Service Composition and Safety Games*

(Lecture Notes for Elective in Software and Services)

Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica
Sapienza Università di Roma
fabio.patrizi@dis.uniroma1.it

December, 2009

This notes are an excerpt from Fabio Patrizi's Ph.D. thesis. They have been revisited by Fabio Patrizi to reduce missing references and to make them as much self-contained as possible. However, it is assumed that the reader is familiar with the notions of: *service composition problem*, *simulation* and *ND-simulation* relations, and all basic notions related to the framework for service composition usually known as "Roman Model".

1 Simulation and Safety Games

In this Section, we show how a service composition problem instance can be encoded into an equivalent game automaton and, correspondingly, how searching for a composition is equivalent to searching for a winning strategy (cf. [1, 2, 5]). The main motivation behind this approach is the availability of software systems, such as

- TLV(`http://www.cs.nyu.edu/acsys/tlv`) based on game structures,
- Ratzy (`http://rat.fbk.eu/ratsy`) based on game structures,
- Lily (`http://www.iaik.tugraz.at/content/research/design_verification/lily`) based on LTL synthesis,
- Anzu (`http://www.iaik.tugraz.at/content/research/design_verification/anzu`) based on game structures,
- Mocha (`http://mtc.epfl.ch/software-tools/mocha`) based on ATL,
- MCMAS (`http://www-lai.doc.ic.ac.uk/mcmas`) based on ATL,

*Excerpted from [4].

which provide *(i)* efficient procedures for strategy computation and *(ii)* convenient languages for representing the problem instance in a modular, intuitive and pretty straightforward way.

1.1 Safety-Game structures

Here, we specialize the *game structures* introduced in [5], used to represent and solve reactive system synthesis problems with invariant properties, to cope with our problem. Even though game structures have been studied also in other work (cf., e.g., [1, 2]), in this context we refer to [5] because the formalism introduced allows for a straightforward understanding of the implementation we will present later on. After introducing the specialization, we show how game structures can be used to describe a service composition problem.

Definition 1.1 *A* safety-game structure *(or $\Box$-game structure or $\Box$-GS, for short) is a tuple $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$, where:*

- $\mathcal{V} = \{v_1, \ldots, v_n\}$ *is a finite set of* state *variables, ranging over* finite *domains $V_1, \ldots, V_n$, respectively. A* valuation *of all variables in $\mathcal{V}$ is a tuple $u = \langle u_1, \ldots, u_n \rangle \in V = V_1 \times \ldots \times V_n$, associating value u_i to variable v_i $(i = 1, \ldots, n)$.*

- $\mathcal{X} \subseteq \mathcal{V}$ *is the set of* environment *variables. Without loss of generality, we assume that $\mathcal{X} = \{v_1, \ldots, v_m\}$ $(m \leq n)$. An* environment state *is a* valuation *of the variables in $\mathcal{X}$, that is, a tuple $\vec{x} = \langle x_1, \ldots, x_m \rangle \in X = V_1 \times \ldots \times V_m$, associating value x_i to environment variable v_i $(i = 1, \ldots, m)$.*

- $\mathcal{Y} = \mathcal{V}/\mathcal{X}$ *is the set of* system *variables. Without loss of generality, we assume that $\mathcal{Y} = \{v_{m+1}, \ldots, v_n\}$. A* system state *is a* valuation *of the variables in $\mathcal{Y}$, that is, a tuple $\vec{y} = \langle y_{m+1}, \ldots, y_n \rangle \in Y = V_{m+1} \times \ldots \times V_n$, associating value y_i to system variable v_i $(i = m+1, \ldots, n)$.*

- Θ *is a formula representing the* initial game states*, where a* game state *is a tuple $\vec{s} = \langle \vec{x}, \vec{y} \rangle \in X \times Y = V$. In details, Θ is a boolean formula without negation symbols, whose atoms are expressions of the form $(v_k = u)$, where $k \in \{1, \ldots, n\}$, $v_k \in \mathcal{V}$ and $u \in V_k$. Given a state $\vec{s} = \langle \vec{x}, \vec{y} \rangle \in V$, we write $\vec{s} \models \Theta$ ($\vec{s}$* satisfies *Θ) if and only if Θ evaluates to $\top$, once each of its atoms $(v_k = u)$ is replaced by* (i) *$\top$, if k-th component of $\vec{s}$ assumes value u, and* (ii) *$\bot$, otherwise.*

