Innovations in Systems and Software Engineering manuscript No.
(will be inserted by the editor)

Timed Service Contract Automata

Davide Basile · Maurice H. ter Beek · Axel Legay

Received: date / Accepted: date

Abstract We equip a recently developed model for the specification of service contracts with real-time constraints. Service contracts offer a means to define the behavioural compliance of a composition of services, typically dictated in a Service-Level Agreement (SLA), as the fulfilment of all service requests through service offers. Depending on their granularity, SLAs vary according to the level of criticality of the involved services and also contain real-time aspects, like the services' response or expiration time. A standard method to refine a spurious service composition into a compliant one is via the synthesis of a safe orchestration, in the form of the most permissive controller from Supervisory Control Theory (SCT). Ideally, safe orchestrations solve competition among matching service requests and offers, in light of their criticalities and their timing constraints, in the best possible way. In this paper, we introduce timed service contract automata as a novel formal model for service contracts with real-time constraints on top of services with varying levels of criticality. We also define a means to efficiently compute their composition and their safe orchestration, using the concept of zones from timed games. The innovations of our contribution are illustrated by intuitive examples and by a preliminary evaluation.

Keywords Services · Contracts · Automata · Real-Time · Synthesis · Orchestration · Games

D. Basile
Università degli Studi di Firenze, Italy
E-mail: davide.basile@unifi.it

M.H. ter Beek
ISTI–CNR, Pisa, Italy
E-mail: maurice.terbeek@isti.cnr.it

A. Legay
Université Catholique de Louvain, Belgium
E-mail: axel.legay@uclouvain.be

1 Introduction

Service computing is an emerging discipline concerned with the creation, publication, discovery and orchestration of services [24, 32]. A typical application is an orchestration of services that are created and published by different organisations and that are dynamically discovered. In the recently published Service Computing Manifesto [18], *service design* is listed as one of the four emerging research challenges in service computing for the next 10 years. The authors of [18] signal that "Service systems have so far been built without an adequate rigorous foundation that would enable reasoning about them" and claim that "The design of service systems should build upon a formal model of services". This paper is part of an encompassing effort to address these issues [7, 11, 13].

Several formal models of service contracts are surveyed in [14, 5]. These concern specification frameworks to formalise the externally observable behaviour of services in terms of obligations (i.e. *service offers*) and requirements (i.e. *service requests*) to be matched. Contracts that are fulfilled characterise *agreement* among services as an *orchestration* (i.e. a composition) of them based on the satisfaction of all requirements through obligations. Orchestrations must be able to dynamically adapt to the discovery of new services, to service updates and to services that are no longer available [6].

In this paper, we include notions of time in one particular such formal model for service contracts, with a precise semantics, namely the (service) contract automata from [7][1]. A service contract automaton represents either a single service (called a principal) or a multi-party composition of services. The goal of each principal is to reach an accepting (final) state by matching its requests with corresponding of-

[1] Our (service) contract automata are not to be confused with the homonym contract automata of [4], cf. the related work discussed below.

fers of other principals. The underlying composition mechanism is orchestration. Service interactions are implicitly controlled by an orchestrator synthesised from the principals, which directs them in such a way that only finite executions in agreement actually happen. Through service contracts it is then possible to characterise the behaviour of an ensemble of services. The (verifiable) notion of *agreement* characterises safe executions of services (i.e. all requests are matched by corresponding offers).

Our formalism also builds upon [11], where service contract automata are equipped with modalities that distinguish between *necessary* and *permitted* requests to mimic the uncontrollable and controllable actions, respectively, as known from Supervisory Control Theory (SCT) [31, 20]. In the resulting (modal) service contract automata, only service requests can be classified as necessary; instead, all service offers are by definition permitted, based on the assumption that a service contract may always withdraw its offers that are not needed to reach agreement. A synthesis algorithm was defined to orchestrate services in agreement by guaranteeing the satisfaction of all necessary requests and negotiating the maximum number of permitted requests that can be satisfied without leading to a spurious service composition. In this case, contracts thus adapt to the agreement by renouncing to permitted but unsatisfiable requirements.

Contribution In this paper, we introduce *timed service contract automata* (TSCA) by endowing service contract automata with *real-time constraints*, which must be fulfilled in successful orchestrations. TSCA also allow to specify different types of necessary requests, called *urgent*, *greedy* and *lazy*, with decreasing levels of criticality borrowed from [9]. These different criticality levels are key aspects to ensure that certain necessary requests must *always* be satisfied (e.g. in each possible context) while others must *eventually* be satisfied (e.g. in specific contexts). To deal with such actions requests in a synthesis algorithm for TSCA, a novel notion of *semi-controllability* is used, which encompasses both the notion of controllability and that of uncontrollability as used in classical synthesis algorithms from SCT [31, 20]. Our synthesis algorithm mixes and extends techniques from SCT with notions from timed games [3, 21].

A TSCA orchestration thus solves multi-party competitions on service actions and as well as on the associated timing constraints, which is a natural scenario in service computing, and provides a winning strategy by which contract requirements are matched by available offers. Moreover, TSCA offer a lot of flexibility in the design of service systems through different levels of critical requests and, in particular, by allowing to indicate those service offers and requests that can possibly be (temporarily) ignored in an orchestration to avoid it to become unsafe. This neatly delimits those fragments (i.e. executions) of service compositions that are allowed in safe orchestrations (cf. Fig. 8 discussed in Sect. 4). By changing the timing constraints or criticality levels, designers can fine-tune such fragments according to their specific needs.

Our contribution can thus be summarised as follows: (i) we introduce TSCA, a novel formal model for service contracts equipped with (ii) real-time constraints and with (iii) different types of service requests of varying levels of criticality, as well as with a means to efficiently compute (iv) composition and (v) safe orchestration of TSCA; we (vi) illustrate the functioning of our formalism with a TSCA model of a Hotel service reservation system and, finally, we (vii) provide a preliminary evaluation of our approach to show the advantages of native support for the above features. We are not aware of any other formalism for service contracts or from component-based software engineering with native support for these features. In fact, the preliminary evaluation compares the innovations of our approach with respect to the model of timed I/O automata [22], but the comparison remains valid with respect to related formalisms, like Interface Automata [1].

Related Work Several foundational formalisms for service contracts and session types are surveyed in [25]. Our model is fundamentally different, since all of the surveyed models lack an explicit notion of time and none of them considers different levels of criticality of services.

Well-known component-based formalisms from software engineering, such as Interface automata [1] and (timed) (I/O) automata [29, 2, 22] cannot model contracts that compete for the same service offer or request, a key feature of TSCA, and also do not cater for modelling services with different levels of criticality (cf. the comparison in Section 5).

Modal transition systems and their extensions [26], as adopted for instance in software product line engineering, like modal I/O automata [27] and modal transition systems with variability constraints [15], do of course distinguish may and must modalities, thus admitting some actions to be more critical than others, but the other aforementioned differences with TSCA remain.

The accidentally homonym contract automata of [4] are meant to formalise generic natural language legal contracts among two parties. They are finite state automata whose states are tagged with deontic modalities in the form of obligations and permissions. A contract is satisfied if all deontic modalities are honoured, and violated otherwise. TSCA target a different domain, viz. multi-party service contracts. Necessary and permitted requirements do resemble such obligations and permissions, but the latter are defined on states while the former are defined on actions. Moreover, the contract automata of this paper distinguish different types of necessary requirements that stem from the multi-party composition not addressed in [4]. Finally, contrary to TSCA, the

contract automata of [4] are not compositional and also do not consider synthesising orchestrations of services in agreement.

To conclude, our synthesis algorithm for TSCA (cf. Section 3) resembles a timed game, but it differs from classical timed game algorithms [3, 21, 22] in the following aspects. The synthesis algorithm for TSCA solves both reachability and safety problems, and a TSCA might be such that all 'bad' configurations are unreachable (i.e. it is safe), while at the same time no final configuration is reachable (i.e. the resulting orchestration is empty). TSCA strategies are defined as relations: the orchestration being the maximal winning strategy, which is computable since only finite traces are allowed [19] and all services terminate by definition. The orchestrator enforces only fair executions. More detailed technical differences are given in Section 3.

Outline Section 2 formalises TSCA, including their semantics, composition, and the controllability of (service) actions. Section 3 presents a synthesis algorithm for computing the safe orchestration of TSCA via zone operators. Section 4 contains examples of the applicability of TSCA. Section 5 highlights the innovations of this paper by means of a preliminary evaluation of TSCA. Section 6 concludes the paper.

This paper is a revised version of a conference publication at VECoS [13], extended with detailed proofs of our results and with a preliminary evaluation of our approach. Moreover, based on suggestions coming from the audience at VECoS, we slightly revisited the formalism by incorporating state invariants for modelling controllability over time.

