An Introduction to Simulation-Based Techniques for Automated Service Composition

Fabio Patrizi
Dipartimento di Informatica e Sistemistica "A. Ruberti"
SAPIENZA Università di Roma
patrizi@dis.uniroma1.it

This work is an introduction to the author's contributions to the SOC area, resulting from his PhD research activity. It focuses on the problem of automatically composing a desired service, given a set of available ones and a target specification. As for description, services are represented as finite-state transition systems, so to provide an abstract account of their behavior, seen as the set of possible conversations with external clients. In addition, the presence of a finite shared memory is considered, that services can interact with and which provides a basic form of communication. Rather than describing technical details, we offer an informal overview of the whole work, and refer the reader to the original papers, referenced throughout this work, for all details.

1 Introduction

This work provides an overview of author's contributions to the SOC area, resulting from his PhD research activity [27], that can be summarized in the proposal of a novel technique for automated composition of *conversational* services, which is optimal (wrt time-complexity) and overcomes many of the obstacles encountered by similar existing proposals. The paper focuses on *automated service composition*, that is, the problem of automatically combining a set of available services so as to meet a desired specification. Such a topic has a lot in common with other research areas, the most closely related being, probably, System Verification and Synthesis (e.g., [29]), but also others provide theoretical frameworks that service composition can be cast over, such as Planning in Artificial Intelligence (e.g., [28]), Reasoning About Actions (e.g., [24]) and even Data Integration (e.g., [35]), thus making available some research achievements that SOC research can benefit from. Due to this interdisciplinarity, several approaches have been proposed to model services, e.g., as atomic actions [24], finite state machines [9, 7, 6] or views over data [36], and to solve the corresponding composition problem. Here, we follow the same approach as in [7, 6] and adopt a *behavioral model* of services. Starting from such work, we *(i)* consider some additional features, such as *operation nondeterminism* and *presence of a shared memory* which allows for basic inter-service communication and, more importantly, *(ii)* introduce a novel solution technique based on the formal notion of *simulation relation* [25], which improves previous techniques in that it allows to compute the *whole set of solutions*, at no additional (worst-case) computational time cost.

1.1 Service Composition: an Overview

Let's take a closer look at service composition, starting from the classical architecture for Web Services. Typically, the parties involved in a web service-based session, besides the client, which can be a service itself, include two additional classes of entities, namely: a *service broker* and some *service providers*. The former is a central, well-known, registry which stores service descriptions and can be accessed by clients when searching for services that meet some desired requirements; the latter are organizations, such as companies, which make services actually available. Providers register their services to brokers and, when invoked, serve clients' requests. A typical session is as follows: *(i)* a client contacts one (or more) broker(s) and requests a service that meets a desired specification; *(ii)* if such a service is found then the broker refers the client to the service provider actually deploying the service; *(iii)* the client, which located the service, based on the broker's information, contacts the desired service and interacts with it. Two classical questions about SOC arise:

1. *How are services described?*
2. *What if the desired service is not found?*

The first question concerns *modeling*, i.e., the definition of a suitable service abstraction, able to capture aspects that can be relevant to client; the second one raises the problem of finding a constructive alternative to the trivial answer: "client's request cannot be fulfilled, unless someone develops and deploys a new service that meets the desired requirements". As one may expect, there exist many *correct* answers to such questions. We address both problems. On the one hand, we propose a conceptual model (not an actual language), that substantially enriches existing ones (e.g., [8]), able to capture service *behavior*, that is, which provides an abstraction of service evolution over time, representing their possible conversations with clients; on the other hand, on top of this model, we propose sound and complete techniques for *building* a solution that fulfills a client's needs, when possible, by combining other available services. In particular, such techniques are shown to be the *best one can do*, in that they return the most general solutions, while being optimal with respect to worst-case time complexity.

2 Describing Service Behavior

In the literature, several approaches to service modeling have been proposed. Rather than actual languages, such as WSDL, widely used to describe Web services, we focus on their *conceptual model*. We can say that WSDL has an underlying *atomic* conceptual model, specified in terms of input-output requirements. For instance, a service providing stock quotes of some market can be successfully described this way, with a single operation that returns the list of quotes. Such a model is useful in many situations, as its popularity over the Web witnesses, however, when more complex specifications need to be exported, it shows severe limitations. For instance, think of the same web service for stock quotes and assume that it provides quotations only to authenticated clients. In an input-output approach, one would describe two operations, say, `auth` and `quote`, as well as the respective data format necessary for interaction. Unfortunately, the input-output approach does not allow for *conversation specification*, i.e., for putting constraints on the order that operations should be executed in. A very natural constraint would be, e.g., requiring clients to authenticate before requesting quotes. Observe also that cases may exist where two services export a same set of operations but allow for different execution sequences. Since this last constraint is not captured by input-output approaches, such services would appear to clients as the same one. In a word, atomic conceptual models export service *interface* but not their *behavior*.