- $\rho_e \subseteq X \times Y \times X$ *is the* environment transition relation *which relates a* current game state *to a possible* next environment state*. In particular, if $\langle \vec{x}, \vec{y}, \vec{x}' \rangle \in \rho_e$, or, equivalently, $\rho_e(\vec{x}, \vec{y}, \vec{x}')$, then $\langle \vec{x}, \vec{y} \rangle$ is referred to as the* current game state *and $\vec{x}'$ as the* next environment state*.*

- $\rho_s \subseteq X \times Y \times X \times Y$ *is the* system transition relation*, which relates a current game state and a (next) environment state to a possible* next system state*. If* $\langle \vec{x}, \vec{y}, \vec{x}', \vec{y}' \rangle \in \rho_s$*, or, equivalently,* $\rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$*, then* $\langle \vec{x}, \vec{y} \rangle$ *is referred to as the* current game state*,* $\vec{x}'$ *as the* next environment state *and* $\vec{y}'$ *as the* next system state*;*

- $\Box\varphi$ *is a formula that represents the invariant property to be guaranteed. In particular,* φ *has the same form as* Θ *and* satisfaction *definition is as above.*

Intuitively, safety-game structures represent a game played by two adversaries: *system* and *environment*. A round is composed of two moves: an environment's followed by a system's, the former being a valuation of state variables and the latter being a valuation of system variables. Observe that, since state variables are, in fact, partitioned into system's and environment's, at the end of each round a game state is fully defined. Moreover, as state variables range over finite domains, game states are finite. Initially, the game can be in any (initial) state that satisfies Θ, according to above Definition 1.1. Players are not allowed to choose all possible valuations, but, assuming that, when a round starts, the game is in state $\vec{s} = \langle \vec{x}, \vec{y} \rangle$, must respect the following constraints:

- environment can choose any valuation $\vec{x}' \in X$ of environment variables such that $\rho_e(\vec{x}, \vec{y}, \vec{x}')$ holds;

- system can choose any valuation $\vec{y}' \in Y$ of system variables such that $\rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$ holds, provided $\vec{x}'$ corresponds to current round environment's move.

This is formalized by defining when a game state is a successor of another game state.

Definition 1.2 *A game state* $\langle \vec{x}', \vec{y}' \rangle$ *is a* successor *of* $\langle \vec{x}, \vec{y} \rangle$*, or, equivalently* $\langle \vec{x}, \vec{y} \rangle \longrightarrow \langle \vec{x}', \vec{y}' \rangle$*, if and only if both* $\rho_e(\vec{x}, \vec{y}, \vec{x}')$ *and* $\rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$ *hold.*

System goal is to keep the game going while maintaining φ satisfied, whereas environment goal is to lead the game to a state such that φ is not satisfied. Notice that the system can observe environment's move before performing its move.
Let us rephrase such intuition by means of formal notions.

Definition 1.3 *Given a* $\Box$*-GS* G *as above, a* play *of* G *is a maximal sequence of* G *states* $\eta = \langle \vec{x}_0, \vec{y}_0 \rangle \langle \vec{x}_1, \vec{y}_1 \rangle \cdots$ *such that:*

- $\langle \vec{x}_0, \vec{y}_0 \rangle \models \Theta$*, and*

- $\langle \vec{x}_j, \vec{y}_j \rangle \longrightarrow \langle \vec{x}_{j+1}, \vec{y}_{j+1} \rangle$*, for each* $j \geq 0$*.*

As it comes out from Definition 1.2, a play is, informally, a sequence of game states compliant with the "rules of the game". We remark that, in general, there are many, though finite, initial states, namely all those that satisfy Θ. The notion of play can be generalized to represent sequences of moves that start from any state:

Definition 1.4 *Given a $\Box$-GS G and a game state $\langle \vec{x}, \vec{y} \rangle \in V$, a $\langle \vec{x}, \vec{y} \rangle$-*play *of G is a maximal sequence of G states $\eta = \langle \vec{x}, \vec{y} \rangle \langle \vec{x}_1, \vec{y}_1 \rangle \cdots$ such that $\langle \vec{x}, \vec{y} \rangle \longrightarrow \langle \vec{x}_1, \vec{y}_1 \rangle$ and $\langle \vec{x}_j, \vec{y}_j \rangle \longrightarrow \langle \vec{x}_{j+1}, \vec{y}_{j+1} \rangle$, for $j \geq 1$.*

Finally, we define when a play is winning for the environment and for the system.

