Composition of Services that Share an Infinite-State Blackboard (Extended Abstract)

Fabio Patrizi and Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

1 Introduction

Recently, the issue of automatically building an orchestrator able to coordinate the execution of a set of available services so to realize a desired service has been investigated (cf. e.g., [Berardi *et al.*, 2003; Pistore *et al.*, 2005; Sardiña *et al.*, 2008; Lustig and Vardi, 2009]). One of the most well-known frameworks to tackle this problem is the so called Roman Model ([Hull, 2005; Berardi *et al.*, 2003]), where services are identified with their conversational behavior, represented by finite-state transition systems. Given a set of (possibly nondeterministic) available services and a deterministic target service, the *service composition problem* amounts to finding an *orchestrator* able to coordinate available services so to show the same behavior as the target service.

It has been shown that the above problem is deeply related to that of finding a simulation relation [Milner, 1971] between two transition systems[Berardi *et al.*, 2008; Sardiña *et al.*, 2008]. The obtained results essentially show that all possible orchestrators, even infinite-state ones, are generated from a (variant of a) simulation relation between the target service and the asynchronous product of available services. Such a result is also the basis for practical procedures to synthesize the orchestrator, based upon the hypothesis of state space finiteness. Even when a shared structure that available services interact with has been introduced, a.k.a. *environment* [De Giacomo and Sardiña, 2007] or *data box* [Patrizi, 2009], its behavior has been assumed finite-state.

In this paper we remove such assumption and propose a new framework where services are able to exchange values with a shared data structure, namely a *blackboard* (i.e., an associative list) that can assume infinite states. In particular, we assume that the blackboard has a bounded amount of space and that it allows a key-based access to its elements. Notably, the values that populate the blackboard range over an infinite domain Δ, which is *totally ordered and dense*. Observe that due to the infiniteness of Δ, the blackboard may traverse infinite states.

In particular, we assume that the community transition system (obtained as the asynchronous product of the transition systems of the available services) has already been computed, and we directly study the *core* problem of finding a *data-aware*-simulation relation of a determnistic transition system (the target) by a nondeterminstic transition system (the comunity of available services). Data-awareness comes from services' capability of interacting with the blackboard and exchanging data with it. In addition, transitions may be subject to particular conditions over blackboard's current state. Once the simulation is computed, then, from this, an orchestrator (generator) can be directly extracted, though we do not deal with such issue in this paper.

Our solution is inspired by [Berardi *et al.*, 2005; Deutsch *et al.*, 2007; 2009], where data are taken into account. As shown is such papers, the main difficulty one gets when infinite-state structures are introduced comes from the fact that usual techniques and tools for system verification and synthesis (e.g., [Burch *et al.*, 1992; Pnueli and Shahar, 1996; Jobstmann *et al.*, 2007; Piterman *et al.*, 2006]) are no longer applicable, as based on the key hypothesis of state space finiteness. We merge ideas from [Sardiña *et al.*, 2008; Berardi *et al.*, 2005; Deutsch *et al.*, 2007; 2009], and, starting from a problem formulation that involves an infinite-state blackboard, we show how an equivalent problem instance which is finite-state can be built, thus making above mentioned techniques still applicable.

2 The Framework

Based on previous achievements [Berardi *et al.*, 2008; Sardiña *et al.*, 2008], we consider the problem of searching for a simulation relation between two transition systems as the *core problem* of building an orchestrator that is a composition of a given target service.

In our setting, we have a countable infinite, dense and totally ordered universe Δ (e.g., alphanumeric strings with lexicographic ordering) with underlying order relation $\leq$. Such universe constitutes the *interpretation domain*[1] of all relational structures introduced in this work. We assume the existence of a *blackboard* B that services can interact with and which, in turn, affects services' evolution.

Definition 2.1 *A* blackboard *schema is a pair* $B = \langle R, b\rangle$ *where* R *is a binary relation schema* R *and* $b \in \mathbb{N}$ *is a* size bound. *Given* $B = \langle R, b\rangle$ *and a (interpretation) domain* Δ, *a* blackboard state $\bar{R}$ *over* Δ *is a finite, functional relation* $\bar{R} \subseteq \Delta^2$ *such that* $|\bar{R}| \leq b$.