The need for a *behavioral* description of services is widely recognized (e.g., [3, 1, 21, 7]), yet, the community suffers from a lack of standard languages for this purpose. In this work, we follow the same approach as the so-called Roman Model ([20]), originally introduced in [7, 6], which proposed the abstraction of conversational services as deterministic transition systems, where each state brings information about both service's (relevant) past history and the potential (atomic) conversations that can be carried out with a client. Inspired by that, we propose a rich model, oriented to describe all *conversations* supported by services, that includes relevant features, such as nondeterminism and shared memory, and, thus, increases the set of actual scenarios that are captured.

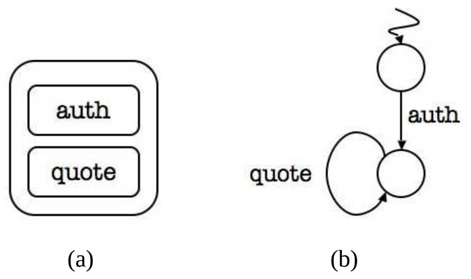

Figure 1: Two approaches to service description: 1(a) Input-output model; 1(b) Behavioral model.

In our model, services export their behavioral specifications by means of an abstract language that represents transition systems, i.e., Kripke structures whose transitions are labeled by service operations, under the assumption that each legal run of the system corresponds to a conversation supported by the service. To clarify this, consider Figures 1(a) and 1(b). The former is a graphical representation of an input-output description of the stock quote service with authentication, as described above, which provides information about which operations can be requested; the latter is a behavioral representation of the same service, where more information is provided: indeed, it tells clients that they *(i)* must authenticate before requesting a `quote` operation and, then, *(ii)* may request any number of quotes. Of course, more sophisticated examples do exist, where several operations, even nondeterministic, can be executed in a state, with nondeterminism modeling partial knowledge about service's internal logic. Also, there are settings relying on the same approach, where operations have parameters and are able to exchange data with other clients and even with an underlying database (cf. e.g., [5]).

A first advantage brought by such a model is its *generality* with respect to service integration, in the sense that it is abstract enough to serve as conceptual model for several classes of scenarios. As an example, it can be used to model web service applications as well as multi-agent system ones. As a consequence, results obtained from this model are also relevant to areas different from SOC. Second, from the SOC viewpoint, it provides a behavioral, stateful, service representation, which allows for describing those inter-operation (temporal) constraints that current languages, e.g., WSDL, do not capture. We remark the importance of such a feature in a perspective of composition automatization: indeed, composition engines are intended to replace human operators, who compose services based on their informal description, often provided in natural language, which includes behavioral information.

Importantly, when dealing with a behavioral model, we can look at services as high-level descriptions of software artifacts. Indeed, they are characterized by states and state transitions triggered by inputs, which, specifically, represent requested operations. This interpretation suggests, hence, to see service (possibly finite) runs as computation fragments, that can be suitably combined to generate more complex services.

3 Composing Services

Many works exist which deal with automated composition of *conversational* services, where service behavior is abstracted by various kinds of transition systems, e.g., [34, 4, 9, 37]. The closest one to our work is [4], that we take as a starting point, where the problem of automatically composing a set of services, described as possibly nondeterministic transition systems, is addressed. Although here we propose a significant extension of [4], in that we devise a novel solution technique which relies