2 Modelling Real-time Service Contracts

Contract automata were introduced to provide a rigorous formal technique for describing and composing to describe and compose service contracts [7]. A contract automaton represents the behaviour of a set of principals (possibly a singleton) which can either request, offer or match services (a match is a pair of complementary request-offer services) or remain idle. The number of principals in a contract automaton is called its rank. The states and actions labelling the transitions of a contract automaton (of rank n) are vectors (of rank n) over the states of its principals and over the actions that each performs, respectively.

Notation We first introduce some notation. Given a finite set S, we denote its complement by $\overline{S}$, while $\varnothing$ denotes the empty set. A vector $\mathbf{v} = (e_1, \ldots, e_n)$ of *rank* $n \geq 1$ is denoted by r_v. We let $\mathbf{v}_{(i)}$ denote its ith element, with $1 \leq i \leq r_v$. We denote the concatenation of m vectors $\mathbf{v}_i$ by $\mathbf{v}_1 \cdots \mathbf{v}_m$.

The set of basic actions of a contract automaton is defined as $\Sigma = \mathsf{R} \cup \mathsf{O} \cup \{\bullet\}$, where $\mathsf{R} = \{a, b, \ldots\}$ is the set of *requests*, $\mathsf{O} = \{\overline{a}, \overline{b}, \ldots\}$ is the set of *offers*, $\mathsf{R} \cap \mathsf{O} = \varnothing$, and $\bullet \notin \mathsf{R} \cup \mathsf{O}$ is a distinguished element representing an *idle* move. We define the involution $co(\cdot) : \Sigma \mapsto \Sigma$ s.t. $co(\mathsf{R}) = \mathsf{O}$, $co(\mathsf{O}) = \mathsf{R}$ and $co(\bullet) = \bullet$.

We stipulate that in an action vector $\mathbf{a}$ over Σ there is either a single offer or a single request, or a single pair of request-offer that matches, i.e. there exist i, j such that $\mathbf{a}_{(i)}$ is an offer and $\mathbf{a}_{(j)}$ is the complementary request, or vice versa; all the other elements of $\mathbf{a}$ contain the symbol $\bullet$ (meaning that the corresponding principals remain idle). We let $\bullet^m$ denote a vector $(\bullet, \ldots, \bullet)$ of rank m.

Table 1: Classification of (basic) actions of TSCA

permitted offers	permitted requests	necessary requests		
		lazy	*greedy*	*urgent*
$\overline{a}$	$a\Diamond$	$a\Box_\ell$	$a\Box_g$	$a\Box_u$

Definition 1 (Actions) Let $\mathbf{a}$ be an action vector over Σ. Let $n_1, n_2, n_3 \geq 0$.

If $\mathbf{a} = \bullet^{n_1} \alpha \bullet^{n_2}$, then $\mathbf{a}$ is a *request (action) on* α if $\alpha \in \mathsf{R}$, whereas $\mathbf{a}$ is an *offer (action) on* α if $\alpha \in \mathsf{O}$.

If $\mathbf{a} = \bullet^{n_1} \alpha \bullet^{n_2} co(\alpha) \bullet^{n_3}$, then $\mathbf{a}$ is a *match (action) on* α, with $\alpha \in \mathsf{R} \cup \mathsf{O}$.

Actions $\mathbf{a}$ and $\mathbf{b}$ are *complementary*, denoted by $\mathbf{a} \bowtie \mathbf{b}$, iff the following holds: (i) $\exists\, \alpha \in \mathsf{R} \cup \mathsf{O}$ s.t. $\mathbf{a}$ is either a request or an offer on α; (ii) $\mathbf{a}$ is an offer on α implies that $\mathbf{b}$ is a request on $co(\alpha)$; (iii) $\mathbf{a}$ is a request on α implies that $\mathbf{b}$ is an offer on $co(\alpha)$.

In [11], the contract automata of [7] were equipped with a notion of 'action' variability by means of *necessary* ($\Box$) and *permitted* ($\Diamond$) modalities that can be used to classify requests (and matches), whereas all offers are by definition classified as permitted. Permitted requests and offers reflect optional behaviour and as such they can thus be discarded in compositions of contract automata.

2.1 Timed Service Contract Automata

In this paper, the set of necessary requests of the service contract automata of [11] is partitioned into *urgent*, *greedy*, and *lazy* requests as borrowed from [9]. Since these requests must be matched to reach an agreement among contracts, they add a layer of 'timed' variability to the formalism: a means to specify 'when' certain (service) requests must be matched in a composition (contract). Table 1 classifies the different types of actions that are considered in this paper.

Notation We borrow some notation concerning clocks from [21]. Let X be a finite set of real-valued variables, called clocks. Let $C(X)$ denote the set of constraints φ generated by the grammar $\varphi ::= x \sim k \mid x - y \sim k \mid \varphi \wedge \varphi$, where $k \in \mathbb{Z}$,

$x, y \in X$, and $\sim \in \{<, \leq, =, >, \geq\}$. Let $B(X)$ denote the subset of $C(X)$ that uses only rectangular constraints of the form $x \sim k$. For simplicity, we consider only such constraints. A *valuation* of the variables in X is a mapping $X \mapsto \mathbb{R}_{\geq 0}$. Let $\mathbf{0}$ denote the valuation that assigns 0 to each clock. For $Y \subseteq X$, we denote by $v[Y]$ the valuation assigning 0 for any $x \in Y$ and $v(x)$ for any $x \in X \setminus Y$. Let $v + \delta$ for $\delta \in \mathbb{R}_{\geq 0}$ denote the valuation s.t. for all $x \in X$, $(v+\delta)(x) = v(x) + \delta$. For $g \in C(X)$ and $v \in \mathbb{R}^X_{\geq 0}$, we write $v \models g$ if v satisfies g and $[g]$ denotes the set of valuations $\{v \in \mathbb{R}^X_{\geq 0} \mid v \models g\}$. A *zone* Z is a subset of $\mathbb{R}^X_{\geq 0}$ s.t. $[g] = Z$ for some $g \in C(X)$.

Definition 2 (TSCA) A *timed service contract automaton* (TSCA) of rank $n \geq 1$ is a tuple

$$\langle Q, \mathbf{q_0}, A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell}, A^o, X, T, I, F \rangle$$

in which

- $Q = Q_1 \times \cdots \times Q_n$ is the product of finite sets of states
- $\mathbf{q_0} \in Q$ is the initial state
- $A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell} \subseteq \mathsf{R}$ are (pairwise disjoint) sets of permitted, urgent, greedy, and lazy requests, respectively, and we denote the set of requests by $A^r = A^{\Diamond} \cup A^{\Box}$, where $A^{\Box} = A^{\Box_u} \cup A^{\Box_g} \cup A^{\Box_\ell}$
- $A^o \subseteq \mathsf{O}$ is the finite set of offers
- X is a finite set of real-valued clocks
- $T \subseteq Q \times B(X) \times A \times 2^X \times Q$, where $A = (A^r \cup A^o \cup \{\bullet\})^n$, is the set of transitions partitioned into *permitted* transitions $T^{\Diamond}$ and *necessary* transitions $T^{\Box}$ with $T = T^{\Diamond} \cup T^{\Box}$ s.t., given $t = (\mathbf{q}, g, \mathbf{a}, Y, \mathbf{q}') \in T$, the following holds:
  - $\mathbf{a}$ is either a request, an offer, or a match
  - $\forall i \in 1 \ldots n : \mathbf{a}_{(i)} = \bullet$ implies $\mathbf{q}_{(i)} = \mathbf{q}'_{(i)}$
  - $t \in T^{\Diamond}$ iff $\mathbf{a}$ is either a request or a match on $a \in A^{\Diamond}$ or an offer on $\overline{a} \in A^o$; otherwise $t \in T^{\Box}$
- $I : Q \mapsto B(X)$ is a function assigning an *invariant* to each state
- $F \subseteq Q$ is the set of final states

A *principal* TSCA (or just *principal*) has rank 1 and it is such that $A^r \cap co(A^o) = \varnothing$.

Unless stated differently, we will assume a fixed TSCA $\mathscr{A} = \langle Q_{\mathscr{A}}, \mathbf{q_0}_{\mathscr{A}}, A^{\Diamond}_{\mathscr{A}}, A^{\Box_u}_{\mathscr{A}}, A^{\Box_g}_{\mathscr{A}}, A^{\Box_\ell}_{\mathscr{A}}, A^o_{\mathscr{A}}, X_{\mathscr{A}}, T_{\mathscr{A}}, I_{\mathscr{A}}, F_{\mathscr{A}} \rangle$ of rank n in the sequel. Subscript $\mathscr{A}$ may be omitted when no confusion can arise. Moreover, if not stated otherwise, each operation $op(A^r)$ (e.g. union) is intended to be performed homomorphically on $op(A^{\Diamond})$, $op(A^{\Box})$, $op(A^{\Box_u})$, $op(A^{\Box_g})$, and $op(A^{\Box_\ell})$. Finally, abusing notation, we may write $T^{\Diamond\cup\Box}$ as shorthand for $T^{\Diamond} \cup T^{\Box}$ and likewise for other transition sets, and we may write a transition t as a request, an offer, or a match, whenever its label is such.