Definition 1.5 *Let G be a $\Box$-GS as above. A ($\langle \vec{x}, \vec{y} \rangle$-)play $\eta = \langle \vec{x}, \vec{y} \rangle \langle \vec{x}_1, \vec{y}_1 \rangle \cdots$ is said to be* winning for the system *if and only if:*

- *it is infinite, and*
- *satisfies the winning condition $\Box\varphi$, that is, $\langle \vec{x}, \vec{y} \rangle \models \varphi$ and $\langle \vec{x}_j, \vec{y}_j \rangle \models \varphi$, for each $j \geq 1$.*

Otherwise, it is winning for the environment*.*

A fundamental question about safety games is whether, given a game structure, the system has a winning *strategy*, that is, if, no matter how the environment plays, it is always able to keep the game going while satisfying φ. The following definitions provide a formalization of this notion.

Definition 1.6 *Let G be a $\Box$-GS as above. A* strategy *for the system is a partial function $f : (X \times Y)^+ \times X \to Y$ such that if $\lambda = \langle \vec{x}_0, \vec{y}_0 \rangle \cdots \langle \vec{x}_n, \vec{y}_n \rangle$ is a* finite sequence *of game states then for all $\vec{x}'$ such that $\rho_e(\vec{x}_n, \vec{y}_n, \vec{x}')$, $f(\lambda, \vec{x}')$ is defined and $\rho_s(\vec{x}_n, \vec{y}_n, \vec{x}', f(\lambda, \vec{x}'))$.*

Definition 1.7 *Let G be a $\Box$-GS and $f()$ a* strategy *for the system. A ($\langle \vec{x}_0, \vec{y}_0 \rangle$-)play $\eta = \langle \vec{x}_0, \vec{y}_0 \rangle \langle \vec{x}_1, \vec{y}_1 \rangle \cdots$ is said to be* compliant with $f()$ *if and only if, for all $i \geq 0$, $f(\langle \vec{x}_0, \vec{y}_0 \rangle \cdots \langle \vec{x}_i, \vec{y}_i \rangle, \vec{x}_{i+1}) = \vec{y}_{i+1}$.*

Definition 1.8 *Given a $\Box$-GS G as above, a (system) strategy $f()$ is* winning for the system *from state $\langle \vec{x}, \vec{y} \rangle$, if and only if all $\langle \vec{x}, \vec{y} \rangle$-plays compliant with $f()$ are so.*

If, from a state $\langle \vec{x}, \vec{y} \rangle$, there exists a strategy f winning for the system, then we expect that, from all possible successors $\langle \vec{x}', \vec{y}' \rangle$ obtained through $f()$, a winning strategy for the system exists. This observation suggests the following co-inductive definition:

Definition 1.9 *Let G be a $\Box$-GS as above. A set $\tilde{W} \subseteq V$ of game states is said to be* winning (for the system) *if and only if for each state $\langle \vec{x}, \vec{y} \rangle \in \tilde{W}$:*

1. $\langle \vec{x}, \vec{y} \rangle \models \varphi$;
2. *for each $\vec{x}' \in X$ such that $\rho_e(\vec{x}, \vec{y}, \vec{x}')$ there exists a $\vec{y}' \in Y$ such that:*
   - $\rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$;
   - $\langle \vec{x}', \vec{y}' \rangle \in \tilde{W}$.

Definition 1.10 *Given a $\Box$-GS G, a game state is* winning (for the system) *if and only if it is contained in some winning set.*

Let $W \subseteq V$ be the set of all winning states for a $\Box$-game G. Clearly, W is itself a winning set and, in particular, is the *largest* winning set of G.