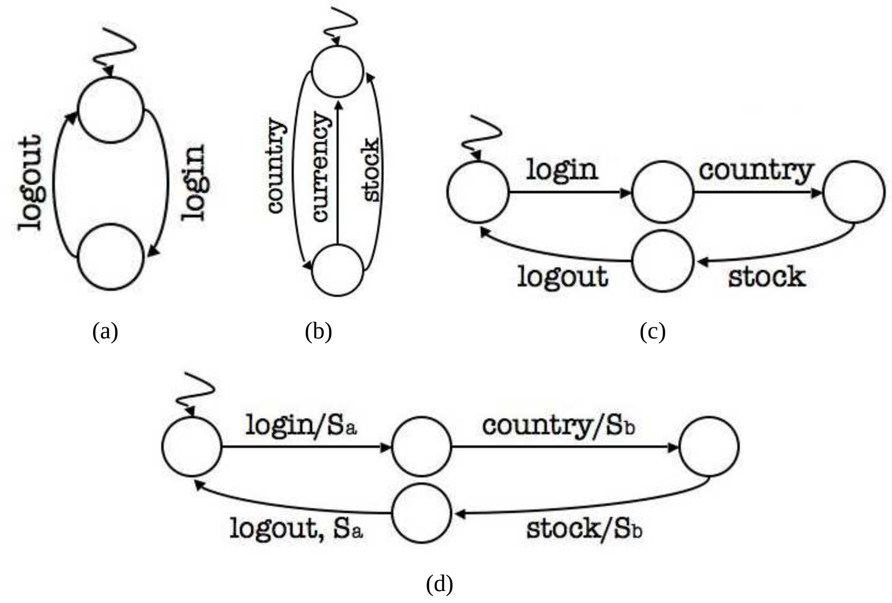

Figure 2: A service composition example in the Roman Model: 2(a),2(b) Available services; 2(c) Target service; 2(d) A Composition.

on effective technologies and yields great advantages, the basic problem has not changed. It can be informally stated as follows:

> Consider a set of available services, a.k.a. *community*, and an additional *target service*, all exporting their conversational behavior. Is it possible to coordinate the available services so to support, at execution time, all conversations supported by the target service?

In other words, the problem amounts to realize a (virtual) target service, by resorting only to (actual) available services. Obviously, how services are combined in the practice depends on the exported behavioral model. To see how this can be done under our model, consider the following example.

Example 3.1 *Figure 2 shows a service composition problem instance in the Roman Model, which includes two available services, represented in Subfigures 2(a) and 2(b), and a target one, in Subfigure 2(c). The one in Subfigure 2(a), say S_a, provides login/logout capabilities, allowing a client to be authenticated and to close an authenticated session, whereas the one in Subfigure 2(b), say S_b, provides market stock quotes from all over the world. Clients willing to interact with S_b are, first, required to input the market country of their interest and, then, are allowed to request either stock quotes or currency rates (versus, e.g., euro and dollar) for that market. As for the target service, say S_c, it provides stock quotes of a selected market only to authenticated clients. Specifically, clients of such service need first to login, then to select a market country, then are allowed to request quotes and, finally, to logout.*

As we said, target services are virtual, *that is, only their specification exists, whereas their implementation is missing. However, it is easily seen that, by resorting to available services, this example's target service can be built. Indeed, it is enough* delegating *login/logout operations to S_a and country selection and stock requests to S_b. Observe that the target service not only provides a set of operations, but imposes a set of constraints over their executions, e.g.,* `stock` *can be requested only after* `country` *has been executed. Since, on their side, also available service operations are subject to such kind of constraints, when a target service is to be realized, they must be met. For instance, had not S_c required* *operation* `country` *be executed before* `stock`*, it would be not realizable, as S_b is the only service that provides* `stock` *and it requires* `country` *to be executed first.*

In Example 3.1, the composition can be realized by a machine which, on the one side, receives client operation requests and, on the other side, forwards them to an appropriate available service which executes them and, consequently, changes its state, where a new set of operations becomes available. Such a machine, similar to a Mealy machine but that can be, in general, infinite-state, is called an *orchestrator*. One that solves the problem of Example 3.1 is shown in Figure 2(d). Each state of the machine corresponds to a state of the target service and each transition is labeled by a pair of the form $operation/service$, with an intuitive semantics: the requested operation is assigned to the output service. For instance, operation `login` is delegated to service S_a.

The example above shows how the existence of temporal constraints among operation executions makes the problem non trivial: each time an operation is to be delegated to some available services, one needs to check whether all constraints are fulfilled, i.e., whether the service chosen for delegation is in a state where the operation is actually executable. This makes the orchestrator construction a hard task, akin to an advanced form of conditional planning [17]. Indeed, in the Roman Model, the service composition problem is shown to be EXPTIME-complete [6, 26].

More complex scenarios can be considered. For instance, nondeterministic available services are also conceivable, where nondeterminism over operation execution represents partial knowledge about service's internal logic. Also, one could think of services communicating through a common blackboard or even exchanging data. All these scenarios require different notions of composition and, hence, different kind of orchestrators. *The aim of our work is to provide a formal model for them and to propose a respective solution technique for the resulting composition problem*.