A blackboard state $\bar{R}$ is also referred to as an R *interpretation*. The first component of each tuple in R is also called the *key*

[1] As usually referred to in Database Theory.

value (or simply *key*) of the tuple. For ease of exposition, we sometimes blur the notion of blackboard B and relation schema R, by often referring, by slight abuse of notation, to the former as to the latter and viceversa.

We introduce two additional basic notions.

Definition 2.2 (Active domain) *Let R be a relation schema and $\bar{R}$ a finite extension of R over a generic domain Δ. The* active domain *of $\bar{R}$ is the set $adom(\bar{R}) \subseteq \Delta$ of all and only Δ elements appearing in some R's tuple.*

Such notion naturally extends to a set of relations $R_1, \ldots, R_n$: $adom(R_1, \ldots, R_n) = adom(R_1) \cup \ldots \cup adom(R_n)$.

Definition 2.3 (Relation restriction) *Let $\bar{R}$ be generic relation extension of arity a. Given a generic set S of elements, the* restriction of R to S, *denoted $\bar{R}|_S$, is the set $\bar{R}|_S = \{\langle r_1, \ldots, r_a\rangle \in \bar{R} \mid r_i \in S \; \forall i = 1, \ldots, a\}$.*

In the following, we will need to represent properties of blackboard states. To this end, we define a *language* of *conditions*.

Definition 2.4 (Conditions) *Given a totally ordered dense domain Δ, with order relation $\leq$, and a binary relation schema R, a parametric* condition *over R and Δ is an expression of the following form:*

- *the constant $\top$ or $\bot$ (*constant term*);*
- *$R(\chi)$ (*functional term*), where χ is either the parameter (symbol) $p \notin \Delta$ or a constant from Δ;*
- *$\alpha \leq \beta$ (*ordering term*), where α and β can be either a functional term or the parameter p, or a constant from Δ;*
- *a boolean combination of above terms, taken as atoms.*

Parametric conditions are referred to as $\phi(p)$, p standing for the only parameter they can possibly contain. A non-parametric condition is a condition containing no occurrences of paramters and is referred to simply as ϕ. A guard *is a non-parametric condition.*

Conditions are evaluated against *(i)* total dense order relation $\leq$ over Δ and *(ii)* R interpretations over Δ.

Parametric conditions $\phi(p)$ are, in fact, formulae and, as such, need their parameter to be instanced, in order to be evaluated.

Definition 2.5 (Semantics of conditions) *Given a total dense order relation $\leq$ over Δ and an R interpretation $\bar{R}$ over Δ, the semantics of a non-parametric condition ϕ over Δ (, $\leq$) and R is as follows:*

- *if $\phi = \top$ then $\langle \bar{R}, \leq\rangle \models \phi$;*
- *if $\phi = \bot$ then $\langle \bar{R}, \leq\rangle \not\models \phi$;*
- *if $\phi = R(c)$, with $c \in \Delta$, then $\langle \bar{R}, \leq\rangle \models \phi$ iff $\exists c' \in \Delta \mid \langle c, c'\rangle \in \bar{R}$;*
- *if $\phi = \alpha \leq \beta$ then $\langle \bar{R}, \leq\rangle \models \phi$ iff $\langle \bar{R}, \leq\rangle \models \phi'$, where $\phi' = \alpha' \leq \beta'$ is the condition obtained from ϕ by replacing each term (i.e., α and β):*
  - *with itself, if it is a constant from Δ;*
  - *with $c' \in \Delta$, if it is a functional term $R(c)$ such that $\langle c, c'\rangle \in \bar{R}$;*
  - *with a default value $d \in \Delta$, otherwise;*
- *if $\phi = \neg\varphi$ then $\langle \bar{R}, \leq\rangle \models \phi$ iff $\langle \bar{R}, \leq\rangle \not\models \varphi$;*
- *if $\phi = \varphi_1 \wedge \varphi_2$ then $\langle \bar{R}, \leq\rangle \models \phi$ iff $\langle \bar{R}, \leq\rangle \models \varphi_1$ and $\langle \bar{R}, \leq\rangle \models \varphi_2$;*
- *if $\phi = \varphi_1 \vee \varphi_2$ then $\langle \bar{R}, \leq\rangle \models \phi$ iff $\langle \bar{R}, \leq\rangle \models \varphi_1$ or $\langle \bar{R}, \leq\rangle \models \varphi_2$.*

As for parametric conditions $\phi(p)$, if $c \in \Delta$ is the value assigned to p, then $\phi(c)$ is a non-parametric condition –whose semantics is defined above– obtained from $\phi(p)$, by replacing each occurrence of p with c.

