Automated Service Composition with Adaptive Planning[⋆]

Sandrine Beauche[1] and Pascal Poizat[1,2]

[1] INRIA/ARLES project-team, France
{sandrine.beauche,pascal.poizat}@inria.fr
[2] IBISC FRE 3910 CNRS – Université d'Évry Val d'Essonne, France

Abstract. Service-Oriented Computing is a cornerstone for the realization of user needs through the automatic composition of services from service descriptions and user tasks, i.e., high-level descriptions of the user needs. Yet, automatic service composition processes commonly assume that service descriptions and user tasks share the same abstraction level, and that services have been pre-designed to integrate. To release these strong assumptions and to augment the possibilities of composition, we add adaptation features into the service composition process using semantic descriptions and adaptive extensions to graph planning.

Keywords: Services, Task-Oriented Computing, Composition, Software Adaptation, Planning, Workflow Languages, Tools.

1 Introduction

Task-Oriented Computing (TOC) envisions a user-friendly world where *user tasks* would be achieved by the automatic assembly of resources available in the environment. Service-Oriented Computing (SOC) is a cornerstone towards the realization of this vision, through the abstraction of heterogeneous resources as services. Yet, services being elements of composition developed by different third-parties, their reuse and assembly naturally raises composition mismatch issues [1]. Moreover, the TOC vision yields a higher description level for the composition requirements, *i.e.*, the user task(s), as the user only has an abstract vision of her/his needs which are usually not described at the service level.

To illustrate these issues, we use a running example [2], inspired by [3], which exposes a set of available services described with a conversation, a capability, inputs and outputs (Figs. 1 and 2). Conversations describe *how* to use services, while capabilities are semantic annotations that enable automatic reasoning for discovery and composition. In our work, conversations are described with a generic workflow language, YAWL, for which transformations from/to BPEL have been defined [4]. The Amazon service can be used to look for an eBook and provides a capability called BookSearch with a conversation (sequence) over

[⋆] This work is supported by the project "PERvasive Service cOmposition" (PERSO) of the French National Agency for Research, ANR-07-JCJC-0155-01.

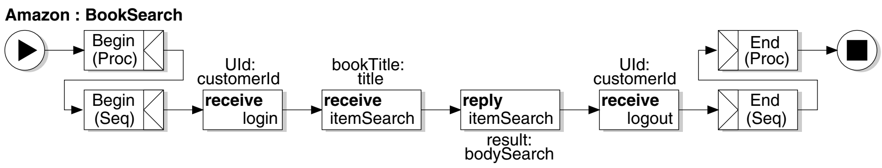

Fig. 1. Amazon service conversation (YAWL)

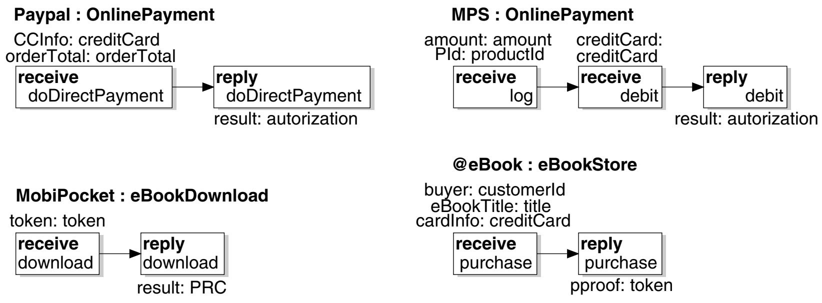

Fig. 2. Service conversations (communication part, YAWL)

three operations: login and logout, with a customer identifier (customerId) as input (in message part UId), and itemSearch with a book title (title) as input (in message part bookTitle) and a structured information on the search result (bodySearch) as output (in message part result). Additionally, Paypal and MPS can be used for payment, while @eBook can be used to search and pay at once. Finally, MobiPocket can be used to download an eBook in PRC format.

