Automatic Service Composition and Synthesis: the Roman Model

Diego Calvanese
Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
calvanese@inf.unibz.it

Giuseppe De Giacomo, Massimo Mecella, Maurizio Lenzerini, Fabio Patrizi
Dipartimento di Informatica e Sistemistica, SAPIENZA – Università di Roma, Italy
lastname@dis.uniroma1.it

Abstract

The promise of Web Service Computing is to use Web services as fundamental elements for realizing distributed applications/solutions. When no available service satisfies a desired specification, one might check whether (parts of) available services can be composed and orchestrated in order to realize the specification. The problem of automatic composition becomes especially interesting in the presence of conversational services. Among the various frameworks proposed in the literature, here we concentrate on the so called "Roman Model", where: (i) each service is formally specified as a transition system that captures its possible conversations with a generic client; (ii) the desired specification is a target service, described itself as a transition system; (iii) the aim is to synthesize an orchestrator realizing the target service by exploiting execution fragments of available services. The Roman Model well exemplifies what can be achieved by composing conversational services and, also, uncovers relationships with automated synthesis of reactive processes in Verification and AI Planning.

1 Introduction

Web services, or simply services, are modular applications that can be described, published, located, invoked, and composed over a variety of networks (including the Internet): any piece of code and any application component deployed on a system can be wrapped and transformed into a network-available service, by using standard (XML-based) languages and protocols (e.g., WSDL, SOAP, etc.). One of the interesting aspects is that this wrapping allows each program to export a simplified description of itself, which abstracts from irrelevant programming details. The promise of Web services is to enable the composition of new distributed applications/solutions: when no available service can satisfy a client request, (parts of) available services can be composed and orchestrated in order to satisfy the request itself.

The work on services has by now largely resolved the basic interoperability problems for service composition (e.g., standards such as WS-BPEL and WS-CDL exist and are widely supported in order to compose services), and designing programs, called orchestrators, that execute compositions by coordinating available services according to their exported description is the bread and butter of the service programmer [1].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

The availability of abstract descriptions of services, has been instrumental to devising automatic techniques for synthesizing service compositions and orchestrators. Several research lines have been opened to investigate this issue. Some works have concentrated on data-oriented services, by binding service composition to the work on data integration [21]. Other works have looked at process-oriented services, in which operations executed by the service have explicit effects on the system. Among these approaches, several consider *stateless* (a.k.a., atomic) services, in which the operations that can be invoked by the client do not depend on the history of interactions, as services do not retain any information about the state of such interactions. Much of this work relies on the literature on Planning in AI [30, 10, 12]. Others consider *stateful* services which impose some constraints on the possible sequences of operations (a.k.a., conversations) that a client can engage with the service. Composing stateful services poses additional challenges, as the composite service should be correct w.r.t. the possible conversations allowed by the component ones. Moreover, when dealing with composition, data (that typically are sent back and forward in the operation invocations and are manipulated by the service) usually play an important role. This work relies on research carried out in different areas, including research on Reasoning about Actions and Planning in AI, and research about Verification and Synthesis in Computer Science [11, 25, 18, 20].

In this paper, we focus on composition of process-oriented stateful services, in particular we consider the framework for service composition adopted in [5, 7, 8, 22, 16, 29], sometimes referred to as the "Roman Model" [19]. In the Roman Model, services are represented as transition systems (i.e., focusing on their dynamic behavior) and the composition aims at obtaining, given a (virtual) target service specifying a desired interaction with the client, an actual composite service that preserves such an interaction.

The Roman Model well exemplifies what can be achieved by composing stateful services, and allows to uncover relationships with automated synthesis of reactive processes in Verification and Planning in AI.

2 The Roman Model

Services in the Roman Model represent software modules capable of performing operations. They are *stateful*: a service, at each step, offers to its clients a choice of operations it can perform, based upon its own state; the client chooses one of the offered operations, and the service executes it, changing its state accordingly. Formally, a *service* is a transition system $\mathcal{S} = \langle \mathcal{O}, S, s^0, S^f, \varrho \rangle$, where: (*i*) $\mathcal{O}$ is the set of possible *operations* that the service recognizes; (*ii*) S is the finite set of service's *states*; (*iii*) $s^0 \in S$ is the *initial state*; (*iv*) $S^f \subseteq S$ is the set of *final states*, i.e., those states where the interaction with the service can be legally terminated by the client (though she does not need to); (*v*) $\varrho \subseteq S \times \mathcal{O} \times S$ is the service's *transition relation*, which accounts for its state changes. When $\langle s, o, s' \rangle \in \varrho$, we say that *transition* $s \xrightarrow{o} s'$ *is in* $\mathcal{S}$. Given a state $s \in S$, if there exists a transition $s \xrightarrow{o} s'$ in $\mathcal{S}$, then operation o is said to be *executable* in s. A transition $s \xrightarrow{o} s'$ in $\mathcal{S}$ denotes that s' is a possible successor state of s, when operation o is executed in s. Notice that we allow for *nondeterministic* services, that is, several transitions can take place when executing a given operation in a given state. So, when choosing the operation to execute next, the client of the service cannot be certain of which choices will be available later on, this depending on which transition actually takes place. In other words, nondeterministic services are only *partially controllable*. We say that a service $\mathcal{S}$ is *deterministic* iff there are no two distinct transitions $s \xrightarrow{o} s'$ and $s \xrightarrow{o} s''$ such that $s' \neq s''$. Notice that given a deterministic service's state and an executable operation in that state, *unique* next service's state is always known. That is, deterministic services are indeed *fully controllable* by just selecting the operation to perform next.