Pictorially, offer actions are overlined while request actions are not. Furthermore, permitted actions label dotted transitions and are suffixed by $\Diamond$, whereas urgent, greedy, and lazy necessary actions label solid transitions (red, orange, and green in the pdf) suffixed by $\Box_u$, $\Box_g$, and $\Box_\ell$, respectively (cf. Table 1).[2] Finally, a state's invariant is written in bold and omitted in case it is true (cf. Fig 1).

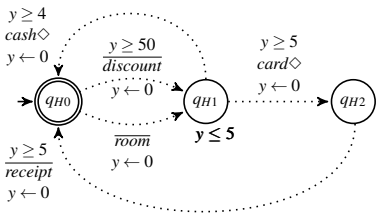

Fig. 1: TSCA model Hotel

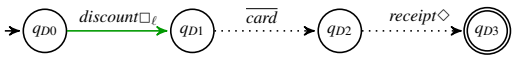

Fig. 2: TSCA model DiscountClient

Example 1 Figures 1 and 2 depict two TSCA models. The one in Fig. 1 specifies a hotel booking system that offers two room types (a normal *room* and a *discount* room). The discount room is only available upon waiting at least 50 time units (t.u. for short). The hotel requests payment from clients, either in *cash* (which takes at least 4 t.u.) or by *card* (at least 5 t.u.). Only in the latter case, the hotel also offers a *receipt* after at least 5 t.u. Note, moreover, that the presence of the invariant $\mathbf{y \leq 5}$ in state q_{H1} effectively requires the payment by *card* to occur (instantaneously) precisely when $y = 5$ holds (for the sake of example).

The TSCA in Fig. 2 depicts a hotel client, who requests (lazily) a *discount* room, after which she offers to pay by *card* and requests a *receipt*. Note that the model has no real-time constraints and all its state invariants are true.

2.2 Semantics

A TSCA recognises a trace language over actions and their modalities. Let $\mathscr{A}$ be a TSCA and let $\bigcirc \in \{\Diamond, \Box_u, \Box_g, \Box_\ell\}$. From now on we use $\bigcirc$ as placeholder for necessary ($\Box$) and permitted ($\Diamond$) transitions. A *configuration* of a TSCA is a triple $(w, \mathbf{q}, v) \in (A \cup \{\bigcirc\})^* \times Q \times \mathbb{R}^X_{\geq 0}$ consisting of a recognised trace w, a state $\mathbf{q}$, and a valuation v of clocks. Recognised traces are such that from a configuration $(w, \mathbf{q}, v)$, a TSCA either lets time progress or performs a discrete step to reach a new configuration. This is formally defined by the transition relation $\rightarrow$ by which a step $(w, \mathbf{q}, v) \xrightarrow{\mathbf{a}\bigcirc} (w', \mathbf{q}', v')$ is executed iff $w = \mathbf{a}\bigcirc w'$ and $(\mathbf{q}, g, \mathbf{a}, Y, \mathbf{q}') \in T^{\bigcirc}$, in which either $v \models g$, $v' = v[Y]$, and $v' \models I(\mathbf{q}')$, or else, for some $\delta \geq$

[2] In this paper, there are no examples of greedy necessary actions.

0, we have $(w,\mathbf{q},v)\xrightarrow{\delta}(w,\mathbf{q},v')$ if $v'=v+\delta$ and $v'\models I(\mathbf{q})$. We let time progress δ be a silent action in the languages recognised by TSCA.

The semantics of a TSCA $\mathscr{A}$ is a labelled transition system $TS_{\mathscr{A}}=(\mathbb{C},c_0,\rightarrow)$, where $\mathbb{C}=(A\cup\{\bigcirc\})^*\times Q\times\mathbb{R}^X_{\geq 0}$ is the set of configurations, $c_0=(w,\mathbf{q}_0,\mathbf{0})\in\mathbb{C}$ is the initial configuration, for some $w\in(A\cup\{\bigcirc\})^*$, and $\rightarrow$ is the transition relation with set of transition labels $(A\{\bigcirc\})\cup\mathbb{R}_{\geq 0}$. A *run* of $\mathscr{A}$ is a sequence of alternating time and discrete transitions in TS_A. Note that the traces recognised by TSCA languages are finite.

By an abuse of notation, the modalities can be attached to basic actions or to their action vector (e.g. $(a\Box_\ell,\overline{a})\equiv(a,\overline{a})\Box_\ell$). Configurations may be written as $(\mathbf{q},v)$ whenever w is immaterial, discrete steps as $(\mathbf{q},v)\xrightarrow{\mathbf{a}\bigcirc}$ whenever $(\mathbf{q}',v')$ is immaterial, and (time or discrete) transitions as $(w,\mathbf{q},v)\rightarrow(w',\mathbf{q}',v')$ whenever $\mathbf{a}\bigcirc$ or δ are immaterial. Let $\rightarrow^*$ denote the reflexive and transitive closure of $\rightarrow$. The language of $\mathscr{A}$ is defined as

$$\mathscr{L}(\mathscr{A})=\{\,w\mid(w,\mathbf{q_0},\mathbf{0})\rightarrow^*(\varepsilon,\mathbf{q},v),\ \mathbf{q}\in F\,\}$$

Behavioural analysis of TSCA is based on exploring a (finite) *simulation graph*, whose nodes are *symbolic configurations*, defined as pairs $(\mathbf{q},Z)$, where $\mathbf{q}\in Q$ and Z is a zone of $\mathbb{R}^X_{\geq 0}$. Let $C\subseteq\mathbb{C}$ be a set of configurations and let $\mathbf{a}\in A$. We define the $\mathbf{a}$-successor of X as

$$Post_{\mathscr{A},\mathbf{a}}(C)=\{\,c'\mid\exists c\in C: c\xrightarrow{\mathbf{a}\bigcirc}c'\,\}$$

and the $\mathbf{a}$-predecessor as

$$Pred_{\mathscr{A},\mathbf{a}}(C)=\{\,c\mid\exists c'\in C: c\xrightarrow{\mathbf{a}\bigcirc}c'\,\}$$

We moreover define the *match/offer predecessor* as

$$moPred_{\mathscr{A}}(C)=\bigcup_{\mathbf{a}\ \textit{match or offer}}Pred_{\mathscr{A},\mathbf{a}}(C)$$

The timed successors and predecessors of C are defined by

$$C^{\nearrow}=\{\,(\mathbf{q},v+\delta)\mid(\mathbf{q},v)\in C,\ v\wedge v+\delta\models I(\mathbf{q}),\ \delta\in\mathbb{R}_{\geq 0}\,\}$$

and

$$C^{\swarrow}=\{\,(\mathbf{q},v-\delta)\mid(\mathbf{q},v)\in C,\ v\wedge v-\delta\models I(\mathbf{q}),\ \delta\in\mathbb{R}_{\geq 0}\,\}$$

respectively. We define $\rightarrow$ to be the transition relation defined over symbolic configurations by $(\mathbf{q},Z)\xrightarrow{\mathbf{a}\bigcirc}(\mathbf{q}',Z')$ if $(\mathbf{q},g,\mathbf{a},Y,\mathbf{q}')\in T^{\bigcirc}$ and $Z'=((Z\cap[g])[Y])^{\nearrow}$.

2.3 Composition

A set of TSCA is *composable* iff their sets of clocks are pairwise disjoint.

Definition 3 (Composable) A set $\{\,\mathscr{A}_i\mid i\in 1\ldots n\,\}$ of TSCA is said to be *composable* iff $\forall X_i,X_j,i\neq j: X_i\cap X_j=\varnothing$.

The operands of the composition operator $\otimes$ are either principals or composite services. Intuitively, a composition interleaves the actions of all operands, with only one restriction: if two operands are ready to execute two complementary actions (i.e. $\mathbf{a}_i\bowtie\mathbf{a}_j$), then only their match is allowed whilst their interleaving is prevented. The formal definition precedes an intuitive explanation. Recall from Definition 2 that the set of actions is $A\subseteq(A^r\cup A^o\cup\{\bullet\})^m$. Also recall that we set $\bigcirc\in\{\Diamond,\Box\}$.