Still, the user knows neither the service capabilities, nor the data that should be exchanged between them to achieve service composition. The user only has a high-level view of her/his needs (user task): a capability, the inputs (s)he is ready to provide and the outputs (s)he expects. In the example, (s)he requires an eBookRetrieve capability, to provide title, customerId, and creditCard information, and finally get an eBook in PRC format. There is clearly a (vertical) mismatch between the user's needs and the service descriptions.

Additionally, the services have been developed by different third-parties. One may expect to compose them, while from the input/output perspective they could not be chained as-is. For example, Amazon should be composed with Paypal or MPS but part of the input data they require (respectively orderTotal and amount+productId) does not correspond to what one gets from a call to Amazon (bodySearch). This illustrates a (horizontal) mismatch.

These two dimensions of interoperability, namely *horizontal* (communication protocol and data flow between services) and *vertical matching* (correspondence between an abstract user task and concrete service capabilities) should be supported in the composition process.

The rest of this paper is organized as follow. Section 2 motivates the use of planning and adaptation, and discusses related work. Then, in Section 3, we present the principles of our approach for which more details can be found in [2], and we end with conclusions.

2 Discussion and Related Work

On the one hand, *planning*, is increasingly applied in SOC due to its support for automatic service composition from underspecified requirements [5]. Chaining-based planning composes services from provided and expected data, while hierarchical planning supports the decomposition of abstract requirements into concrete sets of tasks. Still, planning is not able to solve horizontal mismatch. On the other hand, *software adaptation* [1], is used to augment the possibility for component reusability and assembly, thanks to the automatic generation of software pieces, called adaptors, solving mismatch out in a non intrusive way. In this article we propose to combine planning and adaptation techniques.

Automatic composition is an important issue in SOC and numerous works have addressed this over the last years [6–13]. Various criteria could be used to differentiate these works, yet, due to our motivations, we will focus on issues related to user task requirements, vertical, and horizontal adaptation.

While both data input/output and capability requirements should be supported to ensure composition is correct wrt. the user needs, only [12, 13] do, while [7–11] support data only and [6] supports capabilities only. As far as adaptation is concerned, [9–12] support a form of horizontal (data) adaptation, using semantics associated to data; and [7] a form of vertical (capability abstraction) adaptation, due to its hierarchical planning inheritance. In our proposal, we combine the two techniques to achieve both adaptation kinds.

Few works explicitly add adaptation features to SOC [4, 14]. They adopt a different and complementary view wrt. ours since their objective is not to integrate adaptation within the service composition process in order to increase the composition possibilities, but rather to tackle protocol adaptation between clients and services, *e.g.*, to react to service replacement. Indeed, the most advanced software adaptation works [15, 16, 1] solve protocol mismatch between a fixed set of components, but tackle neither the discovery of the required components nor the composition towards user needs.

More information on planning and related work can be found in [2].

3 Adaptive Planning Composition

The basis of our work is the extension of the GraphHTN hierarchical planning technique [17] with horizontal adaptation features, and its application for service composition. Comprehensive information about the extension is given in [2].

We rely on two structures to support adaptation. Horizontal adaptation is supported by relations in an ontology of data types, in a structure we call *Data Semantic Structure* (DSS). It associates a set of concepts with a composition

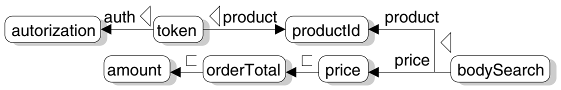

Fig. 3. DSS example

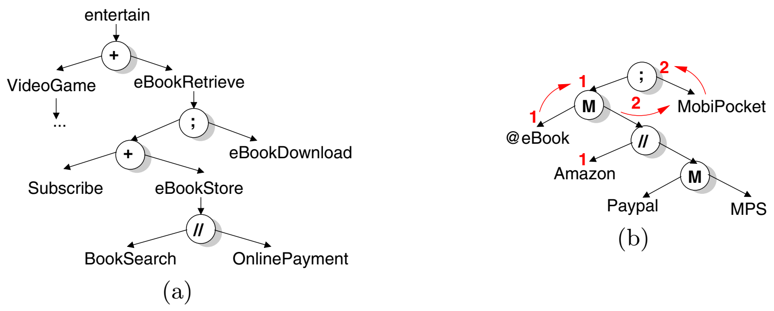

Fig. 4. CSS and l-CSS example (marking is used in graphplan building wrt. Fig. 5)