Before providing details about the techniques for composition problem solution, let us mention a work [10], where the proposed technique applies to a more realistic scenario than those presented so far. In particular, it shows how a workflow, to be carried out by a team of cooperating agents, is realized as coordination, or more precisely *orchestration*, of several behaviors which provide high-level descriptions of agents' capabilities. Also, the approach is currently taken as a starting point for the development of a composition engine aimed at integrating embedded devices typically adopted in home automatization [1].

3.1 Solution Techniques

Once the service composition problem and its solution have been defined, the problem of finding solution techniques that can be automated becomes central.

3.1.1 PDL-based solutions

Previous work [4] addresses the problem of building a single orchestrator that is a solution to the problem. In such work, a technique was developed, able to deal with nondeterministic, finite-state, services, based on an encoding of the problem as a Propositional Dynamic Logic (PDL) [15] formula. Although such a technique is only able to build finite-state orchestrators, it is actually made effective by a crucial result showing that *if an orchestrator exists then there exists one which is finite* [4].

In a nutshell, PDLs constitute a family of logics that allow for specifying evolution of propositional properties over time, in response to events. PDL models are Kripke structures, as often happens when dealing with dynamic systems. Importantly, for each PDL logic there exists an *equivalent* Description

[1]This is part of recently started EU Project SM4All (FP7-224332).

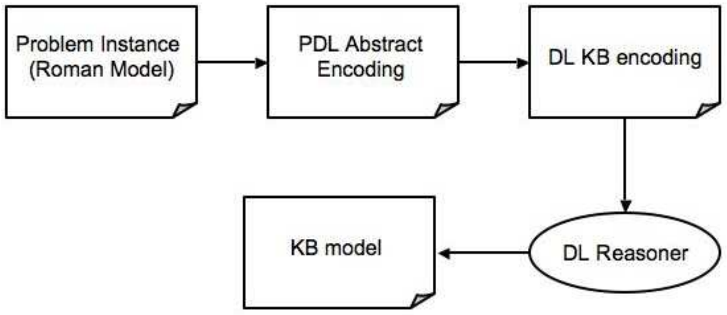

Figure 3: Conceptual schema of PDL-based approach to service composition

Logic (DL), i.e., a logic used for *static* knowledge representation, expressed in terms of *classes* and *relationships* among them [2], such that each model of the former is a model of the latter and vice versa [33, 11]. So, since *(i)* PDLs capture the Roman Model and *(ii)* effective DLs reasoning tools are available (e.g., FaCT [19], Racer [18], Pellet [2]), one can exploit DLs to represent and actually solve composition problems.

A conceptual schema of the PDL-based approach adopted in [4] is depicted in Figure 3. Starting from a description of the problem, where all involved services are described by transition systems, first an abstract PDL formula is generated such that *(i)* each of its models (which are finite-state) corresponds to a (finite-state) orchestrator that is a solution to the original problem and, vice versa, *(ii)* each composition problem's (finite-state) solution has a corresponding model of the PDL-formula. Then, such a formula is translated into a DL knowledge base, represented in an actual format suitable for a DL reasoner which, finally, generates a model of the knowledge base, if consistent. By construction, such a model is also a model of the original PDL formula which, in turn, corresponds to a composition problem's solution. Based on this approach, an actual tool, $\mathscr{ESC}$ (E-Service Composer), has been devised ([4]) which is able, with some limitations, to actually compute an orchestrator that realizes a target service. Unfortunately, this approach has three major drawbacks:

- only finite-state orchestrators are returned;
- the obtained solution is not *flexible*, that is, if a solution has been built which relies on an available service and such a service becomes unavailable at runtime, then the solution is no longer valid and the best one can do, using this approach, is to re-compute a new solution;
- on the practical side, due to implemented DL reasoner limitations, $\mathscr{ESC}$ is actually able to *synthesize* a model only for some particular inputs, though it is complete with respect to *checking* for the existence of a model.

These limitations constitute the essential motivations to our work.

3.1.2 Simulation-based solutions

We propose a novel solution technique based on the formal notion of simulation relation between transition systems [25]. Informally, given two transition systems S_1 and S_2, we say that S_1 *simulates* S_2 if it shows, at least, the same behaviors as S_2. For example, considering Figure 4, assume that S_1 is the

[2]http://clarkparsia.com/pellet

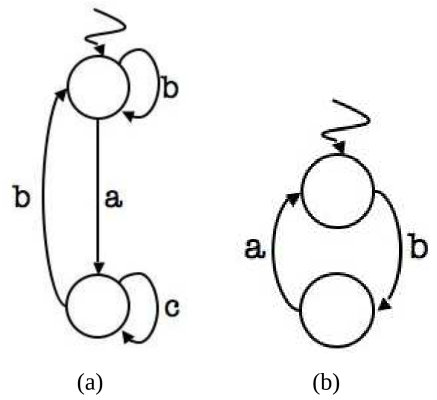

Figure 4: Two transition systems.