Definition 4 (Composition) Let $\mathscr{A}_i$ be a composable TSCA of rank r_i, with $i\in 1\ldots n$. The *composition* $\bigotimes_{i\in 1\ldots n}\mathscr{A}_i$ is the TSCA $\mathscr{A}$ of rank $m=\sum_{i\in 1\ldots n}r_i$, in which

- $Q=Q_1\times\cdots\times Q_n$, with $\mathbf{q_0}=\mathbf{q}_{01}\cdots\mathbf{q}_{0n}$
- $A^r=\bigcup_{i\in 1\ldots n}A^r_i$
- $A^o=\bigcup_{i\in 1\ldots n}A^o_i$
- $X=\bigcup_{i\in 1\ldots n}X_i$
- $T^{\bigcirc}\subseteq Q\times B(X)\times A\times 2^X\times Q$ s.t. $(\mathbf{q},g,\mathbf{a},Y,\mathbf{q}')\in T^{\bigcirc}$ iff, when $\mathbf{q}=\mathbf{q}_1\cdots\mathbf{q}_n\in Q$, either case (1) or case (2) holds:
  1. $\exists i,j,1\leq i<j\leq n$, s.t. $(\mathbf{q}_i,g_i,\mathbf{a}_i,Y_i,\mathbf{q}'_i)\in T_i^{\bigcirc}$, $(\mathbf{q}_j,g_j,\mathbf{a}_j,Y_j,\mathbf{q}'_j)\in T_j^{\bigcirc\cup\Diamond}$, $\mathbf{a}_i\bowtie\mathbf{a}_j$ holds, and
  $$\begin{cases}\mathbf{a}=\bullet^u\mathbf{a}_i\bullet^v\mathbf{a}_j\bullet^z,\text{with } u=r_1+\cdots+r_{i-1},\\ v=r_{i+1}+\cdots+r_{j-1},\ z=r_{j+1}+\cdots+r_n,\\ |\mathbf{a}|=m,\ g=g_i\wedge g_j,\ Y=Y_i\cup Y_j,\ \text{and}\\ \mathbf{q}'=\mathbf{q}_1\cdots\mathbf{q}_{i-1}\,\mathbf{q}'_i\,\mathbf{q}_{i+1}\cdots\mathbf{q}_{j-1}\,\mathbf{q}'_j\,\mathbf{q}_{j+1}\cdots\mathbf{q}_n\end{cases}$$
  or
  $$\begin{cases}k,k'\in\{i,j\},\ k\neq k',\ g=g_k\wedge\neg g_{k'},\ Y=Y_k,\\ \mathbf{a}=\bullet^u\mathbf{a}_k\bullet^v,\text{with } u=r_1+\cdots+r_{k-1},\\ v=r_{k+1}+\cdots+r_n,\ |\mathbf{a}|=m,\ \text{and}\\ \mathbf{q}'=\mathbf{q}_1\cdots\mathbf{q}_{i-1}\,\mathbf{q}'_i\,\mathbf{q}_{i+1}\cdots\mathbf{q}_n\end{cases}$$
  2. $\exists i,1\leq i\leq n$, s.t. $(\mathbf{q}_i,g_i,\mathbf{a}_i,Y_i,\mathbf{q}'_i)\in T_i^{\bigcirc}$, and $\forall j\neq i,1\leq j\leq n$, s.t. $(\mathbf{q}_j,g_j,\mathbf{a}_j,Y_j,\mathbf{q}'_j)\in T_j^{\bigcirc\cup\Diamond}$, $\mathbf{a}_i\bowtie\mathbf{a}_j$ does not hold, and
  $$\begin{cases}\mathbf{a}=\bullet^u\mathbf{a}_i\bullet^v,\text{with } u=r_1+\cdots+r_{i-1},\\ v=r_{i+1}+\cdots+r_n,\ |\mathbf{a}|=m,\ g=g_i,\ Y=Y_i,\ \text{and}\\ \mathbf{q}'=\mathbf{q}_1\cdots\mathbf{q}_{i-1}\,\mathbf{q}'_i\,\mathbf{q}_{i+1}\cdots\mathbf{q}_n\end{cases}$$
- I is s.t. $\forall q\in Q.I(\mathbf{q})=\bigwedge_{i\in 1\ldots n}I(\mathbf{q_i})$
- $F=\{\,\mathbf{q}_1\cdots\mathbf{q}_n\in Q\mid\mathbf{q}_i\in F_i,\ i\in 1\ldots n\,\}$

The composition of (untimed) contract automata has been carefully revisited in Definition 4.

Case (1) generates match transitions starting from complementary actions of two operands' transitions, say $\mathbf{a}_i\bowtie\mathbf{a}_j$. If $(\mathbf{q}_j,g_j,\mathbf{a}_j,Y_j,\mathbf{q}'_j)\in T^{\Box}$, then the resulting match transition is marked necessary (i.e. $(\mathbf{q},g,\mathbf{a},Y,\mathbf{q}')\in T^{\Box}$), with $g=g_i\wedge g_j$ the conjunction of the guards. If both operands' complementary actions $\mathbf{a}_i$ and $\mathbf{a}_j$ are permitted, then so is their

resulting match transition t. All principals not involved in t remain idle. In case two complementary actions $\mathbf{a}_i$ and $\mathbf{a}_j$ are available for a match, i.e. $\mathbf{a}_i \bowtie \mathbf{a}_j$ as before, but only one of their guards (i.e. either g_i or g_j) is satisfied, then only the interleaving is possible and the guard $g = g_k \wedge \neg g_{k'}$ requires the guard of principal k (either g_i or g_j) to be satisfied and that of principal $k' \neq k$ not.

Case (2) generates all interleaved transitions whenever no complementary actions can be executed from the composed source state (i.e. $\mathbf{q}$). Now one operand executes its transition $t = (\mathbf{q}_i, g_i, \mathbf{a}_i, Y_i, \mathbf{q}'_i)$ and all other operands remain idle. Indeed, only the guard of principal i must be satisfied. The resulting transition is marked necessary (permitted) only if t is necessary (permitted, respectively). Note that condition $\mathbf{a}_i \bowtie \mathbf{a}_j$ excludes pre-existing match transitions of the operands to generate new matches.

Example 2 Figure 3 shows an excerpt of the composition Hotel $\otimes$ DiscountClient of the TSCA Hotel and DiscountClient of Figs. 1 and 2. The more relevant part of the TSCA is depicted, viz. whose semantics is an orchestration (from initial to final state). Note that, in the initial state, request $discount\,\square_\ell$ can either be matched with the offer $\overline{discount}$ if $y \geq 50$ or not matched if $y < 50$. Recall, moreover, that state invariants are omitted whenever they are true.

2.4 Controllability

We now revisit the different types of actions of TSCA (cf. Table 1) in light of the orchestration synthesis algorithm we will present in the next section.

To begin with, we define dangling configurations, i.e. those configurations that are either not reachable or from which no final state can be reached (i.e. which are not successful). The orchestration synthesis will be specified as a safety game, in which reachability of final states is satisfied through a dangling predicate. The following definition makes use of a set C of 'bad' configurations that are not to be traversed. Recall that $\mathscr{A}$ is a fixed TSCA.

Definition 5 (Dangling configuration) Let $C \subseteq \mathbb{C}$ and let $c = (\mathbf{q}, v) \in C$.

- We say that c is *reachable* in $\mathscr{A}$ given C, denoted as $c \in Reachable_{\mathscr{A}}(C)$, iff $(\mathbf{q_0}, \mathbf{0}) \xrightarrow{w}{}^* c$ without traversing configurations $(\mathbf{q}_r, v_r) \in C$.
- We say that c is *successful* in $\mathscr{A}$ given C, denoted as $c \in Successful_{\mathscr{A}}(C)$, iff $c \xrightarrow{w}{}^* (\mathbf{q_f}, v') \in F$ without traversing configurations $(\mathbf{q}_r, v_r) \in C$.

The set of *dangling configurations* in $\mathscr{A}$ given C is defined as

$$Dangling_{\mathscr{A}}(C) = \overline{Reachable_{\mathscr{A}}(C) \cap Successful_{\mathscr{A}}(C)}$$

Table 2: Controllability of actions requests and matches

action	requests	matches
urgent $\square_u$	uncontrollable	uncontrollable
greedy $\square_g$	semi-controllable	uncontrollable
lazy $\square_\ell$	semi-controllable	semi-controllable
permitted $\diamond$	controllable	controllable

In the sequel, abusing notation, we simply say that a state $\mathbf{q} \in Q$ is *dangling* (in $\mathscr{A}$ given C), denoted by $\mathbf{q} \in Dangling_{\mathscr{A}}(C)$, iff $(\mathbf{q}, v) \in Dangling_{\mathscr{A}}(C)$ for all possible valuations v. Moreover, let $Dangling(\mathscr{A}) = Dangling_{\mathscr{A}}(\varnothing)$.

As anticipated in the Introduction, orchestration synthesis for (service) contract automata resembles the synthesis of the *most permissive controller* known from SCT [31, 20]. Intuitively, the aim of SCT is to synthesise a most permissive controller enforcing 'good' (a.k.a. *successful*) computations, i.e. runs reaching a final state without traversing any given 'bad' (a.k.a. *forbidden*) state. To do so, SCT distinguishes *controllable* events (which the controller can disable) from *uncontrollable* events (which cannot be disabled). Ideally, actions that ruin a so-called safe orchestration of service contracts (a notion formally defined in Sect. 3, resembling a most permissive controller) should thus be removed by the synthesis algorithm. However, this is only allowed for actions that are effectively controllable in the orchestration.