relation (◁) – supporting (de)composition of data – and a simulation relation (⊏) – supporting data replacement. Using the DSS of our example (Fig. 3), we see that a token could be decomposed into an authorization and a productId (or the other way round) and that a price could replace an orderTotal as input for a service. Vertical adaptation is supported by a hierarchical (tree) structure describing relations between capabilities, that we call *Capability Semantic Structure* (CSS). It expresses (i) decomposition relations between abstract capabilities and more concrete ones, and (ii) ordering constraints between capabilities. The CSS nodes are either capabilities or control structures: sequence (;), choice (+) and parallel (//). In our example (Fig 4(a)), eBookStore is a capability which can be performed directly by a service, or that can be decomposed as the parallel execution of BookSearch and OnlinePayment capabilities.

Given a user task, a set of services, and both a DSS and a CSS, we proceed as follows. The CSS is first used to select, on the basis of their capabilities, services that could be used in the composition. Accordingly, the CSS is labelled with these services (l-CSS). A graph planning structure, named graphplan, is then computed. It chains services capabilities based on input/output dependencies and l-CSS constraints. Finally the graphplan is analyzed to retrieve all service compositions corresponding to the user task (which can be none).

Service Discovery and l-CSS Computation. The CSS is first restricted to the subtree with the user task capability as root. An abstract capability node is replaced by a *method node* (M) which denotes a choice: it can be either instantiated directly by some service *or* its definition (*i.e.*, its subtree) can. In Figure 4(a), eBookStore may be either obtained by calling @eBook or by composing in parallel Amazon (capability BookSearch) and Paypal or MPS (capability OnlinePayment). A capability node is replaced by the service that supports it (or

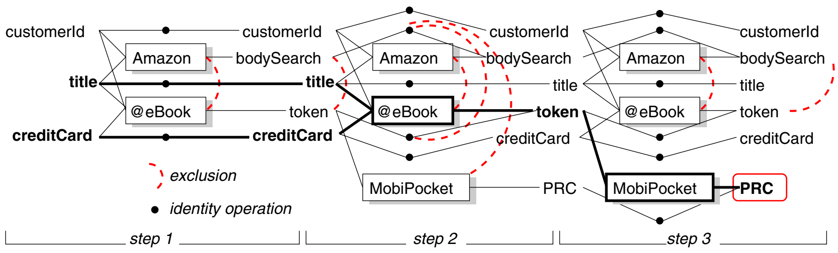

Fig. 5. Adaptive graphplan building (no data adaptation)

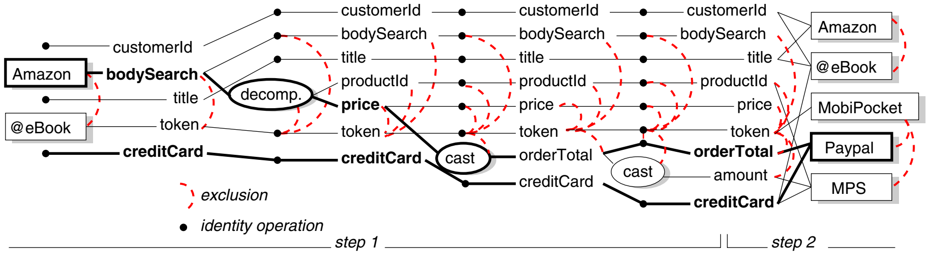

Fig. 6. Adaptive graphplan building (with data adaptation, principle)

siblings under a M node if several services apply, as for OnlinePayment). Finally, branches without service instances are discarded and control nodes with only one child are simplified. The l-CSS for our example is presented in Figure 4(b).