Definition 1.11 *A $\Box$-GS is said to be* winning for the system *if all initial states are so. Otherwise, it is said to be* winning for the environment.

Next, we show how, given a generic $\Box$-GS, the set of all and only states that are winning for the system can be computed. To this end, we introduce the following operator (cf. [1, 5]):

Definition 1.12 *Let $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$ be a $\Box$-GS as above. Given a set $P \subseteq V$ of game states, the set of P*'s controllable predecessors *is*

$$Pre(P) \doteq \{\langle \vec{x}, \vec{y} \rangle \in V \mid \forall\, \vec{x}' \rho_e(\vec{x}, \vec{y}, \vec{x}') \rightarrow \exists\, \vec{y}' \rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}') \wedge \langle \vec{x}', \vec{y}' \rangle \in P\}$$

Intuitively, $Pre(P)$ is the set of states from which the system can force the game to reach a state in P, no matter how the environment evolves. Clearly, $Pre(Pre(P))$ is the set of states from which the system can force the game to reach a state in P after two moves, and so on for $Pre(Pre(\cdots Pre(P) \cdots))$: by iteratively applying operator Pre, one computes the set of game states from which the system can force the game to reach a state in P in a finite set of moves. This procedure is the intuition behind Algorithm 1 which, as Theorem 1.2 shows, computes the maximal set of system winning states for the system, in a given $\Box$-GS $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$.

Theorem 1.1 *Let $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$ be a $\Box$-GS as above. Algorithm 1 terminates in a finite number of steps and returns the maximal winning set W for G.*

The following fundamental result holds:

Theorem 1.2 *Let $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$ be a $\Box$-GS as above and W obtained as in Algorithm 1. Given a state $\langle \vec{x}_0, \vec{y}_0 \rangle \in V$, a winning strategy $f()$ for the system, starting from $\langle \vec{x}_0, \vec{y}_0 \rangle$, exists if and only if $\langle \vec{x}_0, \vec{y}_0 \rangle \in W$.*

Algorithm 1 MAX_WIN – Computes the maximal set of winning states for the system, in a $\Box$-GS

1: $W := \{\langle \vec{x}, \vec{y} \rangle \in V \mid \langle \vec{x}, \vec{y} \rangle \models \varphi\}$
2: **repeat**
3: $\quad W' := W$;
4: $\quad W := W \cap Pre(W)$;
5: **until** $(W' = W)$
6: **return** W

We report the *If Part* of the proof, which shows a procedure to come up with an actual strategy, given W.

Proof:
(If Part.) By Theorem 1.1, $W \subseteq Pre(W)$. So, by Definition 1.12: (†) for each $\langle \vec{x}, \vec{y} \rangle \in W$ and for each $\vec{x}' \in X$ such that $\rho_e(\vec{x}, \vec{y}, \vec{x}')$, there exists a $\vec{y}' \in Y$ such that $\rho_s(\vec{x}, \vec{y}, \vec{x}', \vec{y}')$ and $\langle \vec{x}', \vec{y}' \rangle \in W$. Moreover, by construction of W in Algorithm 1: (‡) for each $\langle \vec{x}, \vec{y} \rangle \in W$, $\langle \vec{x}, \vec{y} \rangle \models \varphi$.
Now, let $\Lambda^i \subseteq (X \times Y)^{i+1}$, for $i \geq 0$. We define, by induction, a function $f : (X \times Y)^+ \times X \longrightarrow Y$ and sets Λ^i as follows:

- $\Lambda^0 = \{\langle \vec{x}_0, \vec{y}_0 \rangle\}$. By hypothesis, $\langle \vec{x}_0, \vec{y}_0 \rangle \in W$;
- for each $\lambda \in \Lambda^\ell$, $\lambda = \langle x_0, y_0 \rangle \cdots \langle x_\ell, y_\ell \rangle$ $(\ell \geq 0)$, assume that $\langle x_\ell, y_\ell \rangle \in W$, then, for each $\vec{x}_{\ell+1} \in X$ such that $\rho_e(\vec{x}_\ell, \vec{y}_\ell, \vec{x}_{\ell+1})$, choose one $\vec{y}'$ among those as in (†) and let:
  - $\vec{y}_{\ell+1} = f(\lambda, \vec{x}_{\ell+1}) = \vec{y}'$;
  - $\lambda' = \langle x_0, y_0 \rangle \cdots \langle x_\ell, y_\ell \rangle \langle x_{\ell+1}, y_{\ell+1} \rangle \in \Lambda^{\ell+1}$.