Hence we need to characterise when an action of a TSCA (and the transition it labels) is controllable and when not. We also define 'when' a necessary request can be matched, stemming from the composition of TSCA (interleavings in Definition 4). Indeed, in TSCA it is possible to require that a necessary action (either a request or a match) must be matched in every possible configuration of the orchestration. Additionally, it is possible to require that a necessary action must be matched in at least one configuration from which it is executed. In the latter situation, it is possible to safely remove those requests (or matches) from the orchestration, as long as they do appear as part of a match in some other transition of the orchestration. Such necessary actions are called *semi-controllable*. Basically, a controllable action becomes uncontrollable in case all possible matches are removed, but not vice versa. Table 2 summarises the controllability of requests and matches of TSCA.

Recall that action offers are by definition permitted. All permitted actions (offers, requests, and matches) are fully controllable. As anticipated in the Introduction, necessary actions (*urgent*, *greedy*, and *lazy* requests) have an increasing degree of (semi-)controllability. An urgent request must be matched in every possible state in which it can be executed. Accordingly, both urgent requests and urgent matches are uncontrollable. A greedy request can be disabled by the controller as long as it is matched elsewhere; once it has been matched, it can no longer be disabled. In this case,

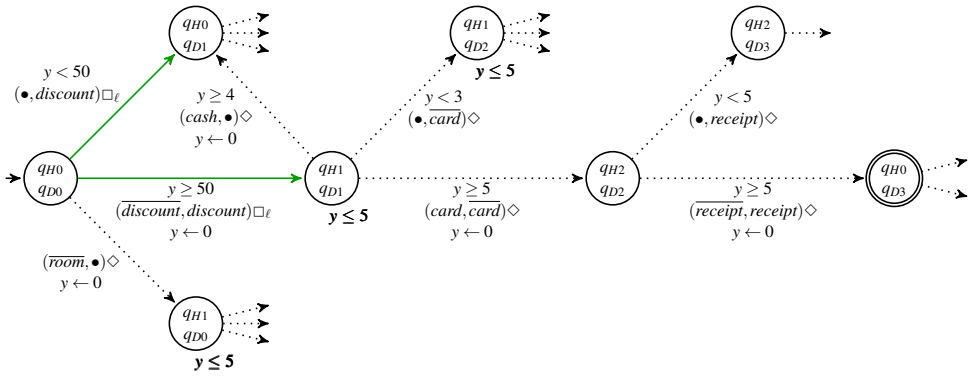

Fig. 3: Excerpt of composition Hotel $\otimes$ DiscountClient of the two TSCA in Figs. 1 and 2

greedy requests are semi-controllable, while greedy matches are uncontrollable. Finally, a lazy action only requires to be matched: its matches are controllable in the orchestration, provided that at least one match is available (i.e. both lazy requests and lazy matches are semi-controllable).

In the remainder of this section, we characterise semi-controllability of transitions (cf. Definition 6). Since we deal with real-time systems, this notion is defined on configurations. Recall from Table 2 that permitted actions are always controllable, while urgent actions are always uncontrollable.

A semi-controllable transition t is either a (greedy or lazy) request or a lazy match, and it is controllable in a TSCA $\mathscr{A}$ given C if there exists a (greedy or lazy) match transition t' in $\mathscr{A}$, which is reachable given C, and in both t and t' the same principal, in the same local state, does the same request, and additionally the target configuration is successful given C. Otherwise, t is uncontrollable. Recall that $\mathscr{A}$ is fixed.

Definition 6 (Semi-controllable transition) Let $C \subseteq \mathbb{C}$ and let $t = (\mathbf{q}_1, g_1, \mathbf{a}_1, Y_1, \mathbf{q}'_1)$ be a transition of $\mathscr{A}$.

Transition t is *semi-controllable* if it is a request on $a \in A^{\Box g} \cup A^{\Box \ell}$ or a match on $a \in A^{\Box \ell}$. At the same time, t is either controllable or uncontrollable in $\mathscr{A}$ given C.

- We say that transition t is *controllable* in $\mathscr{A}$ given C if $\exists t' = (\mathbf{q}_2, g_2, \mathbf{a}_2, Y_2, \mathbf{q}'_2) \in T^{\Box}$, s.t. $\mathbf{a}_2$ is a match, $\exists v$ s.t. $(\mathbf{q}_2, v) \in Reachable_{\mathscr{A}}(C)$, $(\mathbf{q}'_2, v') \in Post_{\mathscr{A}, \mathbf{a}_2}((\mathbf{q}_2, v)^{\nearrow})$, $(\mathbf{q}'_2, v') \in Successful_{\mathscr{A}}(C)$, $\mathbf{q}_{1(i)} = \mathbf{q}_{2(i)}$, and $\mathbf{a}_{1(i)} = \mathbf{a}_{2(i)} \in \mathsf{R} \cap (A^{\Box g} \cup A^{\Box \ell})$
- Otherwise we say that transition t is *uncontrollable* in $\mathscr{A}$ given C

In Definition 6, it does not suffice to require $\mathbf{q}_2$ or $\mathbf{q}'_2$ to be in $Dangling_{\mathscr{A}}(C)$. This is because it could be the case that $\mathbf{q}'_2$ is only reachable from a trace not passing through transition t', while $\mathbf{q}_2$ only reaches a final configuration through a trace not traversing t'. Hence, we need to explicitly require that for some v, $(\mathbf{q}_2, v)$ is reachable, and $(\mathbf{q}'_2, v')$ is a (timed) successor of $(\mathbf{q}_2, v)$ that reaches a final configuration.

Example 3 In Fig. 3, all transitions are permitted, except for the lazy discount actions that can be executed from the initial state. The transition $(\bullet, discount)\Box_\ell$ is thus a controllable lazy request, since the same request of DiscountClient is matched in the transition $(\overline{discount}, discount)\Box_\ell$. In the resulting orchestration (cf. Section 4) this will be the only match available for such a necessary action.

We thus call a transition *uncontrollable* if one of the above cases holds (i.e. urgent or greedy match, uncontrollable greedy or lazy request, or uncontrollable lazy match).

3 Orchestration Synthesis

In this section, we define the aforementioned synthesis of safe orchestrations of TSCA, considering both timing constraints and service requests with different levels of criticality. To this aim, we carefully adapt the synthesis algorithm for (modal) service contract automata defined in [11], which was based on the synthesis of the most permissive controller from SCT [31, 20]. To respect the timing constraints, the synthesis algorithm of TSCA presented below is computed using the notion of zones from timed games [3, 21].

The algorithm we will propose differs from the classical ones presented in [3, 21, 22] by combining two separate games, viz. *reachability* games and *safety* games. Indeed, as said before, the orchestration synthesis of TSCA is based on the synthesis of the *most permissive controller* from SCT, which ensures that (i) forbidden states are never traversed (a.k.a. a safety game) and (ii) marked states must be reachable (a.k.a. a reachability game). In the TSCA formalism, marked states are the final states of the composition of contracts, whereas bad states are those states that ruin an agreement among contracts (cf. Definitions 7 and 9 below).

We start by recalling the notions of *agreement* and *safety* on the languages of service contract automata from [11]. Intuitively, a trace is in agreement if it is a concatenation of matches, offers, and their modalities, while a TSCA is safe if all traces of its language are in agreement, and it admits

agreement if at least one of its traces is. Recall that $\mathscr{A}$ is a fixed TSCA.

Definition 7 (Agreement, safety) A trace accepted by $\mathscr{A}$ is in *agreement* if it belongs to the set

$$\mathfrak{A} = \{\, w \in (\Sigma^n \bigcirc)^* \mid \forall i \text{ s.t. } w_{(i)} = \mathbf{a}\bigcirc, \quad \mathbf{a} \text{ is a match or an offer, } n > 1 \,\}$$

We say that $\mathscr{A}$ is *safe* if $\mathscr{L}(\mathscr{A}) \subseteq \mathfrak{A}$; otherwise $\mathscr{A}$ is said to be *unsafe*. If $\mathscr{L}(\mathscr{A}) \cap \mathfrak{A} \neq \varnothing$, then $\mathscr{A}$ *admits agreement*.

Basically, an orchestration of TSCA enables the largest sub-portion of a composition of TSCA that is safe. Given the timed setting, the orchestration must consider clock valuations for each contract. Hence, the underlying transition system of a TSCA is inspected by the synthesis algorithm. The orchestration will be rendered as a strategy on this transition system such that only traces in agreement are enforced. We start by introducing the notion of strategy on TSCA and that of a well-formed strategy, i.e. a strategy avoiding dangling configurations.