Graphplan Building with Vertical Adaptation. The graphplan is a structure with alternating fact (data) and action (service calls) layers. Dependencies between data and services are represented with arcs. The initial data layer corresponds to the user-provided inputs. The graphplan is then built (Fig. 5) chaining services (i) if their input data is available and (ii) following the orderings imposed by the l-CSS. Once a service is selected, it is tagged in the l-CSS (Fig. 4(b)) and its outputs are added to the next data layer. Identity operations are used to keep data from one data layer to the next one. As an example, the chaining of @eBook at step 1 enables the chaining of MobiPocket at step 2 (see Figs. 4(b) and 5). This would yield a correct composition, still, that should not contain Amazon that has been chained at step 1. To deal with such cases, exclusion relations are used to prevent services with exclusive capabilities in the l-CSS to appear in the same solution. Exclusions are propagated all along the graphplan. Since our objective is to generate all possible compositions, we stop the building when the maximum solution length, calculated with the l-CSS (here, 3), is reached.

Adding Horizontal Adaptation to the Picture. Let us now suppose we are after step 1 of Figure 5 and continue in Figure 6. According to the l-CSS

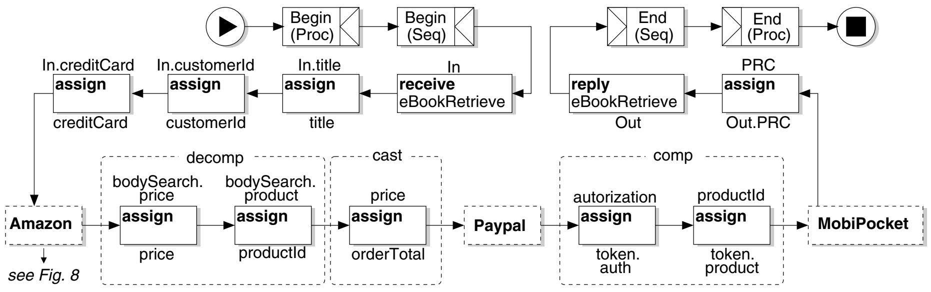

Fig. 7. A composition for user task (eBookRetrieve, {title, customerId, creditCard}, {PRC}) (YAWL)

(Fig. 4(b)), Paypal should be applicable. Yet, it is not, as it requires the unavailable orderTotal data. However, looking at the DSS (Fig. 3), we see that this can be obtained from price which in turn can be obtained using decomposition of bodySearch, which is available. The idea for horizontal adaptation is to add such data transformations in the graphplan building process. Supported transformations are the DSS ones: decomp(d,D) if $D = \{d_i \mid d \lhd d_i\}$) (decomposition), comp(D,d) if $D = \{d_i \mid d \lhd d_i\}$) (composition), and cast(d1,d2) if $d_1 \sqsubset d_2$ (cast). Interestingly, one can have a task vision of these, *e.g.*, task cast above has precondition d_1 and postcondition d_2. Data adaptation planning steps are performed at the end of the basic planning steps and are directed toward the set of data missing for applicable services (here, {orderTotal} for Paypal and {amount, productId} for MPS).

Plan Extraction and Orchestration Generation. Plan extraction is achieved backtracking the graphplan from the user task output data. The l-CSS is used for filtering at extraction time and to ensure that extracted plans respect the CSS constraints. Three plans are generated for our example:

- @eBook;MobiPocket (in bold in Fig. 5),
- Amazon;decomp(bodySearch,{productId,price});cast(price,orderTotal);Paypal; comp({autorization,productId},token);MobiPocket (in bold in Fig. 6), and
- Amazon;decomp(bodySearch,{productId,price});cast(price,orderTotal); cast(orderTotal,amount);MPS;comp({autorization,productId},token);MobiPocket.

Plans are then transformed into YAWL orchestrators, as demonstrated for the second plan in Figure 7. Orchestrators have a single operation, named according to the user task capability. Variables are used for semantic data types (*e.g.*, title) and for messages (*e.g.*, AmazonloginIn, or AloginIn in Fig. 8). Plan control structures are expressed using sequence and flow. Conversations of selected services are integrated reversing them, a receive/reply couple being replaced by an invoke. Finally, assignments are used to encode cast, comp, and decomp tasks.