  Since, by (†), $\langle x_{\ell+1}, y_{\ell+1} \rangle \in W$, function $f()$ is well founded.

Due to this definition, if a $\langle \vec{x}_0, \vec{y}_0 \rangle$-play $\eta = \langle \vec{x}_0, \vec{y}_0 \rangle \langle \vec{x}_1, \vec{y}_1 \rangle \cdots$ is compliant with $f()$ then it is infinite and $\langle \vec{x}_i, \vec{y}_i \rangle \in W$ for all $i \geq 0$, so, by (‡), $\langle \vec{x}_i, \vec{y}_i \rangle \models \varphi$. Therefore η is winning for the system. □

As a consequence of the theorem, we get the following result:

Theorem 1.3 *Let $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$ be a $\Box$-GS as above and W obtained as result of Algorithm 1. G is winning for the system if and only if all of its initial states belong to W.*

Proof: Direct consequence of Theorem 1.2. □

Hence, given a $\Box$-GS G, in order to check whether G is winning for the system and, if so, compute a winning strategy, one can adopt the following procedure:

- execute Algorithm 1 and compute W;
- check whether (†) all initial states are in W;
- if (†) holds then compute a winning strategy for the system by following the construction provided in *If Part* of Theorem 1.2;
- otherwise, notify that G is winning for the environment.

1.2 From Composition Problem to Safety Games

We introduce our approach for reducing a composition problem instance to a $\Box$-GS from a high-level perspective. Recall that the service composition problem consists in finding an orchestrator able to coordinate a community so that the obtained system can always satisfy the requests of a client compliant with a given deterministic target service.

In order to encode the composition problem as a safety-game structure, we need first to individuate which place each component, e.g., target, available services, data box, will occupy in the game representation. To this end, some remarks are worth making:

- given the target service, all and only legal client evolutions result in all possible target service executions;
- the target service is a virtual entity whose operations are to be actually executed by one available service, subject to its current state and capabilities. Therefore, when composing services, both the (virtual) target service and the (actual) community can be soundly thought as evolving synchronously, the latter executing exactly what the former is supposed to;
- our framework prescribes community services to be mutually asynchronous, i.e., *exactly* one moves at each step, yet synchronous with the data box;
- among all the entities involved in the setting, the only one that needs to be synthesized is the orchestrator: it is an automaton which, synchronously with both the target service and the community, outputs, depending on the current community state and the target service evolution that is being composed, an identifier used to delegate the requested operation to one available service.

Conceptually, our goal is to refine an *unconstrained* orchestrator –that is, an automaton capable of selecting, at each step, one among all the available

services– in a way such that the community is always able to satisfy target service requests [1]. This suggests to identify, in the game structure:

- the orchestrator with the *system*, which is the game player we want to synthesize a winning strategy for and, consequently,
- community services, target service and data box, properly combined, with the *environment*, of course guaranteeing that all synchronization requirements are met.

Finally, once the game players are described, a winning condition needs to be encoded. Recall that system's goal amounts to guaranteeing that the game goes on while keeping the winning condition verified. In our composition framework, we require that an orchestrator delegates operations, requested by the target service, so that

- if the target service is in a final state, all community services do, as well;
- the service selected by the orchestrator is able to perform the operation currently requested by the target service.

The winning condition will be formally encoded as conjunction of the above high-level properties, in addition to a third one needed to deal with a single (artificial) initial state, rather than many.

Now, we can show how to derive, given a composition problem instance, a $\Box$-GS.