Definition 8 (Strategy) A *strategy* f is a relation defined as $f : (\Sigma^n\{\bigcirc\} \cup \mathbb{R}^X_{\geq 0})^* \times (\Sigma^n\{\bigcirc\} \cup \mathbb{R}^X_{\geq 0})$ mapping traces of $\mathscr{A}$ to actions or delays s.t. given a trace $(\mathbf{q_0}, \mathbf{0}) \xrightarrow{w}{}^*(\mathbf{q}, v)$ of $\mathscr{A}$, then $(\mathbf{q}, v) \xrightarrow{\lambda} (\mathbf{q}', v')$, for some $\lambda \in f(w)$ and $(\mathbf{q}', v') \in \mathbb{C}$.

Furthermore, f is said to be *well-formed* given $C \subseteq \mathbb{C}$ if it is never the case that $(\mathbf{q}', v') \in \mathit{Dangling}_{\mathscr{A}}(C)$.

The language recognised by $\mathscr{A}$ following the strategy f is denoted by $\mathscr{L}_f(\mathscr{A})$ and $f^{\mathbb{C}}$ denotes the strategy allowing to traverse all and only configurations in $\mathbb{C}$.

We discuss some further differences compared to timed games. A TSCA strategy game can be seen as a 2-player game, in which a controller (i.e. an orchestrator) executes controllable transitions to enforce agreement among contracts, while an opponent executes uncontrollable transitions to drive the orchestrator to some 'bad' configuration, from which an agreement can no longer be enforced (cf. Definition 9). The opponent has precedence over the orchestrator, as long as its uncontrollable transitions are enabled (i.e. satisfied clock guards). Finally, fairness of TSCA guarantees that a final state is eventually reached, because traces recognised by TSCA languages are finite.

In timed games, strategies cannot prevent uncontrollable transitions from being executed. This follows from the notion of *outcome* of a strategy, which is used to characterise winning strategies. In TSCA, winning strategies are defined as those avoiding 'bad' configurations, while at the same time enforcing agreement among contracts. Next we will formally define bad configurations, i.e. configurations in *uncontrollable disagreement*. Basically, a configuration is in uncontrollable disagreement if the orchestrator cannot prevent a request of a principal from being executed without a corresponding offer (i.e. no match). In such configurations, the controller loses and thus the orchestration is unsafe. Note that the opponent can only win by reaching one such configuration. Indeed, unfair traces are ruled out in TSCA.

Definition 9 (Configuration in uncontrollable disagreement) Let $C \subseteq \mathbb{C}$.

We say that a transition $t = \mathbf{q} \xrightarrow{\mathbf{a}} \in T_{\mathscr{A}}$ is *forced* from a configuration $(\mathbf{q}, v)$ given C iff $(\mathbf{q}, v) \xrightarrow{\mathbf{a}}$ and (i) t is uncontrollable in $\mathscr{A}$ given C or (ii) $\mathbf{q} \notin F$ and no other $t' = \mathbf{q} \xrightarrow{\mathbf{a}'} \in T_{\mathscr{A}}$ is s.t. $(\mathbf{q}, v) \xrightarrow{\delta} (\mathbf{q}, v') \xrightarrow{\mathbf{a}'}$ for some delay δ.

A non-dangling configuration $(\mathbf{q}, v_1) \notin \mathit{Dangling}_{\mathscr{A}}(C)$ is said to be in *uncontrollable disagreement* in $\mathscr{A}$ given C iff $(\mathbf{q}, v_1) \xrightarrow{w}{}^*(\mathbf{q}_1, v_2)$ s.t. only timed or forced transitions are enabled and either (i) $w \notin \mathfrak{A}$, (ii) some configuration in C was traversed, or (iii) $\nexists w' \in \mathfrak{A}$ s.t. $(\mathbf{q}_1, v_2) \xrightarrow{w'}{}^*(\mathbf{q}_f, v_3)$ with $\mathbf{q}_f \in F_{\mathscr{A}}$ without traversing configurations in C.

A safe orchestration of TSCA can be interpreted as a *winning strategy* in terms of timed games, as defined next. Basically, a winning strategy enforces agreement among contracts, i.e. no bad configurations will ever be traversed.

Definition 10 (Winning strategy) Let f be a strategy given $C \subseteq \mathbb{C}$ and let U be a set of configurations in uncontrollable disagreement in $\mathscr{A}$ given C.

We say that f is a *winning strategy* given C if it is well-formed given C, it never traverses configurations in U, and $\mathscr{L}_f(\mathscr{A}) \subseteq \mathfrak{A}$. A winning strategy f given C is *maximal* if there is no winning strategy f' given C s.t. $\mathscr{L}_f(\mathscr{A}) \subseteq \mathscr{L}_{f'}(\mathscr{A})$.

Notation Before defining the synthesis of a safe orchestration, we define some auxiliary notions. Given a set of configurations $C \subseteq \mathbb{C}$ of a $\mathscr{A}$, the *uncontrollable predecessor* predicate $\mathit{uPred}_{\mathscr{A}}(C)$ is defined as all configurations from which some configuration in C is reachable by executing an uncontrollable transition. Formally,

$$\mathit{uPred}_{\mathscr{A}}(C) = \{\, c \mid \exists c' \in C, \quad c \xrightarrow{\mathbf{a}\square} c' \text{ is uncontrollable in } \mathscr{A} \text{ given } C \,\}$$

We borrow the notion of *safe timed predecessor* of a set $C_1 \subseteq \mathbb{C}$ with respect to a set $C_2 \subseteq \mathbb{C}$ from [21]. Intuitively, a configuration c is contained in the *(safe timed) predecessor* predicate $\mathit{Pred}_{\mathscr{A},t}(C_1, C_2)$ if from c it is possible to reach a configuration $c' \in C_1$ by time elapsing and the trace from c to c' avoids configurations in C_2. Formally,

$$\mathit{Pred}_{\mathscr{A},t}(C_1, C_2) = \{\, c \in \mathbb{C} \mid \exists\, \delta \in \mathbb{R}_{\geq 0} \text{ s.t. } c \xrightarrow{\delta} c', \quad c' \in C_1, \text{ and } \mathit{Post}_{\mathscr{A},[0,\delta]}(c) \subseteq \overline{C_2} \,\}$$

in which

$$\mathit{Post}_{\mathscr{A},[0,\delta]}(c) = \{\, c' \in \mathbb{C} \mid \exists t \in [0, \delta] \text{ s.t. } c \xrightarrow{t} c' \,\}$$

and

$$\overline{C_2} = \mathbb{C} \setminus C_2$$

We are now ready to define the synthesis of a safe orchestration of TSCA. Let $\mathscr{A}^-$ denote the TSCA obtained from $\mathscr{A}$ by replacing $T_{\mathscr{A}}$ with $T_{\mathscr{A}^-} = \{ t = \mathbf{q} \xrightarrow{\mathbf{a}^{\bigcirc}} \mid t \in T_{\mathscr{A}}$ and $(\bigcirc \neq \Diamond \vee \mathbf{a}$ is not a request$)\}$, i.e. all permitted requests are pruned from $\mathscr{A}$.

Definition 11 (Safe orchestration) Let $\phi : 2^{\mathbb{C}} \to 2^{\mathbb{C}}$ be a monotone function on the complete partial order $(2^{\mathbb{C}}, \subseteq)$ s.t. $\phi(\mathbb{C}_{i-1}) = \mathbb{C}_i$, where

$$\mathbb{C}_0 = \{ c \mid c \in \mathbb{C},\ c \xrightarrow{\mathbf{a}\Box},\ \mathbf{a} \text{ is an uncontrollable request in } \mathscr{A}^- \text{ given } \varnothing \}$$

and

$$\mathbb{C}_i = Pred_{\mathscr{A}^-,t}(\mathbb{C}_{i-1} \cup uPred_{\mathscr{A}^-}(\mathbb{C}_{i-1}), moPred_{\mathscr{A}^-}(\overline{\mathbb{C}_{i-1}})) \cup Dangling_{\mathscr{A}^-}(\mathbb{C}_{i-1}) \cup \mathbb{C}_{i-1}$$

and let

$$\mathbb{C}^* = sup(\{ \phi^n(\mathbb{C}_0) \mid n \in \mathbb{N} \})$$

be the least fixed point of ϕ.

The *safe orchestration* of $\mathscr{A}$ is defined as the strategy:

$$f^* = \begin{cases} \bot & \text{if } (\mathbf{q_0}, \mathbf{0}) \in \mathbb{C}^* \\ f^{\overline{\mathbb{C}}^*} & \text{otherwise} \end{cases}$$

This definition is such that whenever the initial configuration $(\mathbf{q}_0, \mathbf{0})$ belongs to $\mathbb{C}^*$, then the orchestration is *empty* (i.e. there exists no strategy to enforce agreement among contracts, while avoiding configurations in uncontrollable disagreement). If $(\mathbf{q}_0, \mathbf{0})$ does not belong to $\mathbb{C}^*$, then the set $\overline{\mathbb{C}}^*$ identifies a winning strategy that characterises a safe orchestration of contracts. This strategy allows as many transitions as possible without ever traversing configurations in $\mathbb{C}^*$. Indeed, the controller can avoid principals reaching bad configurations in $\mathbb{C}^*$, meanwhile guaranteeing all requirements to be satisfied. $\overline{\mathbb{C}}^*$ moreover identifies the maximal winning strategy, i.e. f^* allows all controllable match/offer transitions to configurations not belonging to $\mathbb{C}^*$ (recall that f^* is not a function). Note that f^* is computable due to the finiteness of the symbolic configurations and the monotonicity of the fixed-point computation [21]. As the next theorem states, f^* is the maximal well-formed winning strategy.