transition system shown in Subfigure 4(b) and S_2 is the one in Subfigure 4(a). Seen as services, S_1 simulates S_2, as each conversation supported by S_2, i.e., runs of the form $(\texttt{ab})^*$, is also supported by S_1. Of course, the vice versa does not hold: for instance, $(\texttt{b})^*$ is supported by S_1 but not by S_2. Although, here, we resort to regular languages as a means for describing service conversations, in our approach we do not adopt pure Finite State Machines (FSMs) as service models. Indeed, generic transition systems are better suited for our purposes, as we are interested in which *choices* a service actually provides in each state.

The following definition [25], where S^f is used to identify the set of a system's final states, provides the central notion that our work is built upon. Actually, the one which follows can be used for our purposes only when dealing with deterministic transition systems, whereas for nondeterministic ones, a different notion, namely *ND-simulation* is required. However, modulo formal details, the approach and the essential ideas remain the same.

Definition 3.1 *Given two transition systems $\mathscr{S}_t$ and $\mathscr{S}_\mathscr{C}$, a simulation relation of $\mathscr{S}_t$ by $\mathscr{S}_\mathscr{C}$ is a relation $R \subseteq S_t \times S_\mathscr{C}$, such that:*

$\langle s_t, s_\mathscr{C} \rangle \in R$ implies:

1. *if $s_t \in S_t^f$ then $s_\mathscr{C} \in S_\mathscr{C}^f$;*
2. *for each transition $s_t \xrightarrow{o} s_t'$ in $\mathscr{S}_t$ there exists a transition $s_\mathscr{C} \xrightarrow{o} s_\mathscr{C}'$ in $\mathscr{S}_\mathscr{C}$ and $R(s_t', s_\mathscr{C}')$.*

As it can be seen, this is a stronger relation than equivalence of FSMs, seen as language acceptors. Indeed, cases exist where two transition systems accept a same language but their states are not in simulation relation.

Essentially, our technique amounts to reducing the composition problem to the search for a simulation relation between the target service and the available service asynchronous product, which is itself a transition system. Precisely, it is the transition system which describes all the possible interleaved executions of available services or, in other words, represents all *potentialities* of the community services, seen as a whole. Recalling the above informal definition of a simulation relation, this corresponds to answering the question: *"can available services be combined in order to include the same behaviors as the target service?"*, which is precisely our problem.

A fundamental advantage brought by our technique is that simulation-based solutions are, in fact, *universal solutions*, or *composition generators*, i.e., finite structures that represent *all possible, even*

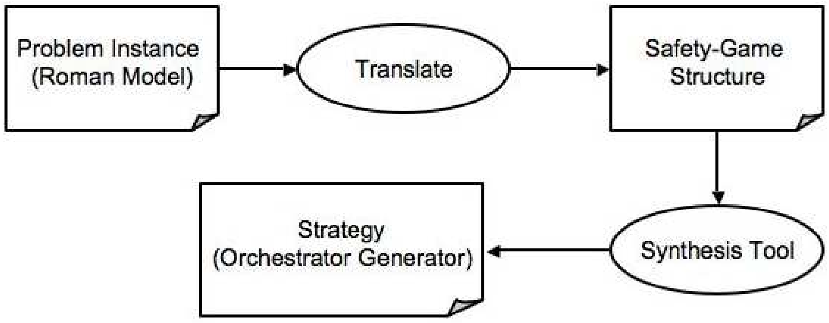
Problem Instance (Roman Model)
Translate
Safety-Game Structure
Synthesis Tool
Strategy (Orchestrator Generator)

Figure 5: Conceptual schema of the Game-based approach to service composition

infinite-state, orchestrators that realize a target service. Importantly, this does not affect worst-case time complexity: indeed, it is known that searching for a simulation between two transition systems is polynomial in the size of the systems, hence, since an asynchronous product result has exponential size with respect to its factor's size, it comes out that our technique requires exponential time with respect to the size of the (original) input systems. This, along with the observation that the service composition problem is proven EXPTIME-complete [26], yields that also our simulation-based technique is optimal with respect to worst-case time complexity. In fact, with respect to the PDL-based approach, we obtain a complexity characterization refinement.

But our approach yields two additional benefits.

