semantics. In this context, [5] considers offered service protocols only. [21, 18] additionally analyze the quality attributes for matching service descriptions. Our protocol matching mechanism can be compared to [18, 21], where [18] comes up with an LTS-composition mechanism based on operation signature matching. [21] considers the protocols specified using different languages and translates them to a common semantic domain before checking their conformance.

In this context, [22] comes up with a comprehensive service composition mechanism for OWL-S based service descriptions, where matching is based on operation signatures, operation semantics as well as service protocols. However, it has certain shortcomings in terms of heterogeneity resolution features, which we will discuss shortly.

Concerning the resolution of the multifaceted heterogeneity of service partners, existing approaches follow different strategies. Some approaches [2, 1, 7] realizes the need for ontological heterogeneity resolution and come up with mechanism for this purpose. For instance, [1] provides semantic annotation for WSDL elements, which can support an ontological heterogeneity resolution mechanism, whereas [7] comes up with an elaborate mediator-based mechanism for this purpose. [21] provides a solution for linguistic heterogeneity by using abstract state machines as the common semantic domain. Other service composition approaches either tend to ignore the fact that the service request and offers can be heterogeneous in a realistic scenario [15, 22] or they [11, 5, 18] assume the existence of such a mechanism that can overcome these difference. These approaches define their composition mechanisms for normalized request/offers and therefore avoid major complexities of the problem at hand.

From this discussion, it becomes evident that there is a need for approaches that: 1) match and compose the service description by comprehensively matching the structural and behavioral aspects of service descriptions 2) provide multifaceted heterogeneity resolution mechanism for the service partners in realistic scenarios. We claim that our approach fulfills these requirements and hence is a promising approach for On-The-Fly SOAs.

7 Conclusion and Future Work

To enable the vision of *On-The-Fly* SOAs, we propose an automated composition mechanism based on our earlier work on rich service descriptions and automatic service discovery [9, 10]. Our proposed mechanism ensures accurate service discovery results as it relies on comprehensive matching of the service request and offers based on their structural as well as behavioral features. Additionally, it comes up with a service protocol composition mechanism that determines possible service compositions. We have implemented the RSDL Workbench as a

part of the service computing platform being developed at the Collaborative Research Center 901 "On-The-Fly Computing" and evaluated our approach on a real-world case study of our industrial partner HRS.

In future, we aim to evaluate our approach more extensively through further case studies. We also intend to consider service protocols with additional complexity, such as, loops, multiple regions, etc. We also aim to further strengthen our heterogeneity resolution mechanism with features, such as, complex mappings between ontologies, etc.

References

1. Aggarwal, R., Verma, K., Miller, J.A., Milnor, W.: Constraint Driven Web Service Composition in METEOR-S. In: IEEE International Conference on Services Computing (SCC'04). pp. 23–30. IEEE Computer Society (2004)
2. Bartalos, P., Bieliková, M.: QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions. In: Proceedings of IEEE International Conference on Web Services (ICWS'10). pp. 345–352. IEEE Computer Society, Los Alamitos, CA, USA (2010)
3. Bartalos, P., Bieliková, M.: Automatic Dynamic Web Service Composition: A Survey and Problem Formalization. Computing and Informatics 30(4), 793–827 (2011)
4. Bernardi, S., Donatelli, S., Merseguer, J.: From UML Sequence Diagrams and Statecharts to analysable Petri Net Models. In: Proceedings of the 3rd International Workshop on Software and Performance. pp. 35–45. ACM, NY, USA (2002)
5. Brogi, A., Corfini, S., Popescu, R.: Semantics-based Composition-oriented Discovery of Web Services. ACM Trans. Internet Technol. 8(4), 19:1–19:39 (2008)
6. ESSI WSMO Working Group: Web Service Modelling Ontology. http://www.wsmo.org/
7. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: IEEE International Conference on Web Services (ICWS'05). pp. 321–328. IEEE Computer Society (2005)
8. Hausmann, J.H.: Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages. Ph.D. thesis, University of Paderborn (2005)
9. Huma, Z., Gerth, C., Engels, G., Juwig, O.: 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). LNCS, vol. 7590, pp. 709–725. Springer-Verlag, Berlin, Heidelberg (2012)
10. Huma, Z., Gerth, C., Engels, G., Juwig, O.: 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. CEUR Workshop Proceedings, vol. 855, pp. 90–97. CEUR-WS.org (2012)
11. Kona, S., Bansal, A., Blake, M.B., Gupta, G.: Generalized Semantics-Based Service Composition. In: IEEE International Conference on Web Services (ICWS'08). pp. 219–227. IEEE Computer Society, Washington, DC, USA (2008)
12. Küster, J.M., Stehr, J.: 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 (2003)
13. Lohmann, M.: Kontraktbasierte Modellierung, Implementierung und Suche von Komponenten in serviceorientierten Architekturen. Ph.D. thesis, University of Paderborn (2006)

14. LSDIS Lab: Web Service Semantics. http://lsdis.cs.uga.edu/projects/WSDL-S/wsdl-s.pdf
15. Naeem, M., Heckel, R., Orejas, F., Hermann, F.: Incremental Service Composition based on Partial Matching of Visual Contracts. In: Proceedings of Fundamental Approaches to Softw. Eng. (FASE'10). LNCS, vol. 6013, pp. 123–138. Springer (2010)
16. OWL-S Coalition: OWL-based Web Service Ontology. http://www.ai.sri.com/daml/services/owl-s/1.2/ (2006)
17. Pathak, J., Basu, S., Honavar, V.: Modeling Web Service Composition using Symbolic Transition Systems. In: Proceedings of AAAI Workshop on AI-Driven Technologies for Service-Oriented Computing. AAAI Press, California, USA (2006)
18. Pathak, J., Basu, S., Honavar, V.: 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). LNCS, vol. 4294, pp. 314–326. Springer-Verlag, Berlin, Heidelberg (2006)
19. Patil, A.A., Oundhakar, S.A., Sheth, A.P., Verma, K.: Meteor-s Web Service Annotation Framework. In: Proceedings of the 13th International Conference on World Wide Web (WWW'04). pp. 553–562. ACM, New York, NY, USA (2004)
20. Rao, J., Su, X.: 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). vol. 3387, pp. 43–54. Springer-Verlag, Berlin, Heidelberg (2004)
21. Spanoudaki, G., Zisman, A.: Discovering Services during Service-Based System Design Using UML. IEEE Trans. on Softw. Eng. 36(3), 371–389 (2010)
22. Vaculin, R., Neruda, R., Sycara, K.: The process mediation framework for semantic web services. Int. J. Agent-Oriented Softw. Eng. 3(1), 27–58 (Feb 2009)
23. W3C: Web Service Description Language(WSDL). http://www.w3.org/TR/wsdl20/ (2007)