Automated Service Composition for on-the-Fly SOAs

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

[1] Department of Computer Science, University of Paderborn, Germany*
{zille.huma,gerth,engels}@upb.de
[2] HRS-Hotel Reservation Service, Germany
Oliver.Juwig@hrs.de

Abstract. In the service-oriented computing domain, the number of available software services steadily increased in recent years, 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 service requests and available offers based on their structural as well as behavioral aspects. Afterwards, possible service compositions are determined by composing 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 through service-oriented architecture (SOA). For this purpose, a SOA developer defines a *service request* to discover and compose the services that are developed and published on service markets by service providers in terms of *service offers*.

With the advent of cloud computing, the growing plethora of available services provides enormous opportunities for the development of future *On-The-Fly* SOAs that are highly flexible and can be aligned more easily to meet constantly changing requirements. 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 rich service specifications that comprise structural as well as behavioral aspects of requested and offered services.

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

S. Basu et al. (Eds.): ICSOC 2013, LNCS 8274, pp. 524–532, 2013.
© Springer-Verlag Berlin Heidelberg 2013

Secondly, service discovery and composition mechanisms must deal with the existing multifaceted heterogeneity of the involved service partners, such as their independent domain ontologies, selection of different languages/notations to specify service descriptions, and different granularity levels of the service descriptions as a result of their independent domain knowledge, etc. To overcome these challenges, we proposed a UML-based rich service description language (RSDL) [6] and an automatic service discovery mechanism for RSDL-based service descriptions [5]. 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. 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.

In the next section, we briefly introduce our proposed language for comprehensive service specifications and our service discovery mechanism. In Section 3, we describe our service composition mechanism in detail. Section 4 briefly introduces our tool support. In Section 5, we discuss related work and finally, we conclude the paper and give an outlook on future work in Section 6.

2 Foundations

To realize our vision of a comprehensive service specification, we proposed a UML-based rich service description language (RSDL) [6]. Our RSDL provides notations to describe the structure and the behavior of service requests and offers. Figure 1(a) shows a RSDL-based *service request* of HRS consisting of three parts. *(A)* specifies operation signatures, i.e., *findRoom()*, *viewDetails()*, *bookRoom()*, ..., using the Web Service Description Language (WSDL) [17]. *(B)* specifies operation semantics in terms of pre- and post-conditions for individual operations specified using UML-based visual contracts (VC) [9]. 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. Finally, a desired invocation sequences is specified in *(C)* in terms of a requestor protocol as UML sequence diagram. Similarly, Figure 1(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 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. A more detailed description of our RSDL is given in [7].

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

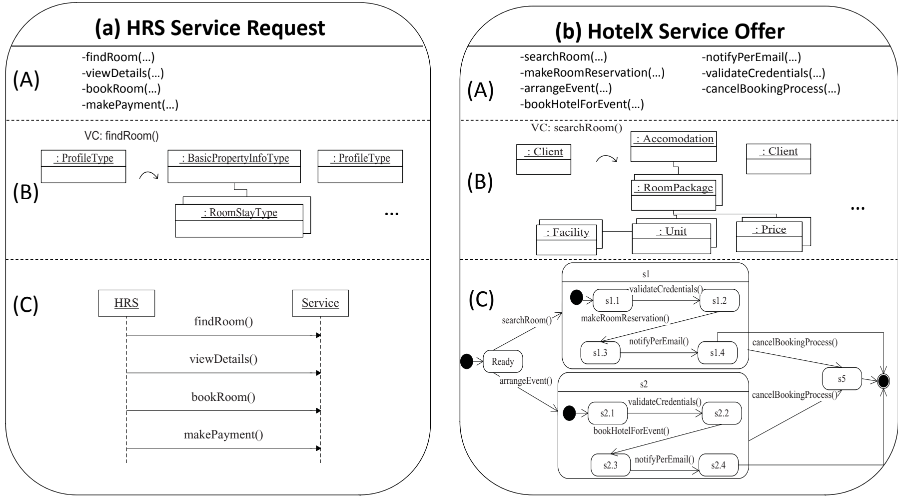

Fig. 1. (a) RSDL-based Service Request of HRS and (b) RSDL-based Service Offer of HotelX

To enable an automated service discovery and composition for such RSDL-based service requests and offers, we proposed a multi-step approach, whose overview is given in [7]. 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 [5,7]. In this paper, our main focus is on service composition, which we will discuss in detail in the next section.