First, by using composition generators, we obtain *flexible* solutions, that is, able to change their behavior based on information available at runtime. The PDL-based approach does not provide this feature, since it returns a single, *rigid*, solution that cannot be modified at runtime. For example if, during execution, an available service that the executing orchestrator relies on becomes unavailable, but can be replaced by another one, then with our solution we can change, without need for recomputation, the available service used to realize the target service. Differently put, we are able to *switch* orchestrators during execution.

Second, a set of effective tools, such as TLV [30], Lily [22] or Anzu [23], for computing simulation is becoming available. Precisely, such tools are aimed at synthesizing finite-state dynamic systems that meet desired temporal properties. In particular, they can be used to compute *winning strategies* for *safety games*, a.k.a. *invariant* games, i.e., games where a player, in order to win, is required to maintain a given property all along game evolution. Indeed, our work proposes a reduction of the service composition problem into a safety game, such that computing a winning strategy for the obtained game corresponds to computing an orchestrator generator for the original problem, as defined in the simulation-based context. A conceptual schema of such an approach is depicted in Figure 5. As synthesis engines are available, our efforts focused on defining the *Translation* module, which implements a procedure for automatic reduction of a service composition instance into a game structure. By doing so, we make available, for computing simulation relations, tools from the System Verification and Synthesis area, thus getting the major advantage of efficiency, as such tools resort to ordered binary decision diagrams (OBDD) for the internal representation of dynamic structures, thus limiting the typical state space explosion associated with synthesis procedures.

4 Further Issues

In this Section, we briefly discuss additional issues which constitute interesting research directions in service composition, that have been or are currently under investigation.

4.1 Dealing with Data

Data-Management capability is a desirable service property to capture: as a matter of fact, real services deal with data. Including such features in service representations would result in a more complete model, thus yielding the possibility of facing even more realistic problems. However, as several works witness, e.g., [5, 14, 13], the presence of data poses a major obstacle to service composition and verification: data is infinite by its nature and makes services infinite-state. Unfortunately, the techniques described above, as well as others proposed in the literature, are effective only when dealing with finite-state systems, whereas infinite-state system verification and synthesis are known as a hard task for most non-trivial properties, and undecidable for general ones. Consequently, introducing data in service composition frameworks has a great impact on the solution techniques that can be adopted, thus making the problem a major challenge for the SOC community.

In our work, we propose a model that allows for dynamic and finite-state data structure representation [3]. Precisely, in addition to available services, a community contains a so-called finite-state *data box*, i.e., an additional transition system modeling the evolution of a shared data structure whose state is affected by available services' execution, and that can be used to realize a basic form of inter-service communication. From a modeling perspective, a data box can be seen as a database with a finite-state behavior, used to capture some situations where services deal with data over finite domains. This way, we keep dealing with finite-state systems, thus making simulation-based techniques still applicable, but introduce a simple form of data-awareness.

Such an extension, besides increasing the set of scenarios that can be captured and, hence, making the model appealing also for other areas [4], provides the bases for more ambitious research, i.e., devising techniques for solving composition problem instances over generic data-aware services. The basic idea is *abstracting* over actual data: first, a symbolic representation of actual database instances is built by using only a finite set of symbols; then, the infinite-state synchronous evolution of both the available services and the database is described as a finite-state system. This way, one can reduce the original problem to one defined over finite-state symbolic structures, thus making the above techniques still applicable. One attempt in this direction, in a synthesis context, is the COLOMBO framework described in [5], where services are able to exchange data from infinite domains and to interact with a relational database, by reading/writing scalar values. As for service verification, [14, 13] propose two similar abstraction techniques allowing for data-aware service verification.

We observe that [5] tackles the composition problem by relying on a PDL-based approach. However, under the same model, one can recast the problem in terms of (data-aware) simulation, that is, by defining a relation between two data-aware services that interact with a common underlying data structure, whose data content may come from an infinite domain. This way, one would get the advantages brought by a simulation-based approach, though the actual resolution would be more complex, due to state space infiniteness, which calls for some abstraction procedure.

[3]Such a feature was first introduced in [12], in a different context.