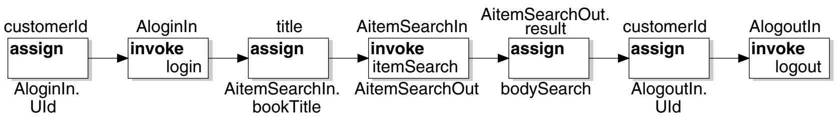

Fig. 8. Amazon conversation integration (YAWL)

4 Conclusion

In this paper we have proposed a technique that integrates adaptation features in the service composition process. We support both horizontal and vertical adaptation, which has been achieved combining semantic descriptions and hierarchical planning. We are also able to generate different composition solutions to the user task requirements, while ensuring they are correct from both data and semantics points of view. Our technique is fully automated thanks to GraphAdaptor, a prototype tool which takes as input a set of description files for user task, service and semantic structures, and outputs a YAWL file for each possible service composition. The main perspective of this work is the extension of our service model with conversations over capabilities and security features.

References

1. Canal, C., Poizat, P., Salaün, G.: Model-based Adaptation of Behavioural Mismatching Components. IEEE Transactions on Software Engineering 34(4), 546–563 (2008)
2. Beauche, S., Poizat, P.: Automated Service Composition with Adaptive Planning (long version). In: Poizat, P. Web page
3. Marconi, A., Pistore, M., Poccianti, P., Traverso, P.: Automated Web Service Composition at Work: the Amazon/MPS Case Study. In: Proc. of ICWS 2007 (2007)
4. Brogi, A., Popescu, R.: Automated Generation of BPEL Adapters. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
5. Peer, J.: Web Service Composition as AI Planning – a Survey. Technical report, University of St.Gallen (March 2005)
6. Berardi, D., Giacomo, G.D., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In: Proc. of ICSOC 2004, pp. 105–114. ACM, New York (2004)
7. Klush, M., Gerber, A., Schmidt, M.: Semantic Web Service Composition Planning with OWLS-Xplan. In: Proc. of the AAAI Fall Symposium on Agents and the Semantic Web (2005)
8. Brogi, A., Popescu, R.: Towards Semi-automated Workflow-based Aggregation of Web Services. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 214–227. Springer, Heidelberg (2005)
9. Constantinescu, I., Binder, W., Faltings, B.: Service Composition with Directories. In: Löwe, W., Südholt, M. (eds.) SC 2006. LNCS, vol. 4089, pp. 163–177. Springer, Heidelberg (2006)

10. Liu, Z., Ranganathan, A., Riabov, A.: Modeling Web Services using Semantic Graph Transformation to Aid Automatic Composition. In: Proc. of ICWS 2007 (2007)
11. Benigni, F., Brogi, A., Corfini, S.: Discovering Service Compositions that Feature a Desired Behaviour. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 56–68. Springer, Heidelberg (2007)
12. Ben Mokhtar, S., Georgantas, N., Issarny, V.: COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support. Journal of Systems and Software 80(12), 1941–1955 (2007)
13. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated Synthesis of Composite BPEL4WS Web Services. In: Proc. of ICWS 2006 (2006)
14. Motahari-Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-Automated Adaptation of Service Interactions. In: Proc. of WWW 2007, pp. 993–1002. ACM, New York (2007)
15. Inverardi, P., Tivoli, M.: Deadlock Free Software Architectures for COM/DCOM Applications. Journal of Systems and Software 65(3), 173–183 (2003)
16. Bracciali, A., Brogi, A., Canal, C.: A Formal Approach to Component Adaptation. Journal of Systems and Software 74(1), 45–54 (2005)
17. Lotem, A., Nau, D.S., Hendler, J.A.: Using Planning Graphs for Solving HTN Planning Problems. In: Proc. of AAAI/IAAI 1999 (1999)