Default value d in above definition is irrelevant and can be arbitrarily chosen. Observe that conditions (containing terms) of the form $\alpha \leq \beta$ where at least one between α and β is a functional term of the form $R(c)$ ($c \in \Delta$) have a special semantics: if c does not appear as a key in any $\bar{R}$'s tuple, then $R(c)$ is not defined and hence cannot be compared with any value. In order to be able to evaluate the condition also in this case, $R(c)$ is replaced by a default value d. Clearly, this situation is undesirable. To avoid this, one can first *validate* the condition by testing whether $R(c)$ holds, i.e., whether comparison $\alpha \leq \beta$ is meaningful. For instance, let $\alpha = R(c)$ and $\beta = c'$ ($c, c' \in \Delta$) and consider condition: $R(c) \wedge R(c) \leq c'$. Clearly, it evaluates to $\top$ iff c is a key of some $\bar{R}$'s tuple and $R(c) \leq c'$ wrt $\leq$ over Δ, which is a clear semantics.

Next, we introduce operations that can be executed on B.

Definition 2.6 (Atomic operations) *Given B and Δ as above, an* atomic operation *(over R and Δ) is any of the following expressions:* (i) *$\neg R(\chi)$ (deletion),* (ii) *$R(\chi) = \upsilon$ (insertion/modification)* (iii) *nop (empty operation), where χ and υ can be either the parameter q or a constant from Δ.*

Definition 2.7 (Operations) *Let B and Δ be as above. An* operation *$o \in \mathcal{O}$ (over R and Δ) is a set of pairs $o = \{\langle \phi_1(p), \nu_1(p)\rangle, \ldots, \langle \phi_m(p), \nu_m(p)\rangle\}$, where each $\phi_i(p)$ is a (parametric) condition over Δ and R, and $\nu_i(p)$ is a finite sequence of atomic operations (over R and Δ).*

We assume a finite set of operations $\mathcal{O} = \{o_1, \ldots, o_n\}$ that can be executed on B. Semantics of general operations, under parameter instantiation, is defined based on atomic's one.

Definition 2.8 (Semantics of atomic operations) *Let B and Δ be as above and consider a B state $\bar{R}$ over Δ. Given an atomic operation (with actual parameter) $op(\underline{p}) \equiv \neg R(\underline{p})$ or $op(\underline{p}) \equiv R(\underline{p}) = v$ or $op(\underline{p}) = nop$, where $\underline{p}, v \in \Delta$, a B state $\bar{R}'$ is an $\bar{R}$* atomic successor under *$op(\underline{p})$, denoted $\bar{R} \xrightarrow{op(\underline{p})} \bar{R}'$, iff:*

- *if $op(\underline{p}) \equiv \neg R(\underline{p})$, then $\langle r, r'\rangle \in \bar{R}'$ iff $\langle r, r'\rangle \in \bar{R}$ and $r \neq \underline{p}$;*
- *if $op(\underline{p}) \equiv R(\underline{p}) = v$, then $\langle r, r'\rangle \in \bar{R}'$ iff* (i) *$\langle r, r'\rangle \in \bar{R}$ and $r \neq \underline{p}$ or* (ii) *$\langle r, r'\rangle = \langle \underline{p}, v\rangle$;*
- *if $op(\underline{p}) \equiv nop$, then $\bar{R}' = \bar{R}$;*

Semantics of operations $op(p)$, where $p \notin \Delta$, is defined as above, given a constant $\underline{p} \in \Delta$ intended to replace all occurrences of p, thus obtaining $op(\underline{p})$ with no parameters.