[4]Cf., e.g., [31, 10] where multi-agent scenarios have been formulated as service composition ones

4.2 Distributed Orchestrators

In SOC, centralized scenarios, though conceivable, are rare. Even when all community services actually reside in a same repository, they communicate through common *middleware* that enables communication and interoperability. Hence, the resulting abstraction is still a distributed community, where services are seen as programs, located at different places, that interoperate through some protocol. In such a distributed context, there can be cases where a central coordinating entity, such as an orchestrator, is not realizable nor convenient. In [32, 10], two examples of such scenarios are reported. In particular, [10] shows one where all services execute on distributed peer devices, communicating via a wireless network. Clearly, in such situations one cannot rely on a single, central, coordinating unit: indeed, network disconnections are possible and can dramatically impact the overall system effectiveness. If, for instance, the coordinator permanently loses a connection, peers are no longer able to cooperate. Also less catastrophic events, such as temporary disconnections, may affect a system's efficiency and/or effectiveness.

In our work, we built a peer-based framework for distributed orchestrator execution, that is, for execution in a scenario where each service has an attached *local* orchestrator, able to communicate with other ones, in order to cooperate for achieving a common goal.
As discussed above, orchestrators make their choices, i.e., select available services for delegation, based on community current state, provided they have full observability on available service state. However, in this case, such assumption is no longer valid as, on the one hand, services reside at different places and, on the other hand and more importantly, there is no central entity able to *observe* their state. In fact, local orchestrators have full observability only on the service they are attached to, and know nothing about other services. Therefore, even if a technique for distributed execution of a centralized orchestrator were given, since, in general, services are nondeterministic, a problem of *state reconstruction* would arise. To solve this, we rely on the service communication ability, which is exploited to broadcast messages to other peers. Essentially, our technique amounts to building a usual orchestrator as if it were to be executed on a central coordinating device, and then, from this, generating one *local orchestrator* for each available (distributed) service. Local orchestrators replicate a centralized one's behavior except for two features: *(i)* they are able to send/receive messages that are useful for community state reconstruction, i.e., they make their decisions based on received messages and current state of the service they are attached to, and *(ii)* they issue commands only to the service they are attached to, while using received messages to *keep track* of current community state. Differently put, each local orchestrator, through received messages, *reconstructs* community current state and history until it is *its turn*, and only then activates the respective service by issuing the command requested by the client.

This technique, along with the possibility of executing sevices in parallel, and not only under an interleaving policy, contributes to make the service-based approach quite general and, thus, appealing also for research areas other than SOC, e.g., Multi Agent Systems (MAS).

5 Contributions

The main goal of our work is to address the drawbacks associated with a PDL-based solution approach to the service composition problem. In doing this, the achieved results introduce a set of novelties with respect to service composition state-of-the-art:

- We propose a rich conceptual model for service behavioral description that makes services abstract enough to represent a variety of dynamic, possibly communicating, entities such as web services, physical devices or agents.

- We devise a novel approach to service composition that exploits the formal notion of simulation, thus bringing the following major advantages:
  - a huge literature about simulation, and simulation-related problems is available, as it is a standard notion, widely and very well studied in Computer Science and related areas, thus providing access to a set of useful results, such as efficient procedures for computing simulation relations (e.g., [16]);
  - the notion of simulation relation is associated with System Verification and Synthesis, thus making effective tools, based on Model Checking approach, available. In fact, we resort to one such tools to build our composition engine.
- the proposed techniques overcome the obstacles met by the PDL-based approach:
  - they return the whole (in general, infinite) set of solutions (including infinite-state ones), represented as a *finite orchestrator generator*, without requiring additional computational efforts, with respect to previous approaches. This is undoubtedly the most relevant achievement, which shows the optimality of the approach.
  - *Just-in-time* solutions, able to adapt to run-time exceptional situations, can be built. With respect to the PDL-based approach, this corresponds to the possibility of *switching* orchestrators during target service realization. Importantly, one can take advantage of information available at runtime, such as unexpected faults, which were considered *catastrophic* in the previous approach;
  - we get a refinement of the complexity bound obtained in [4]. Precisely, we identify the exponential term as the number of available services, rather than their size.
- We provide actual techniques that take advantage of flexible solutions to efficiently recompute an orchestrator generator when some exceptional situations arise, such as temporal/permanent unavailability of available services;
- We propose a formulation of the composition problem in a distributed scenario and show how it can be solved. The problem is not introduced to face a *pure* service composition scenario, though it might be conceivable. Rather, it finds natural application in the Multi Agent System field, where several agents are described as possibly nondeterministic, communicating, finite-state transition systems (i.e., the same as services), operating in a common environment and communicating through a distributed shared memory;
- We show how the problem can be encoded into a safety-game structure, then written in an actual language, SMV, and, finally, processed by an implemented system, TLV, to compute the whole set of solutions;
- Based on above game structure encoding, an actual system, Web Service Composition Engine (WSCE) was devised, which exploits TLV for efficient problem solution.