A *community of available services* $\mathcal{C} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_n \rangle$ consists of n nondeterministic available services that share the same operations $\mathcal{O}$. A *target service* is a desired *deterministic* service that shares the operations in $\mathcal{O}$. The requirement of being deterministic is due to the fact that we want such a service to be fully controllable by its clients. The goal of the composition in the Roman Model is to maintain with the client the same, possibly infinite, interaction that she would have with the (virtual) target service, by suitably orchestrating the (concrete) available services. An *orchestrator* is a system component able to activate, stop, and resume any of the available services, and to instruct them to execute an operation among those executable in their current state. Essentially, the orchestrator, at each step, will consider the operation chosen by the client (according to the target service) and delegate it to one of the services for which the operation is executable, on so on, possibly at infinitum. The aim of the orchestrator is to maintain the interaction with the client, as if it was interacting with the target service, without ever failing to be able to delegate an operation chosen by the client to one of the available services. We assume here that the orchestrator has *full observability* on the available services, that is, it can keep track (at runtime) of their current states. Although other choices are possible, full observability is the natural one in this context, since the available services, modeled through finite transition systems as above, are already suitable abstractions for *actual* modules: if details have to be hidden, this can be done directly within the abstract behaviors exposed by services, possibly exploiting nondeterminism.

Formally, an orchestrator is a *function* from (*i*) the *history* of the whole system (which includes the state trajectories of all available services and the trace of the operations chosen by the client, and executed by the services), and (*ii*) the *operation* currently chosen by the client, to the index i of the service $\mathcal{S}_i$ to which the operation has to be delegated. Intuitively, the orchestrator *realizes* a target service if and only if, at every step, given the current history of the system, it is able to delegate every operation executable by the target to one of the available services.

3 Composition techniques

The goal of service composition is to synthesize an orchestrator that realizes the target service by exploiting available services. Such problem is related to synthesis of reactive processes [27], where an environment (in our case, the available service community) is to be controlled by an automatically-generated controller (in our case, the orchestrator), so that a desired specification (in our case, mimicking the target service) is fulfilled.

The specific composition problem has been tackled with different techniques: at first by exploiting a reduction to Satisfiability in a well known logic of programs, namely PDL [5, 7, 4, 16]. Notably, Logics of Programs are tightly related to Description Logics, for which highly optimized satisfiability checkers exist (e.g., RacerPro, Pellet, FACT, etc.). More recently [23], the problem has been tackled by directly appealing to techniques for Linear Time Logic (LTL) synthesis [26], based on model checking of game structures for the so called *safety-games* (see also ATL [3, 2]). Another approach recently proposed is based on directly computing compositions by exploiting (variants of) the formal notion of simulation [9, 29, 23]. The two latter approaches promise both a high level of scalability, since in practice they can be based on symbolic model checking technologies. Here we concentrate on the simulation-based approach.

Let $\mathcal{C} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_n \rangle$ be a community of available services and $\mathcal{S}_t$ a target service, where $\mathcal{S}_i = \langle S_i, s_i^0, S_i^f, \varrho_i \rangle$, for $i \in \{t, 1 \ldots, n\}$. An *ND-simulation relation* of $\mathcal{S}_t$ by $\mathcal{C}$ is a relation $R \subseteq S_t \times S_1 \times \ldots \times S_n$ such that $\langle s_t, s_1, \ldots, s_n \rangle \in R$ implies that if $s_t \in S_t^f$ then $s_i \in S_i^f$, for $i \in \{1, \ldots, n\}$, and for each $o \in \mathcal{O}$, there exists a $k \in \{1, \ldots, n\}$ such that for all transitions $s_t \xrightarrow{o} s_t'$ in $\mathcal{S}_t$ we have that: (*i*) there exists a transition $s_k \xrightarrow{o} s_k'$ in $\mathcal{S}_k$; (*ii*) for all $s_k \xrightarrow{o} s_k'$ in $\mathcal{S}_k$, it holds that $\langle s_t', s_1, \ldots, s_k', \ldots, s_n \rangle \in R$. An ND-simulation is essentially a simulation between $\mathcal{S}_t$ and the asynchronous product of the services $\mathcal{S}_i$ in $\mathcal{C}$. However, differently from the usual notion of simulation, we need to take into account available services' nondeterminism. To this end, we require that (*i*) for each target service's operation an available service k can be selected to perform the operation and (*ii*) *all its successor states* are still included in the ND-simulation.