Definition 2.9 (Semantics of operations) *Let R and Δ be as above and consider an R interpretation $\bar{R}$ over Δ. Given an operation $o = \{\langle \phi_1(p), \nu_1(p)\rangle, \ldots, \langle \phi_m(p), \nu_m(p)\rangle\}$ and a value $\underline{p} \in \Delta$, an R interpretation $\bar{R}'$ is an $\bar{R}$* successor under $\langle o, \underline{p}\rangle$, *denoted $\bar{R} \xrightarrow{o,\underline{p}} \bar{R}'$, iff:*

- *there exists $i \in \{1, \ldots, m\}$ such that $\langle \bar{R}, \leq\rangle \models \phi_i(\underline{p})$;*
- *for i as above and $\nu_i(\underline{p}) = op_1(\underline{p}) \cdots op_{l_i}(\underline{p})$ ($l_i > 0$), there exist $\bar{R}_1, \ldots, \bar{R}_{l_i}$ such that* (i) $\bar{R} \xrightarrow{op_1(\underline{p})} \bar{R}_1$, (ii) $\bar{R}_j \xrightarrow{op_j(\underline{p})} \bar{R}_{j+1}$ *for $j = 1, \ldots, l_i$ and* (iii) $\bar{R}_{j+1} = \bar{R}'$.

We can now define *services*. Intuitively, they represent high-level behaviors that interact with a blackboard B, through operations in $\mathcal{O}$.

Definition 2.10 (Services) *Given Δ and B as above, a* nondeterministic, guarded, blackboard-aware service, *or simply* BA-service, *over B and Δ, is a tuple $\mathcal{S} = \langle O, S, s_0, G, \varrho, S^f\rangle$, where:*

- *$O \subseteq \mathcal{O}$ is a finite set of service's* operation specifications*;*
- *S is the finite set of service's* states*;*
- *$s_0 \in S$ is the (single) service's* initial state*;*
- *G is the finite set of service's* guards *over R and Δ;*
- *$\varrho \subseteq S \times G \times O \times S$ is the service's* transition relation. *We freely interchange expressions $\langle s, g, o, s'\rangle \in \varrho$ and "$s \xrightarrow{g,o} s'$ (is) in $\mathcal{S}$";*
- *$S^f \subseteq S$ is the finite set of service's* final states*;*

Moreover, for convenience, for each service, we define the finite set C of all constants appearing either in (i) *a condition or effect of some operation specification with formal parameters $o \in \mathcal{O}$, or* (ii) *in some guard.*

Service transitions depend on the chosen operation and not on the actual parameter which is provided only at runtime.

We are interested in checking whether, and *how*, given two services over a same blackboard B, one can *simulate*, i.e., show the same behavior as, the other one. Informally, the problem amounts to check whether a service is at least as capable as the other one (that is the *simulated* one). In the following Section, we clarify such notion, by providing a formal statement of the problem.

3 Simulation

We start by introducing our notion of simulation relation between two services defined over the same blackboard.

Definition 3.1 *Consider a possibly nondeterministic service $\mathcal{S}_\mathcal{C}$ and a deterministic one $\mathcal{S}_t$, both defined over the same blackboard B. A* blackboard-aware *simulation relation, BA-simulation for short, of $\mathcal{S}_t$ by $\mathcal{S}_\mathcal{C}$ over B and Δ is a relation $\Sigma \subseteq S_t \times S \times \Delta^2$ such that $\langle s_t, s, \bar{R}\rangle \in \Sigma$ implies:*

1. *$\bar{R}$ is a B state (i.e., such that $|adom(\bar{R})| \leq b$);*
2. *if $s_t \in S_t^f$ then $s \in S^f$;*
3. *for each transition $s_t \xrightarrow{g_t,o} s_t'$ in $\mathcal{S}_t$ s.t. $\langle \bar{R}, \leq\rangle \models g_t$ and for each actual parameter $\underline{p} \in \Delta$ such that for some $\bar{R}'$, $R \xrightarrow{o,\underline{p}} \bar{R}'$:*
   (a) *there exists a transition $s \xrightarrow{g,o} s'$ in $\mathcal{S}$ such that $\langle \bar{R}, \leq\rangle \models g$;*
   (b) *for all transitions* (i) $s \xrightarrow{g,o} s'$ *in $\mathcal{S}$ as above and* (ii) *all $\bar{R} \xrightarrow{o,\underline{p}} \bar{R}''$, $\langle s_t', s', \bar{R}''\rangle \in \Sigma$ holds.*