Theorem 1 (Maximal winning strategy) *Let strategy f^* be as computed through Definition 11.*

If $f^* = \bot$, then there exists no well-formed winning strategy f given $\mathbb{C}^*$. Otherwise, f^* is the maximal well-formed winning strategy in $\mathscr{A}$ given $\mathbb{C}^*$.

Proof If $(\mathbf{q_0}, \mathbf{0}) \in \mathbb{C}^*$, then trivially there exists no well-formed strategy f given $\mathbb{C}^*$ (i.e. $(\mathbf{q_0}, \mathbf{0}) \in Dangling(\mathbb{C}^*)$). It remains to prove that if $(\mathbf{q_0}, \mathbf{0}) \notin \mathbb{C}^*$, then (1) f^* is a well-formed winning strategy given $\mathbb{C}^*$ and (2) f^* is the maximal winning strategy.

1. By Definition 10, we distinguish the next three cases: (1a) f^* never traverses configurations in $Dangling_{\mathscr{A}}(\mathbb{C}^*)$, (1b) $\mathscr{L}_{f^*}(\mathscr{A}) \subseteq \mathfrak{A}$ (i.e. it recognises traces in agreement), and (1c) f^* never traverses configurations in U.
   (a) From Definition 11 and the fixed-point computation, it follows that $Dangling_{\mathscr{A}}(\mathbb{C}^*) \subseteq \mathbb{C}^*$, hence f^* is well-formed given $\mathbb{C}^*$.
   (b) First, by Definition 11, all controllable requests are disabled in f^* via $\mathscr{A}^-$. Hence, it remains to prove that no uncontrollable requests can be fired by f^*.
   By contradiction, we assume that there exists a trace $(\mathbf{q_0}, \mathbf{0}) \to^* (\mathbf{q}, v) \xrightarrow{\lambda} (\mathbf{q}', v')$ with $\lambda = \mathbf{a}\Box$ allowed by f^* s.t. $(\mathbf{q_0}, \mathbf{0}) \cdots (\mathbf{q}, v) \notin \mathbb{C}^*$ and $(\mathbf{q}', v')$ is the first configuration encountered s.t. $\mathbf{a}$ is an uncontrollable request in $\mathbb{C}^*$ (i.e. a trace not in agreement is recognised by f^*). Assume $i-1$ to be the fixed-point iteration s.t. $\mathbf{a}$ is an uncontrollable request in $\mathbb{C}_{i-1}$. By Definition 11, we have $(\mathbf{q}, v) \in \mathbb{C}_i \subseteq \mathbb{C}^*$ by $Pred_{\mathscr{A},t}$, a contradiction.
   (c) Again by contradiction, we assume that there exists a trace $(\mathbf{q_0}, \mathbf{0}) \to^* (\mathbf{q}, v) \to (\mathbf{q}', v')$ allowed by f^* s.t. $(\mathbf{q_0}, \mathbf{0}) \cdots (\mathbf{q}, v) \notin \mathbb{C}^*$ and $(\mathbf{q}', v')$ is the first configuration encountered s.t. $(\mathbf{q}', v') \in U$ given $\mathbb{C}^*$. As we already proved that $(\mathbf{q}', v') \notin Dangling_{\mathscr{A}}(\mathbb{C}_{i-1})$, i.e. case (1a), and requests cannot be fired, i.e. case (1b), it follows that conditions (1) and (3) of Definition 9 are not met. It remains to show that condition (2) of Definition 9 leads to a contradiction.
   Assume by contradiction that condition (2) holds, i.e. from $(\mathbf{q}', v')$ a configuration $c \in \mathbb{C}^*$ is reached by only executing forced transitions and time steps, and let $i-1$ be the first fixed-point iteration s.t. $c \in \mathbb{C}_{i-1}$. We proceed by induction on the length of the trace $(\mathbf{q}', v') \to^* c$. In the base case, $(\mathbf{q}', v') \xrightarrow{\delta} (\mathbf{q}', v'') \xrightarrow{\mathbf{a}\Box} c$ for some delay δ and an uncontrollable transition $(\mathbf{q}', v'') \xrightarrow{\mathbf{a}\Box}$. Now note that it cannot be the case that $(\mathbf{q}', v'') \xrightarrow{\mathbf{a}\Diamond}$ (i.e. item (ii) of forced transition in Definition 9). Indeed, in this case we would have $(\mathbf{q}', v') \in Dangling_{\mathscr{A}}(\mathbb{C}_{i-1})$, a contradiction. Then $(\mathbf{q}', v') \in Pred_{\mathscr{A},t}(\mathbb{C}_{i-1} \cup uPred_{\mathscr{A}}(\mathbb{C}_{i-1}),\ moPred_{\mathscr{A}}(\overline{\mathbb{C}_{i-1}}))$ and, by Definition 11, $(\mathbf{q}', v') \in \mathbb{C}_i \subseteq \mathbb{C}^*$. Thus transition $t = (\mathbf{q}, v) \xrightarrow{\lambda} (\mathbf{q}', v')$ is not allowed in f^* (recall that f^* only allows transitions to configurations in $\overline{\mathbb{C}}^*$), a contradiction. For the inductive step, we assume $(\mathbf{q}', v') \xrightarrow{\delta} (\mathbf{q}', v'') \xrightarrow{\mathbf{a}\Box} (\mathbf{q}'', v'') \to^* c$ and $(\mathbf{q}'', v'') \in \mathbb{C}_j$, for some $j > i$, by the induction hypothesis. Similarly to the base case, we reach the contradiction $(\mathbf{q}', v') \in \mathbb{C}_{j+1} \subseteq \mathbb{C}^*$.

2. We need to show that f^* is the maximal well-formed winning strategy given $\mathbb{C}^*$. Assume by contradiction that there exists another winning strategy f' given $\mathbb{C}^*$ s.t. $\mathscr{L}_{f^*}(\mathscr{A}) \subset \mathscr{L}_{f'}(\mathscr{A})$. By Definition 11, all transitions that are disabled by f^* are either (2a) controllable requests or (2b) transitions leading to configurations in $\mathbb{C}^*$. Hence, f' enables a transition satisfying either case (2a) or (2b):
   (a) In this case, $\mathscr{L}_{f'}(\mathscr{A}) \not\subseteq \mathfrak{A}$, i.e. f' is not winning, a contradiction.
   (b) By Definition 11,

$$Pred_{\mathscr{A}^-,t}(\mathbb{C}^* \cup uPred_{\mathscr{A}^-}(\mathbb{C}^*), moPred_{\mathscr{A}^-}(\overline{\mathbb{C}^*})) \cup Dangling_{\mathscr{A}^-}(\mathbb{C}^*) = \mathbb{C}^*$$

   This implies that f' enables a transition to a configuration into one of the following two cases. Either (2(b)i) $Pred_{\mathscr{A}^-,t}(\mathbb{C}^* \cup uPred_{\mathscr{A}^-}(\mathbb{C}^*), moPred_{\mathscr{A}^-}(\overline{\mathbb{C}^*}))$ or (2(b)ii) $Dangling_{\mathscr{A}^-}(\mathbb{C}^*)$.
      i. In this case, by Definition 11, it is not difficult to see that $Pred_{\mathscr{A}^-,t}$ computes exactly all configurations from which an uncontrollable request cannot be avoided to be executed or a dangling configuration cannot be avoided to be reached, i.e. configurations that are in uncontrollable disagreement given $\mathbb{C}^*$. Hence, by Definition 10, f' is not a winning strategy, a contradiction.
      ii. In this case, by Definition 10, f' is not well-formed, and hence not a winning strategy, a contradiction. □

Example 4 Recall the composition $\mathsf{Hotel} \otimes \mathsf{DiscountClient}$ from Fig. 3. We can apply the synthesis algorithm to compute its safe orchestration f^*. In f^*, the request transition $(\bullet, discount\Box_\ell)$ is removed because it is controllable (cf. Example 3). So the language recognised by f^* is the singleton

$$\mathscr{L}_{f^*}(\mathsf{Hotel} \otimes \mathsf{DiscountClient}) = \{(\overline{discount}, discount)\Box_\ell(card, \overline{card})\Diamond(\overline{receipt}, receipt)\Diamond\}$$

From [21] (Theorem 4), we can reduce the computation of the set $Pred_{\mathscr{A},t}$ to the following basic operations on zones:

$$Pred_{\mathscr{A},t}(C_1, C_2) = (C_1^{\swarrow} \setminus C_2^{\swarrow}) \cup ((C_1 \cap C_2^{\swarrow}) \setminus C_2)^{\swarrow}$$

Similarly, we will now provide procedures to compute the newly introduced sets $moPred_{\mathscr{A}}$, $uPred_{\mathscr{A}}$, and $Dangling_{\mathscr{A}}$ using basic operations on zones. Together, these provide an effective procedure for computing $\mathbb{C}^*$ (and hence a safe orchestration). The set $moPred_{\mathscr{A}}$ can be computed from $Pred_{\mathscr{A}}$ by only considering discrete steps that are not requests. Conversely, both $uPred_{\mathscr{A}}$ and $Dangling_{\mathscr{A}}$ require visiting the symbolic configurations of $\mathscr{A}$. We now show how to compute these sets, first $Dangling_{\mathscr{A}}$ and then $uPred_{\mathscr{A}}$.