References

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijai Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Boualem Benatallah, Fabio Casati, Farouk Toumani, and Rachid Hamadi. Conceptual Modeling of Web Service Conversations. In *CAiSE*, pages 449–467, 2003.

[4] Daniela Berardi. *Automatic Service Composition: Models, Techniques and Tools*. PhD thesis, Università degli Studi di Roma - La Sapienza, 2005.

[5] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Rick Hull, and Massimo Mecella. Automatic Composition of Transition-based Semantic Web Services with Messaging. In *Proc. of VLDB 2005*, 2005.

[6] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic Composition of e-Services that Export their Behavior. In *Proc. of ICSOC 2003*, pages 43–58, 2003.

[7] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. e-Service Composition by Description Logics Based Reasoning. In *Description Logics*, 2003.

[8] Daniela Berardi, Fahima Cheikh, Giuseppe De Giacomo, and Fabio Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19(2):429–451, 2008.

[9] Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation Specification: A New Approach to Design and Analysis of E-Service Composition. In *Proc. of WWW 2003*, 2003.

[10] Giuseppe De Giacomo, Massimiliano De Leoni, Massimo Mecella, and Fabio Patrizi. Automatic Workflows Composition of Mobile Services. In *ICWS*, pages 823–830, 2007.

[11] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *AAAI*, pages 205–212, 1994.

[12] Giuseppe De Giacomo and Sebastian Sardiña. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI 2007*, pages 1866–1871, 2007.

[13] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic Verification of Data-Centric Business Processes . In *Proc. of ICDT 2009*, 2009.

[14] Alin Deutsch, Liying Sui, and Victor Vianu. Specification and verification of data-driven web applications. *J. Comput. Syst. Sci.*, 73(3):442–474, 2007.

[15] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.

[16] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *J. Autom. Reasoning*, 31(1):73–103, 2003.

[17] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kauffman, 2004.

[18] Volker Haarslev and Ralf Möller. Description of the RACER System and its Applications. In *Description Logics*, 2001.

[19] Ian Horrocks. The FaCT System. In *TABLEAUX*, pages 307–312, 1998.

[20] Richard Hull. Web services composition: A story of models, automata, and logics. In *2005 IEEE International Conference on Services (SCC 2005)*, 2005.

[21] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-Services: a Look Behind the Curtain. In *Proc. of PODS 2003*, pages 1–14, 2003.

[22] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL synthesis. In *Proc. of FMCAD ’06*, pages 117–124, Washington, DC, USA, 2006. IEEE Computer Society.

[23] Barbara Jobstmann, Stefan Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *Proc. of CAV 2007*, pages 258–262, 2007.

[24] Sheila A. McIlraith and Tran Cao Son. Adapting Golog for Composition of Semantic Web Services. In *KR*, pages 482–496, 2002.

[25] Robin Milner. An algebraic definition of simulation between programs. In *Proc. of IJCAI 1971*, pages 481–489, 1971.

[26] Anca Muscholl and Igor Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.

[27] Fabio Patrizi. *Simulation-Based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA

Università degli Studi di Roma, 2009.

[28] Marco Pistore, Paolo Traverso, and Piergiorgio Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proc. of ICAPS 2005*, pages 2–11, 2005.

[29] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proc. of POPL 1989*, pages 179–190, 1989.

[30] Amir Pnueli and Elad Shahar. The TLV system and its applications. Technical report, Weizmann Institute, 1996.

[31] Sebastian Sardiña, Giuseppe De Giacomo, and Fabio Patrizi. Behavior composition in the presence of failure. In *Proc. of KR'08*, 2008.

[32] Sebastian Sardiña, Fabio Patrizi, and Giuseppe De Giacomo. Automatic synthesis of a global behavior from multiple distributed behaviors. In *Proc. of AAAI 2007*, pages 1063–1069, 2007.

[33] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *IJCAI*, pages 466–471, 1991.

[34] Maurice H. ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Formal Methods for Service Composition. *Annals of Mathematics, Computing and Teleinformatics*, 1(5):1–10, 2007.

[35] Snehal Thakkar, Josè Luis Ambite, and Craig A. Knoblock. A data integration approach to automatically composing and optimizing web services. In *In Proceedings of the ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

[36] Snehal Thakkar, Craig A. Knoblock, and Josè Luis Ambite. A view integration approach to dynamic composition of web services. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*, 2003.

[37] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *Proc. of ISWC-04*, volume 3298 of *LNCS*, pages 380–394. Springer, 2004.