Composing Web Services with Nondeterministic Behavior

Daniela Berardi, Giuseppe De Giacomo, Massimo Mecella
Università di Roma "La Sapienza", Dipartimento di Informatica e Sistemistica
{berardi,degiacomo,mecella}@dis.uniroma1.it

Diego Calvanese
Libera Università di Bolzano/Bozen, Facoltà di Scienze e Tecnologie Informatiche
calvanese@inf.unibz.it

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 such a request. Service composition involves two different issues: the *synthesis*, in order to synthesize, either manually or automatically, a specification of how coordinating the component services to fulfill the client request, and the *orchestration*, i.e., how executing the previous obtained specification by suitably supervising and monitoring both the control flow and the data flow among the involved services.

In this work, we address the automatic composition synthesis when the behavior of the available services is nondeterministic, and hence is not fully controllable by the orchestrator. The service behavior is modeled by the possible conversations the service can have with its clients. The presence of nondeterministic conversations stems naturally when modeling services in which the result of each interaction with its client on the state of the service can not be foreseen. Let us consider as an example, a service that allows buying items by credit card; after invoking the operation, the service can be in a state `payment_OK`, accepting the payment, or in a different state `payment_refused`, if the credit card is not valid, with not enough credit, etc. Note that the client of a nondeterministic service can invoke the operation but cannot control what is the result of it. In other words, the behavior of the service is partially controllable, and the orchestrator needs to cope with such partial controllability. Note also that if one observes the status in which the service is after an operation, then s/he understand which transition, among those nondeterministically possible in the previous state, has been undertaken by the service. We assume that the orchestrator can indeed observe states of the available services and take advantage of this in choosing how to continue a certain task [1].

From a formal point of view, we adhere to the setting proposed in [1, 2, 3] whose distinguished features can be summarized as follows.

- The available services are grouped together into a so called *community*.
- Services in the community share a common set of actions Σ, the *actions of the community*. In other words, each available service in the community exports its behavior to the community itself in terms of the actions in Σ (the actions recognized be the community).
- Each action in Σ denotes a (possibly complex) interaction between the service and a client, and as a result of such interaction the client may acquire new information (not necessarily modeled explicitly) that may be of help in choosing the next action to perform.
- The behavior of each available service is described in terms of a *finite transition system* (aka finite state machine) that makes use of the actions in Σ. Since in this work we assume that the behavior of the available services is nondeterministic, differently form [1, 2, 3], such a transition system are nondeterministic in general.
- The client request itself is expressed as a finite transition system that makes use of the actions in Σ. Such a transition system, called *target service*, is deterministic as in [1, 3], since we assume that there is no uncertainty on the behavior that the client want to realize through composition of the available services.
- The orchestrator has the ability of scheduling services on a *step-by-step* basis. Hence the orchestrator has the

[1] The reader should observe that also the standard proposal WSDL 2.0 has a similar point of view: the same operation can have multiple output messages (the `out message` and various `outfault messages`), and the client observe how the service behaved only after receiving a specific output message.

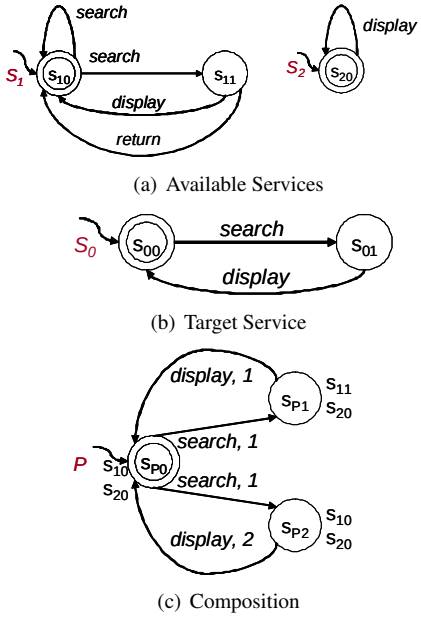

Figure 1. Composition of nondeterministic services

ability of *controlling the interleaving* of multiple services executed concurrently.

- The *composition synthesis* consists on synthesizing a program for the orchestrator such that by suitably scheduling the available services it can provide the target service to the client.

Figure 1(a) shows a community of services for getting information on books. The community includes two services: $\mathcal{S}_1$ that allows one to repeatedly *(i)* search the ISBN of a book given its title (`search`) then, *(ii)* in certain cases (e.g., if the record with cataloging data is currently accessible), it allows for displaying the cataloging data (such as editor information, year of publication, authors, copyrights, etc.) of the book with the selected ISBN (`display`), or *(iii)* simply returns without displaying information (`return`); $\mathcal{S}_2$ allows for repeatedly displaying cataloging data of books given the ISBN (`display`), without allowing researches. Figure 1(b) shows the target service $\mathcal{S}_0$: the client wants to have a service that allows him to search for a book ISBN given its title (`search`), and then display its cataloging data (`display`). Note that the client wants to display the cataloging data in any case and hence he/she can neither directly exploit $\mathcal{S}_1$ nor $\mathcal{S}_2$.

Figure 1(c) shows an orchestrator program P for available services $\mathcal{S}_1$ and $\mathcal{S}_2$ in Figure 1(a), that realizes the target service $\mathcal{S}_0$ in Figure 1(b). Essentially, P behaves as follows: it repeatedly delegates to $\mathcal{S}_1$ the action `search` (notice that both transitions labeled with this actions are delegated to $\mathcal{S}_1$); then it checks the resulting state of $\mathcal{S}_1$ and, depending on this state, it delegates the action `display` to either $\mathcal{S}_1$ or $\mathcal{S}_2$.

The contribution of this work is to devise a formal technique to perform automatic composition synthesis, when available services are nondeterministic and hence partially controllable by the orchestrator. In the extended version [4], we show that the technique proposed is sound, complete and terminating. Moreover we characterize the computational complexity of the problem and show that the proposed technique is optimal wrt (worst-case) computational complexity.

The technique proposed here is based on reduction to satisfiability in Propositional Dynamic Logic (PDL) with a limited use of the reflexive-transitive-closure operator[2]. Now, PDL satisfiability shares the same basic algorithms behind the success of the description logics-based reasoning systems used for OWL[3], such as FaCT[4], Racer[5], Pellet[6], and hence its applicability in the context of composition synthesis appears to be quite promising. Indeed a prototype implementation is available as open source software.

References

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

[2] ——, "Synthesis of underspecified composite e-Services based on automated reasoning", in *ICSOC 2004*.

[3] ——, "Automatic service composition based on behavioural descriptions", *Int. J. of Cooperative Information Systems*, vol. 14, no. 4, pp. 333–376, 2005.

[4] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella, "Automatic Composition of Web Services with Nondeterministic Behavior", Univ. Roma LA SAPIENZA, Technical Report, 2006.

[2] As in [1, 3], but more sophisticated this time in order to correctly deal with nondeterministic behavior of the available services.

[3] `http://www.omg.org/uml/`

[4] `http://www.cs.man.ac.uk/~horrocks/FaCT/`

[5] `http://www.sts.tu-harburg.de/~r.f.moeller/racer/`

[6] `http://www.mindswap.org/2003/pellet/`