A state s_t *is ND-simulated by* $\langle s_1, \ldots, s_n \rangle$, denoted $s_t \preceq \langle s_1, \ldots, s_n \rangle$, if and only if there exists an ND-simulation R of $\mathcal{S}_t$ by $\mathcal{C}$ such that $\langle s_t, s_1, \ldots, s_n \rangle \in R$. Observe that this is a *coinductive definition*. As a result, the relation $\preceq$ is itself an ND-simulation, and is in fact the *largest ND-simulation relation*, i.e., all ND-simulation relations are contained in $\preceq$. It can be shown that there exists a compositions if and only if $s_t^0 \preceq \langle s_1^0, \ldots, s_n^0 \rangle$.

Synthesizing composition using simulation has a very interesting property: the maximal simulation $\preceq$ contains enough information to allow for extracting every possible composition, through a suitable choice function. This allows for devising compositions in a "just-in-time" fashion: we compute the maximal simulation then, based on it, we start executing the composition, choosing the next step according to criteria that can depend on information available at run-time (actual availability of services, network communication problems or cost, etc.), so that simulation is preserved. This, also, opens up the possibility of having failure resistant compositions that reactively or parsimoniously adjust to failures of available services, avoiding recomputing the whole composition from scratch [29].

4 Conclusion

Several extensions and variants of the model presented here have been studied, e.g.: forms of target service's loose specifications [6], lookahead [14], trust aware services [13], distributed orchestrators [28], shared environments or other infrastructure for communication among services [16, 15], data-aware services [4]. Also, the approach described in this paper is related to composition based on planning [25], where the crucial difference is the desired specification to realize: in the composition via planning, this is a desired state of affair to be reached after some interactions while, in our case, it amounts to indefinitely maintain the specified interaction itself.

We conclude by stressing out that *dealing with data* is certainly one of the most critical and difficult issues we currently face in service composition and, more generally, in process verification. Indeed, current verification and synthesis techniques apply to finite state systems, while the presence of data typically results in infinite states. Therefore, suitable means for *abstraction* from infinite to finite states are needed, and indeed virtually all results on combining data with processes are directly or indirectly based on such a notion [4, 17, 24].

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.

[3] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proc. of CAV 1998*.

[4] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proc. of VLDB 2005*.

[5] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-Services that export their behavior. In *Proc. of ICSOC 2003*.

[6] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of underspecified composite e-Services based on automated reasoning. In *Proc. of ICSOC 2004*.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *International Journal of Cooperative Information Systems*, 14(4):333–376, 2005.

[8] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Composition of services with nondeterministic observable behavior. In *Proc. of ICSOC 2005*.

[9] D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *Int. J. Found. Comput. Sci.*, 19(2):429–451, 2008.

[10] J. Blythe and J. L. Ambite, editors. *Proc. of ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[11] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proc. of WWW 2003*.

[12] J. Cardoso and A. Sheth. Introduction to semantic web services and web process composition. In *Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*.

[13] F. Cheikh, G. De Giacomo, and M. Mecella. Automatic web services composition in trustaware communities. In *Proc. of SWS 2006*.

[14] Z. Dang, O. H. Ibarra, and J. Su. On composition and lookahead delegation of e-services modeled by automata. *Theor. Comput. Sci.*, 341(1–3):344–363, 2005.

[15] G. De Giacomo, M. de Leoni, M. Mecella, and F. Patrizi. Automatic workflows composition of mobile services. In *Proc. of ICWS 2007*.

[16] G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI 2007*.

[17] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.

[18] C. Gerede, R. Hull, O. H. Ibarra, and J. Su. Automated composition of e-services: Lookaheads. In *Proc. of ICSOC 2004*.

[19] R. Hull. Web services composition: A story of models, automata, and logics. In *Proc. of ICWS 2005*.

[20] S. A. McIlraith and T. C. Son. Adapting Golog for composition of semantic web services. In *Proc. of KR 2002*.

[21] M. Michalowski, J. L. Ambite, C. A. Knoblock, S. Minton, S. Thakkar, and R. Tuchinda. Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

[22] A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proc. of FoSSaCS 2007*, 2007.

[23] F. Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA – Università di Roma, Dipartimento di Informatica e Sistemistica, 2008. To appear.

[24] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *Proc. of IJCAI 2005*.

[25] M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proc. of ICAPS 2005*.

[26] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive designs. In *Proc. of VMCAI 2006*.

[27] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL'89*.

[28] S. Sardina, F. Patrizi, and G. De Giacomo. Automaticsynthesis of a global behavior from multiple distributed behaviors. In *Proc. of AAAI 2007*.

[29] S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc. of KR 2008*.

[30] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proc. of ISWC 2003*.