1.2.1 Formal Reduction to Safety-Game Structure

Before providing technical details, let us address an important issue about TS modeling. As we saw, a game is winning for a system if all plays starting from an initial state are winning. To be such, in particular, a play needs to be infinite. In the reduction we propose next, the composition problem is encoded as a safety game, so that each winning strategy for the system corresponds to a composition and viceversa. To do so, we require that the target service has only infinite runs, otherwise finite plays are allowed and thus, as it will be clear soon, there might be loss of solutions. So, given a service composition problem instance, we preliminarily apply the following transformations, aimed at *extending* all finite runs that are possibly finite:

- if all target service states have at least one successor state (according to transition function), then nothing changes;

[1] Here and in the following, we blur the notions of client and target service as, like we said before, all client evolutions yield all target executions, so one can see a target execution as an alternative representation of some client's evolution.

- otherwise:
  - a special operation, say *nop*, not already present in $\mathcal{O}$, is added to $\mathcal{O}$;
  - for each state s with no successor, a (looping) transition $s \xrightarrow{\top,nop} s$ is added to the (functional) transition relation ϱ_t;
  - for all available services and the data box, a loop as above is added for each of their states.

Intuitively, we require that, after a finite run has been traversed, the target service can only decide to remain still, that each available service can be delegated, at any time, to execute such operation, and that no available service or the data box is affected by *nop*. In other words, we are explicitly representing the fact that the whole system remains still over time. From now on, we assume to deal only with infinite-run target services, possibly obtained by applying the above manipulation. Therefore, each community and target service state admit always a successor state (possibly itself).

Now, let $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ be a community and $\mathcal{S}_t$ a target service, where $\mathcal{S}_i = \langle \mathcal{O}, S_i, s_{i0}, \varrho_i, S_i^f \rangle$ $(i = t, 1 \ldots, n)$. From $\mathcal{C}$ and $\mathcal{S}_t$, we derive a $\Box$-GS $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_c, \Box\varphi \rangle$, as follows:

- $\mathcal{V} = \{s_t, s_1, \ldots, s_n, o, ind\}$, where:
  - s_i ranges over $\hat{S}_i \doteq S_i \cup \{init\}$ $(i = t, 1, \ldots, n)$;
  - o ranges over $\hat{\mathcal{O}} \doteq \mathcal{O} \cup \{init\}$;
  - ind ranges over $\{1, \ldots, n\} \cup \{init\}$;

  with an intuitive semantics: each complete valuation of $\mathcal{V}$ represents *(i)* the current state of community (variables $s_1, \ldots, s_n$,) and target service (variable s_t), *(ii)* the operation to be performed next (variable o) and *(iii)* the available service selected to perform it (variable ind). Special value $init$ has been introduced for convenience, so to have a single initial state;
- $\mathcal{X} = \{s_t, s_1, \ldots, s_n, o\}$ is the set of environment variables;
- $\mathcal{Y} = \{ind\}$ is the (singleton) set of system variables;
- $\Theta = (\bigwedge_{i=t,0,\ldots,n}(s_i = init)) \wedge (o = init) \wedge (ind = init)$, represents a *full* assignment to state variables, which identifies the initial state;
- Let $S_G = \hat{S}_t \times \hat{S}_1 \times \ldots \times \hat{S}_n \times \hat{\mathcal{O}}$ be the set of G's environment states. $\rho_e \subseteq S_G \times \{1, \ldots, n, init\} \times S_G$ is defined as follows:
  - $\langle \langle init, \ldots, init \rangle, init, \langle s_t, s_1, \ldots, s_n, o \rangle \rangle \in \rho_e$ if and only if:

1. $s_i = s_{i0}$, for $i = t, 1, \ldots, n$;
2. there exists a transition $s_{t0} \xrightarrow{o} s'_t$ in $\mathcal{S}_t$;

– if $s_i \neq init$, for $i = t, 1, \ldots, n$, $o \neq init$ and $ind \neq init$ then $\langle\langle s_t, s_1, \ldots, s_n, o\rangle, ind, \langle s'_t, s'_1, \ldots, s'_n, o'\rangle\rangle \in \rho_e$ if and only if the following holds:

1. there exists a transition $s_t \xrightarrow{o} s'_t$ in $\mathcal{S}_t$;
2. there exists a transition $s_{ind} \xrightarrow{o} s'_{ind}$ in $\mathcal{S}_{ind}$;
3. $s'_i = s_i$, for all $i = 1, \ldots, n$ such that $i \neq ind$;
4. there exists a transition $s'_t \xrightarrow{o'} s''_t$ in $\mathcal{S}_t$;

- $\rho_s \subseteq S_G \times \{1, \ldots, n, init\} \times S_G \times \{1, \ldots, n, init\}$ is defined as follows: $\langle\langle s_t, s_1, \ldots, s_n, o\rangle, ind, \langle s'_t, s'_1, \ldots, s'_n, o'\rangle, ind'\rangle \in \rho_s$ if and only if:
  – $\rho_e(\langle s_t, s_1, \ldots, s_n, o\rangle, ind, \langle s'_t, s'_1, \ldots, s'_n, o'\rangle)$, and
  – $ind' \in \{1, \ldots, n\}$;
- Formula φ is defined depending on current service states $s_t, s_1, \ldots, s_n$, operation o and service selection ind, as follows (for succinctness, variables that each term depends on are omitted, e.g., we write $fail_i$ instead of $fail_i(s_i, o, ind)$):

$$\varphi \doteq \Theta \vee (\bigwedge_{i=1}^{n} \neg fail_i) \wedge (final_t \rightarrow \bigwedge_{i=1}^{n} final_i),$$

where:

– Θ is defined as above;
– $fail_i \doteq (ind = i) \wedge (\bigwedge_{\langle s, op, s'\rangle \in \varrho_i} (s_i \neq s \vee o \neq op))$, encodes the fact that service i has been selected but, in its current state, no transition can take place which executes the requested operation;
– $final_i \doteq \bigvee_{s \in S_i^f} (s_i = s)$ encodes the fact that service $i = t, 1, \ldots, n$ is currently in one of its final states.

Observe how, since game structures do not allow for transition labeling, the operation requested by a client appears as a state variable, in the game representation. This transformation is similar, in the spirit, to the procedure usually adopted to transform a Mealy into a Moore machine. As for the winning condition, it is an invariant property that needs to be always satisfied, in order for the system to win or, rephrased from service composition viewpoint, in order for an orchestrator to be a composition. It should be intuitively clear that each game play essentially reproduces a synchronous execution of the target service and the community, when sequences of operations compliant with target service specification are executed. Analogously,

the converse holds. Such intuitions will be formally addressed later on in this chapter.

The basic idea behind the reduction to a safety-game structure is that, once the service composition problem is encoded as a $\Box$-GS, from the maximal winning set, computed as in Algorithm 1, one can extract a NCG, thus a composition which solves the original problem.

1.2.2 Technical Results

We can now show how the obtained game structure allows for computing a composition generator. Recall that, in order to define the NCG, one needs to build an ND-simulation. The following Theorem shows that this can be equivalently done by computing the maximal set of winning states for the system.

Theorem 1.4 *Let $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ be a community and $\mathcal{S}_t$ a target service where, as usual, $\mathcal{S}_i = \langle \mathcal{O}, S_i, s_{i0}, \varrho_i, S_i^f \rangle$ $(i = 1 \ldots, n, t)$. From $\mathcal{C}$ and $\mathcal{S}_t$ derive a $\Box$-GS $G = \langle V, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$, as shown above, and let $W \subseteq V$ be the maximal set of winning states for the system. Then, $\langle init, \ldots, init \rangle \in W$ if and only if $s_{t0} \preceq s_{\mathcal{C}0}$.*

Based upon this result, the following theorem provides an actual procedure to build an ND composition generator and, from this, all possible compositions.