Analogously to a (ND-)simulation relation (cf. [Berardi *et al.*, 2008; Sardiña *et al.*, 2008]), a BA-simulation relation is intended to capture the ability of $\mathcal{S}_\mathcal{C}$ to mimicking $\mathcal{S}_t$ behavior on B.

We are now ready to formally state our problem:

Definition 3.2 (Core DA-service composition problem) *Consider a blackboard schema B whose states are defined over an infinite, countable, totally ordered and dense domain Δ. In addition, let* (i) *$\mathcal{S}_t$ be a deterministic (*target*) DA-service over B and Δ and* (ii) *$\mathcal{S}_\mathcal{C}$ be another, possibly nondeterministic, DA-service over B and Δ. The* core DA-service composition problem *is the problem of finding a BA-simulation relation of $\mathcal{S}_t$ by $\mathcal{S}_\mathcal{C}$ on B and Δ.*

Clearly, BA-simulation relations are, in general, infinite, due to infiniteness of Δ, that input parameters range over, which yields an infinite set of B states. Hence, procedures based on iterative fixpoint computations, such as the one defined in, e.g., [Sardiña *et al.*, 2008], are no longer useful, since there is no finite bound on the number of iterations. However, through an abstraction procedure, one can focus only on a finite set of actual values plus some symbolic values, so to reduce the problem to, essentially, searching for a simulation relation in a setting where states and data are finite.

4 Symbolic Simulation

We define a *symbolic simulation*, i.e., a simulation relation with additional information about value ordering. Such structure is, indeed, a *finite representation of an infinite simulation relation*. As a consequence, one can compute the simulation on this, rather than on an infinite structure (of course, this is not required if Δ is finite).

Definition 4.1 *Consider two services $\mathcal{S}, \mathcal{S}_t$ defined over a same blackboard $B = \langle R, b\rangle$. Let $C = C_t \cup C_\mathcal{C} = \{c_1, \ldots, c_m\}$ be the (finite) set of constants contained in either operations or guards appearing, in turn, in either $\mathcal{S}_t$ or $\mathcal{S}_\mathcal{C}$. Let $\hat{\Delta} = \{a_0, \ldots, a_{2b}\} \cup C$, where $\{a_0, \ldots, a_{2b}\} \cap \Delta = \emptyset$. A* symbolic *BA-simulation relation is a relation $\hat{\Sigma} \subseteq S_t \times S \times \hat{\Delta}^2 \times \hat{\Delta}^2$ such that $\langle s_t, s, \hat{R}, \hat{\leq}\rangle \in \hat{\Sigma}$ implies:*

1. *$\hat{R}$ is an R interpretation over $\hat{\Delta}$ such that $|adom(\hat{R})| \leq b$;*
2. *$\hat{\leq}$ is a total order relation over $adom(\hat{R}) \cup C$, such that $\hat{\leq}|_C = \leq|_C$;*

3. *for each total order relation $\hat{\leq}_{\underline{p}}$ obtained by extending $\hat{\leq}$ with a value $\underline{p} \in \hat{\Delta}$ (possibly contained in $adom(\hat{R}) \cup C$) such that $\hat{\leq}_{\underline{p}}|_{adom(\hat{\leq})} = \hat{\leq}$, for each transition $s_t \xrightarrow{g_t,o} s'_t$ in $\mathcal{S}_t$ with $\langle \hat{R}, \hat{\leq} \rangle \models g_t$ and (for some $\hat{R}'$) $\hat{R} \xrightarrow{o,\underline{p}} \hat{R}'$, where operation conditions are evaluated over $\hat{\leq}_{\underline{p}}$, the following holds:*

   (a) *there exists a transition $s \xrightarrow{g,o} s'$ in $\mathcal{S}$ such that $\langle \hat{R}, \hat{\leq} \rangle \models g$;*