After operation matching phase (Step 3), the result is a set of $1:1$, $1:n$, $n:1$ and (partially) $n:m$ mappings between requested operations in a service request and offered operations in available service offers. A mapping in this set is represented as (m_r, m_o), where m_r is an operation or a sequence of operations in the service request r, which is mapped to m_o, which is an operation or a sequence of operations in a service offer o.

As an example, the operation mappings obtained by matching the service request of *HRS* and the service offer of *HotelX* and two further service offers of *HotelY* and *PayOnline* are shown in Figure 2. One mapping for 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.

3 Automated Service Composition

To compose service offers, our approach compares and composes the protocol of a service request and the protocols of service offers that contain matched operations. For that purpose, we evaluate whether the mapped operations of the offered service can be invoked in the desired order resulting in valid service

Requestor HRS \ Providers	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. 2. Operation Mappings between the HRS Request and three available Service Offers

compositions. The *service composition* consists of three main tasks: Translation of the protocols to labeled transition systems (LTS), composition of LTSs, and an analysis of the composed LTS. Due to space constraints, the details of the first task are described in [7]. Its outcome are LTSs for the service protocols that are shown in Figure 3. In the following sections, we discuss the last two tasks in detail.

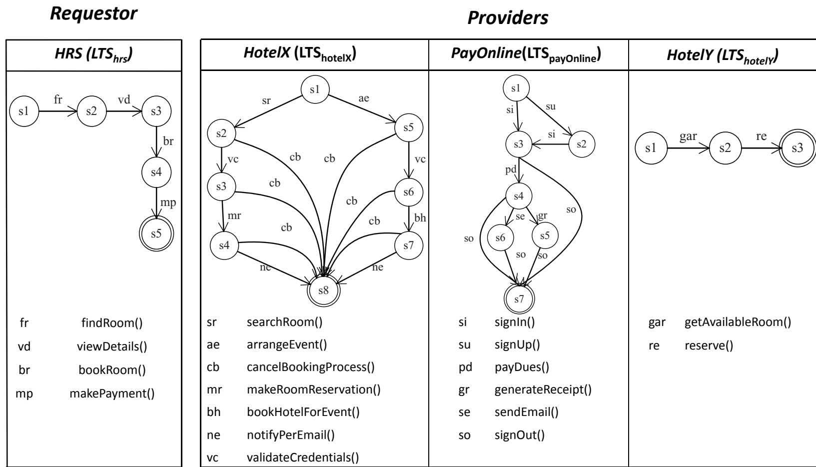

Fig. 3. LTSs for the Service Partners in our Running Example

LTS Composition: In order to automatically detect possible service compositions, we compose the LTSs of a service request and service offers by overlapping them on the basis of the operation mappings determined earlier. As output our algorithm returns a set of possible service compositions or a failure notification in case of no possible service composition. In this case, the requestor is provided with suggestions to restructure his/her request based on identified partial compositions. In the following, we describe our algorithm given in Listing 1.

1. A composed state s_{comp}, i.e, a composition of the initial states of all participating LTSs, is created and added to the yet empty composed LTS lts_{comp}.

Figure 4 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 currently traversed state s_{cur}, 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 instance, for $cs1$ in Figure 4, one of the two invocable mappings in $OpMap_{hrs}$ is $(hrs.findRoom() \rightarrow hrs.viewDetails()$, $hotelX.searchRoom())$ as $hrs.findRoom() \rightarrow hrs.viewDetails()$ can be invoked from $hrs.s1$ and $hotelX.searchRoom()$ can be invoked from $hotelX.s1$.