Theorem 1.5 *Let $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, $\mathcal{S}_t$ and $G = \langle V, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \Box\varphi \rangle$ be as in Theorem 1.4 above. In addition, let W be the winning set for the system. If $\langle init, \ldots, init \rangle \in W$, then the ND composition generator $NCG = \langle \mathcal{O}, \{1, \ldots, n\}, S_g, s_{g0}, \omega_g, \delta_g \rangle$ of $\mathcal{C}$ for $\mathcal{S}_t$ can be built from W, as follows:*

- *$\mathcal{O}$ is the usual set of operations and $\{1, \ldots, n\}$ the set of available services' indexes;*
- *$S_g \subseteq S_t \times S_{\mathcal{C}}$ is such that $\langle s_t, \langle s_1, \ldots, s_n \rangle \rangle \in S_g$ if and only if there exists a game state $\langle s_t, s_1, \ldots, s_n, o, ind \rangle \in W$, for some $o \in \mathcal{O}$ and $ind \in \{1, \ldots, n\}$;*
- *$s_{g0} = \langle s_{t0}, \langle s_{10}, \ldots, s_{n0} \rangle \rangle$ is NCG's* initial state*;*
- *$\omega_g : S_g \times \mathcal{O} \longrightarrow 2^{\{1,\ldots,n\}}$ is defined as $\omega_g(\langle s_t, \langle s_1, \ldots, s_n \rangle \rangle, o) = \{i \in \{1, \ldots, n\} \mid \langle s_t, s_1, \ldots, s_n, o, i \rangle \in W\}$;*
- *$\delta_g \subseteq (S_g \times \mathcal{O} \times \{1, \ldots, n\} \times S_g)$ is such that $\langle \langle s_t, \langle s_1, \ldots, s_n \rangle \rangle, o, k, \langle s'_t, \langle s'_1, \ldots, s'_n \rangle \rangle \rangle \in \delta_g$ if and only if $\langle s_t, s_1, \ldots, s_n, o, k \rangle \in W$ and there exist $o' \in \mathcal{O}$ and $k' \in \{1, \ldots, n\}$ such that $\langle \langle s_1, \ldots, s_n, s_t, o \rangle, k, \langle s'_1, \ldots, s'_n, s'_t, o' \rangle, k' \rangle \in \rho_s$.*

Above theorems show how one can exploit tools from system synthesis for computing all compositions of a given target service. In details, starting from $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ and $\mathcal{S}_t$, one can build the corresponding game structure G as shown above, then compute the set W through the use of some available tool and, if it contains G's initial state, use such set to generate the NCG. In fact, this last step is not really needed. Indeed, it is not hard to convince oneself that given a current state $\langle s_t, s_1, \ldots, s_n \rangle$ and an operation to be executed $o \in \mathcal{O}$ a service selection ind is "good" (i.e, the selected service can actually execute the operation and the whole community can still simulate the target service) if and only if W contains a tuple $\langle s_t, s_1, \ldots, s_n, o, ind \rangle$, for some $ind \in \{1, \ldots, n\}$. Consequently, at each step, on the basis of the current state s_t of the target service, the states $s_1, \ldots, s_n$ of available services and the requested operation o requested, one can select a tuple from W, extract the ind component, and use this for next service selection.

Finally, observe that time complexity of Algorithm 1 is polynomial in $|V|$, that is the size of input $\Box$-GS' state space. Since in our encoding $|V|$ is polynomial in $|S_1|, \ldots, |S_n|, |S_t|$ and exponential in n, we get the following result.

Theorem 1.6 *Let $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ be a community and $\mathcal{S}_t$ a target service over $\mathcal{DB}$. Checking the existence of compositions by reduction to safety games can be done in polynomial time with respect to $|S_1|, \ldots, |S_n|, |S_t|$ and exponential time with respect to n.*

That is, the technique is actually optimal wrt worst-case time complexity, the composition problem being EXPTIME-hard [3].

References

[1] Asarin, E., Maler, O., and Pnueli, A. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II* (1995), Springer-Verlag, pp. 1–20.

[2] Asarin, E., Maler, O., Pnueli, A., and Sifakis, J. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control* (1998), Elsevier, pp. 469–474.

[3] Muscholl, A., and Walukiewicz, I. A lower bound on web services composition. In *Proc. of the 10th Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS 2007)* (2007), vol. 4423 of *LNCS*, Springer.

[4] Patrizi, F. *Simulation-based Techniques for Automated Service Composition.* PhD thesis, Sapienza Università di Roma, 2008.

[5] Piterman, N., Pnueli, A., and Sa'ar, Y. Synthesis of reactive(1) designs. In *VMCAI* (2006), pp. 364–380.