   (b) *for all transitions* (i) $s \xrightarrow{g,o} s'$ *in $\mathcal{S}$, as above, and* (ii) *all $\hat{R} \xrightarrow{o,\underline{p}} \hat{R}''$, as above, $\langle s'_t, s', \hat{R}'', \hat{\leq}' \rangle \in \hat{\Sigma}$, where $\hat{\leq}|_F = \hat{\leq}'|_F$, for $F = adom(\hat{\leq}) \cap adom(\hat{\leq}')$.*

A symbolic BA-simulation relation is intended to capture only the information actually relevant in a BA-simulation relation: *(i)* the state that each service is in, *(ii)* the whole set of constants appearing in either $\mathcal{S}_t$ or $\mathcal{S}_\mathcal{C}$ specifications and *(iii)* instead of the *actual* values contained in $\bar{R}$, their mutual order and their relationships with constants from $\mathcal{S}_t$ and $\mathcal{S}_\mathcal{C}$. Indeed, it can be shown that, as long as constant values from $\mathcal{S}_t$ and $\mathcal{S}_\mathcal{C}$ specification are kept unchanged, renaming values in $adom(\bar{R})$ while preserving the mutual order does not affect condition evaluations and, consequently, the possibility of executing operations and their effects on the blackboard. Since the blackboard contains at most $2b$ elements, some of which may come from C, with our abstraction approach one can provide an abstract description of each original BA-simulation tuple, using at most $2b + 1$ symbolic values, the additional one being intended to represent the actual input parameter.

The following theorem shows that one can exploit a symbolic BA-simulation relation in order to solve the BA-service composition core problem.

Theorem 4.1 *Given a blackboard B and an interpretation domain Δ as above, plus a target service $\mathcal{S}_t$ and an additional service $\mathcal{S}_\mathcal{C}$ over B, consider a (possibly infinite) simulation relation Σ of $\mathcal{S}_t$ by $\mathcal{S}_\mathcal{C}$ over B. There exists a corresponding symbolic simulation relation $\hat{\Sigma}$ of $\mathcal{S}_t$ by $\mathcal{S}_\mathcal{C}$ over B, with $\hat{\Delta} = \{a_0, \ldots, a_{2b}\} \cup C$, where $C = \{c_1, \ldots, c_m\}$ is the set of all constants in Δ occurring in $\mathcal{S}_t$ or $\mathcal{S}_\mathcal{C}$ specifications.*

Proof: *(Hint) The proof is based on building, for each tuple $t = \langle s_t, s, \bar{R} \rangle \in \Sigma$, a mapping $h : adom(\bar{R}) \cup C \longrightarrow \hat{\Delta}$, which is the identity on C, such that* (i) *$\bar{R}$ isomorphically maps into a relation $\hat{R}$ and* (ii) *$\leq|_{adom(\bar{R}) \cup C}$ isomorphically maps into $\hat{\leq}$. Observe that this is made possible by $\bar{R}$ boundedness. Starting from this, one shows that the so obtained symbolic tuple $\hat{t} = \langle s_t, s, \hat{R}, \hat{\leq} \rangle$ is* isomorphic *to $t = \langle s_t, s, \bar{R} \rangle$, in the sense that* (i) *an operation, possibly with actual parameter $\underline{p}$, is executable on s_t, s and $\bar{R}$ iff it is executable on s_t, s and $\hat{R}$ when all conditions are evaluated over $\hat{\leq}_{\underline{p}}$ and* (ii) *that the obtained* successor *tuples $t' = \langle s'_t, s', \bar{R}' \rangle$ and $\hat{t} = \langle s'_t, s', \hat{R}', \hat{\leq}' \rangle$ are pairwise isomorphic in the same sense. In other words, one can partition the infinite set of Σ elements into a finite set of equivalence classes, represented by $\hat{\Sigma}$ elements.* □

Importantly, the viceversa also holds:

Theorem 4.2 *Given a blackboard B and an interpretation domain Δ as above, plus a target service $\mathcal{S}_t$ and an additional service $\mathcal{S}_\mathcal{C}$ over B, consider a symbolic BA-simulation relation $\hat{\Sigma}$ of $\mathcal{S}_t$ by $\mathcal{S}$ on B. There exists a corresponding BA-simulation relation Σ of $\mathcal{S}_t$ by $\mathcal{S}_\mathcal{C}$ over B.*

Proof: *(Hint) The proof is based on two steps. First, from each $\hat{\Sigma}$ tuple, a set of* isomorphic *tuples is built through a mapping which is the identity on C and assigns each value in $adom(\hat{R})$ a value in Δ. Observe that this is made possible by $\leq$ density. Next, one shows, with an inductive argument, that the whole set of so obtained tuples is, in fact, a BA-simulation relation.* □

Above results show that, in order to solve the core problem, one can search for a symbolic BA-simulation instead of a non-symbolic one. Observe that the former is a finite structure which, apart from minor details, is, essentially, a BA-simulation relation itself (over finite interpretation universe $\hat{\Delta}$), in the sense of [Sardiña *et al.*, 2008]. Hence, by relying on iterative procedures for fixpoint computation, one can build a BA-simulation relation by actually building a symbolic BA-simulation.

5 Discussion

[Berardi *et al.*, 2008] shows how from a simulation relation an *orchestrator generator* can be built. It is, essentially, an enriched Mealy machine which, given *(i)* a state s of the service community, *(ii)* a state s_t of the target service simulated by s, and *(iii)* an operation o supposed to be executed by $\mathcal{S}_t$, outputs the whole set of available services that, in s, are actually able to execute o (there exists at least one) so that the simulation relation is preserved also in all possible successor states. Such a machine plays a central role as, from this, one can easily generate all orchestrators which solve the composition problem, by just picking up, at each step, one of the available services returned by the orchestrator generator.

When dealing with finite-state services, given the simulation relation, computing the orchestrator generator is a straightforward task: it simply amounts to enumerate all simulation states and, for each of them, compute all *good* services, i.e., all possible *witnesses* proving that s simulates s_t (which requires checking for local conditions only). However, in a setting where the whole community shows an infinite-state behavior, a different approach needs to be adopted, since state enumeration is clearly unfeasible.

As we briefly discussed in proof sketches of Theorems 4.1 and 4.2, there exists a correspondence between actual and symbolic BA-simulations, identified by a set of isomorphisms between actual and symbolic relations' elements. Based on this, rather than computing the actual orchestrator generator starting from the actual BA-simulation relation, we build a *symbolic* BA-orchestrator generator starting from a symbolic BA-simulation relation. Then, at execution time, given current community, target and blackboard states, plus the operation with actual parameter to be executed, in order to select

a service for operation execution, one transforms, through a proper isomorphism, current states into symbolic ones and, then, selects a *good* service from those returned by the symbolic orchestrator. Theorems 4.1 and 4.2 guarantee that such a choice is *sound*, in the sense that the service chosen in the symbolic context will be able to execute the requested operation in the concrete context, too. Also, we get that an orchestrator exists if and only if it can be generated in this way, which shows that the procedure is also *complete*.

Observe that reducing the service composition problem in the presence of an infinite-state blackboard to the finite case makes all previous results about composition by simulation applicable to this case, and, in addition allows a set of automated tools for system verification and synthesis (e.g., TLV [Pnueli and Shahar, 1996] or ANZU [Jobstmann *et al.*, 2007]) to be exploited for actual solution computation.

References

[Berardi *et al.*, 2003] 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.

[Berardi *et al.*, 2005] 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.

[Berardi *et al.*, 2008] 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.

[Burch *et al.*, 1992] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[De Giacomo and Sardiña, 2007] 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.

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

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

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

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

[Lustig and Vardi, 2009] Yoad Lustig and Moshe Y. Vardi. Synthesis from component libraries. In *FOSSACS*, pages 395–409, 2009.

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

[Patrizi, 2009] Fabio Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA, Univ. Roma, 2009.

[Pistore *et al.*, 2005] 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.

[Piterman *et al.*, 2006] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[Pnueli and Shahar, 1996] A. Pnueli and E. Shahar. The TLV system and its applications. Technical report, Weizmann Institute, 1996.

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