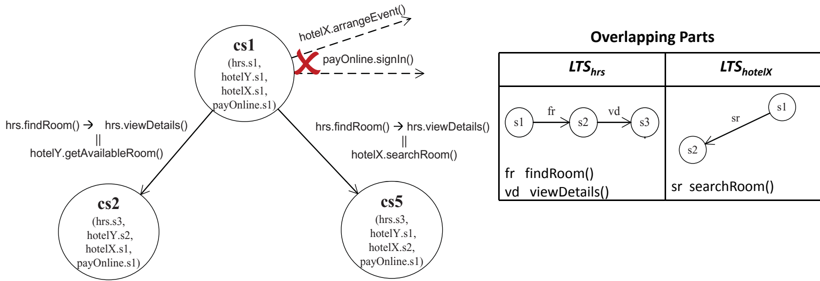

Fig. 4. Composed LTS after the first Iteration

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, the overlapping parts of LTS_{hrs} and LTS_{hotelX} for the invocable mapping mentioned earlier are shown in the right-hand side of Figure 4.
5. 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 4 represents the parallel invocation of $hrs.findRoom() \rightarrow hrs.viewDetails()$ and $hotelX.searchRoom()$in LTS_{hrs} and LTS_{hotelX}, respectively.
6. Analogously, the composed states $cs2$ and $cs5$ are traversed and as a result, lts_{comp} is further constructed until there are no more states to be traversed. Figure 5 shows the completely composed LTS lts_{comp} of our running example, which is analyzed to determine any possible service compositions in the 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, the composed LTS is smaller in size and easier to analyze as compared

Listing 1. Algorithm to compose LTSs of the service partners

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_o1, ..., lts_ok}, 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 ∈ {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 --m_r--> r.s_t AND
            o.s_c --m_o--> o.s_t;                                          // ⑤
            add t_comp to lts_comp, where t_comp=s_cur --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
```

to a conventionally composed LTS. For example, a conventional LTS composition mechanism may include some other transitions in the composed LTS, e.g., from $cs1$, some other possible transitions are $hotelX.arrangeEvent()$ or $payOnline.$-$signIn()$.

Analysis of the Composed LTS: In our given example, $cs1 \rightarrow cs2 \rightarrow cs3 \rightarrow cs4$ and $cs1 \rightarrow cs5 \rightarrow cs6 \rightarrow cs7$ represent two possible service compositions (see Figure 5). Based on these results, a service requestor (e.g. HRS) may decide for a particular composition based on quality attributes of the provided services, which can be easily added by extending our rich service specification language.

The algorithm notifies a failure in finding a valid service composition, if lts_{comp} does not have any complete trace. In this case, the service requestor gets feedback in terms of the 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. A detailed example for such a failure scenario is specified in [7]. Finally, a set of possible service compositions is obtained that satisfy the service request or a failure notification with feedback for a requestor is returned.

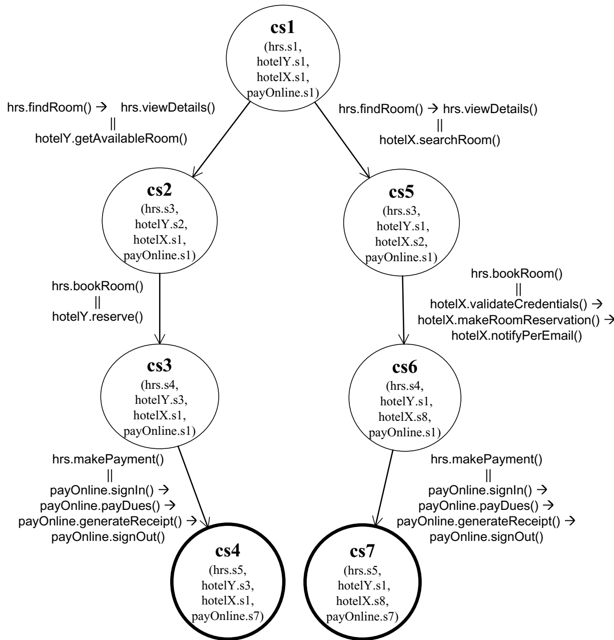

Fig. 5. Completely Composed LTS of our running Example

4 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"[2]. It provides an editor and matcher for service partners to specify and automatically match their RSDL-based service descriptions.

The RSDL Workbench has been realized as an Eclipse plugin and is implemented using EMF, GMF, and Henshin[3]. For more detailed information, the interested reader is referred to [7].

5 Related Work

For this section, our particular focus is on the *workflow-based* approaches for automatic service composition[14].

Concerning service request and offer matching, none of these approaches [2,8,11,1,3,15,13] are comprehensive enough. Some [2,8,11] examine operation signatures and operation semantics assuming that the requested/offered service consist of a single operation and hence, do not consider service protocols. Similar to our approach, [11] specifies the operation semantics in terms of visual contracts. On the contrary, METEOR-S [1], which allows service discovery

[2] `http://sfb901.uni-paderborn.de`

[3] `http://www.eclipse.org/modeling/emft/henshin/`

and composition of WSDL-S-based [10] service descriptions, only considers a requested service protocol and offered services are assumed to provide only a single operation. Similarly, [3,15,13] propose service composition mechanisms for OWL-S [12] and UML-based service descriptions based on the operation signatures and service protocols but do not consider operation semantics. In this context, [16] comprehensively matches OWL-S-based service specifications but has certain shortcomings in terms of heterogeneity resolution features, which we will discuss shortly.

Concerning the resolution of the multifaceted heterogeneity of service partners, some approaches [2,1,4] realizes the need for ontological heterogeneity resolution and come up with mechanism for this purpose. For instance, in [1] semantic annotation for WSDL elements are described, which can support an ontological heterogeneity resolution mechanism, whereas an elaborate mediator-based mechanism is used in [4]. The resolutions of linguistic heterogeneity is considered in [15]. Other service composition approaches either do not address the underlying heterogeneity [11,16] or they [8,3,13] assume the existence of a resolution mechanism and therefore avoid major complexities of the problem at hand.

We claim that our approach overcomes the weaknesses of most of the approaches discussed here and hence is a promising approach for On-The-Fly SOAs.

6 Conclusion and Future Work

To enable the vision of *On-The-Fly* SOAs, we proposed an automated composition mechanism based on our earlier work on rich service descriptions and automatic service discovery [5,6]. Our proposed mechanism ensures accurate results as it relies on comprehensive matching and composition of the service request and offers based on their structural as well as behavioral features. We have implemented the RSDL Workbench and evaluated our approach on a real-world case study of our industrial partner HRS.

In future, we aim to evaluate the effectiveness of our approach more extensively through further case studies in the CRC environment. We also aim to further strengthen our heterogeneity resolution mechanism with features, such as, complex mappings between ontologies.

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 2004), 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 Int. Conf. on Web Services (ICWS 2010), pp. 345–352. IEEE Comp. Soc. (2010)
3. 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)

4. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX - A Semantic Service-Oriented Architecture. In: IEEE International Conference on Web Services (ICWS 2005), pp. 321–328. IEEE Computer Society (2005)
5. Huma, Z., Gerth, C., Engels, G., Juwig, O.: Towards an Automatic Service Discovery for UML-based Rich Service Descriptions. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MODELS 2012. LNCS, vol. 7590, pp. 709–725. Springer, Heidelberg (2012)
6. 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 Conference on Advanced Information Systems Engineering (CAiSE 2012). CEUR Workshop Proceedings, vol. 855, pp. 90–97. CEUR-WS.org (2012)
7. Huma, Z., Gerth, C., Engels, G., Juwig, O.: Automated Service Discovery and Composition for On-the-Fly SOAs. Tech. Rep. TR-RI-13-333, University of Paderborn, Germany (2013), http://is.uni-paderborn.de/uploads/tx_sibibtex/tr-ri-13-333.pdf
8. Kona, S., Bansal, A., Blake, M.B., Gupta, G.: Generalized Semantics-Based Service Composition. In: IEEE International Conference on Web Services (ICWS 2008), pp. 219–227. IEEE Computer Society, Washington, DC (2008)
9. Lohmann, M.: Kontraktbasierte Modellierung, Implementierung und Suche von Komponenten in serviceorientierten Architekturen. Ph.D. thesis, University of Paderborn (2006)
10. LSDIS Lab: Web Service Semantics, http://lsdis.cs.uga.edu/projects/WSDL-S/wsdl-s.pdf
11. Naeem, M., Heckel, R., Orejas, F., Hermann, F.: Incremental Service Composition based on Partial Matching of Visual Contracts. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 123–138. Springer, Heidelberg (2010)
12. OWL-S Coalition: OWL-based Web Service Ontology (2006), http://www.ai.sri.com/daml/services/owl-s/1.2/
13. 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 (2006)
14. Rao, J., Su, X.: A Survey of Automated Web Service Composition Methods. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005)
15. Spanoudaki, G., Zisman, A.: Discovering Services during Service-Based System Design Using UML. IEEE Trans. on Softw. Eng. 36(3), 371–389 (2010)
16. Vaculin, R., Neruda, R., Sycara, K.: The process mediation framework for semantic web services. Int. J. Agent-Oriented Softw. Eng. 3(1), 27–58 (2009)
17. W3C: Web Service Description Language (WSDL) (2007), http://www.w3.org/TR/wsdl20/