Theorem 2 (Compute dangling configuration) *Let $C \subseteq \mathbb{C}$ and let ϕ be as defined in Definition 11 s.t. $\phi(\mathbb{C}_{i-1}) = \mathbb{C}_i$ and $\mathbb{C}^* = sup(\{\phi^n(\mathbb{C}_0) \mid n \in \mathbb{N}\})$.*

1. *The set of* reachable configurations *in $\mathscr{A}$ given C is computed as $Reachable_{\mathscr{A}}(C) = \mathbb{C}^*$, with*

$$\mathbb{C}_0 = (\mathbf{q}_0, \mathbf{0})^{\nearrow} \setminus C^{\nearrow}$$

and

$$\mathbb{C}_i = \bigcup_{\mathbf{a}} (Post_{\mathscr{A},\mathbf{a}}(\mathbb{C}_{i-1})^{\nearrow} \setminus C^{\nearrow}) \cup \mathbb{C}_{i-1}$$

2. *The set of* successful configurations *in $\mathscr{A}$ given C is computed as $Successful_{\mathscr{A}}(C) = \mathbb{C}^*$, with*

$$\mathbb{C}_0 = \{(\mathbf{q}_f, v) \mid \mathbf{q}_f \in F_{\mathscr{A}} \text{ and } v \in \mathbb{R}_{\geq 0}^{X_{\mathscr{A}}}\} \setminus C$$

and

$$\mathbb{C}_i = Pred_{\mathscr{A},t}(\mathbb{C}_{i-1} \cup (Pred_{\mathscr{A}}(\mathbb{C}_{i-1}) \setminus C), C) \cup \mathbb{C}_{i-1}$$

3. *The set of* dangling configurations *in $\mathscr{A}$ given C is computed as*

$$Dangling_{\mathscr{A}}(C) = \overline{Successful_{\mathscr{A}}(C \cup \overline{Reachable_{\mathscr{A}}(C)})}$$

Proof Note that in the first two cases (i.e. computing the sets of reachable and successful configurations) the function ϕ is monotonic, and by finiteness of the symbolic graph all fixed-point computations are computable.

1. We have to prove $Reachable_{\mathscr{A}}(C) = \mathbb{C}^*$. By Definition 5, the set $Reachable_{\mathscr{A}}(C)$ contains all configurations that are (1a) reachable without (1b) traversing configurations in C.
   (a) By the definition of timed successor $^{\nearrow}$ and $Post_{\mathscr{A},\mathbf{a}}$, we only have to prove that the set difference operation does not introduce disconnected configurations, or else all configurations are trivially reachable. Take a configuration $c \in C \cap Post_{\mathscr{A},\mathbf{a}}(\mathbb{C}_{i-1})^{\nearrow}$ for some $\mathbb{C}_i = \bigcup_{\mathbf{a}}(Post_{\mathscr{A},\mathbf{a}}(\mathbb{C}_{i-1})^{\nearrow} \setminus C^{\nearrow}) \cup \mathbb{C}_{i-1}$, where i is the least index s.t. c appears. (The case for $\mathbb{C}_0$ is identical.)
   By hypothesis, $c \notin \mathbb{C}_{i-1}$ and $\mathbb{C}_{i-1}$ is such that it only contains reachable configurations. Given a configuration $c' \in Post_{\mathscr{A},\mathbf{a}}(\mathbb{C}_{i-1})$, either there exists a time interval δ s.t. $c' \xrightarrow{\delta} c$, or $\delta = 0$ and $c = c'$. It remains to prove that $\forall \delta' \nexists c'' \in \mathbb{C}_i$ s.t. $c \xrightarrow{\delta'} c''$. Indeed, c'' would be the only possible case of a disconnected configuration belonging to $\mathbb{C}_i$. By definition of timed successor, $\forall \delta' \forall c''$ s.t. $c \xrightarrow{\delta'} c''$ implies $c'' \in C^{\nearrow}$, hence by set difference $c'' \notin \mathbb{C}_i$, and the statement follows.
   (b) By contradiction, assume there exists $c \in \mathbb{C}^* \cap C$, and let i be the iteration s.t. $\mathbb{C}_i = \mathbb{C}^*$. Then either $\mathbb{C}_i = (\mathbf{q}_0, \mathbf{0})^{\nearrow} \setminus C^{\nearrow}$ or $\mathbb{C}_i = \bigcup_{\mathbf{a}}(Post_{\mathscr{A},\mathbf{a}}(\mathbb{C}_{i-1})^{\nearrow} \setminus C^{\nearrow}) \cup \mathbb{C}_{i-1}$, and in both cases by definition of set difference $c \notin C$, a contradiction.

2. We now prove $Successful_{\mathscr{A}}(C) = \mathbb{C}^*$. By Definition 5, $Successful_{\mathscr{A}}(C)$ contains all configurations from which it is possible to reach a final configuration without traversing configurations in C. Similar to the previous case, we must prove that (2a) no disconnected configurations (from all final configurations) are contained in $\mathbb{C}^*$ and (2b) $C \cap \mathbb{C}^* = \varnothing$.
   (a) To begin with, all configurations in $\mathbb{C}_0$ are trivially connected to a final configuration. Furthermore, by the definition of $Pred_{\mathscr{A},t}$, each configuration in $\mathbb{C}_i$ is connected to some configuration in $\mathbb{C}_{i-1}$ (and by induction to a final configuration) by time delay and by a discrete step (i.e. $Pred_{\mathscr{A}}(\mathbb{C}_{i-1})$).
   (b) For the base case it holds that $C \cap \mathbb{C}_0 = \varnothing$. Furthermore, for each $\mathbb{C}_i$, it follows from the definition of $Pred_{\mathscr{A},t}(\mathbb{C}_{i-1} \cup (Pred_{\mathscr{A}}(\mathbb{C}_{i-1}) \setminus C), C)$ that no configuration in C is ever traversed.
3. Finally, we must prove the following: $Dangling_{\mathscr{A}}(C) = \overline{Successful_{\mathscr{A}}(C \cup \overline{Reachable_{\mathscr{A}}(C)})}$. From Definition 5, $Dangling_{\mathscr{A}}(C) = \overline{Reachable_{\mathscr{A}}(C) \cap Successful_{\mathscr{A}}(C)}$, i.e. all configurations that are not reachable or not successful. In a similar way, $Successful_{\mathscr{A}}(C \cup \overline{Reachable_{\mathscr{A}}(C)})$ contains by definition all configurations that are successful without traversing either unreachable configurations given C or configurations in C. This then implies that $Successful_{\mathscr{A}}(C \cup \overline{Reachable_{\mathscr{A}}(C)}) \subseteq Reachable_{\mathscr{A}}(C)$, i.e. it contains only successful configurations given C that are reachable given C, which implies the statement. In comparison with $\overline{Reachable_{\mathscr{A}}(C) \cap Successful_{\mathscr{A}}(C)}$, the computation of $\overline{Successful_{\mathscr{A}}(C \cup \overline{Reachable_{\mathscr{A}}(C)})}$ is quicker because it first computes all reachable configurations given C', and then all successful ones. Indeed, unreachable configurations are identified as 'bad' to avoid them in the computation of the $Successful_{\mathscr{A}}$ predicate, which is not the case for the former definition. □

Note that the dangling configurations are efficiently computed by combining a forward exploration (i.e. reachable configurations) with a backward exploration (i.e. successful configurations), which makes it possible to ignore unreachable successful configurations. We have thus determined an effective procedure to compute $Dangling_{\mathscr{A}}(C)$ by means of basic operations on zones.

It remains to show how to compute the set $uPred_{\mathscr{A}}$ of uncontrollable predecessors. To this aim, we define the following procedure, which makes use of Theorem 2.

Lemma 1 (Compute uncontrollable predecessors) *Let* $C \subseteq \mathbb{C}$.

The set of uncontrollable predecessors *of* C *in* $\mathscr{A}$ *is computed as*

$$uPred_{\mathscr{A}}(C) = \{\, c \in \mathbb{C} \mid \exists c' \in C : c \xrightarrow{\mathbf{a}\Box} c' \in unc_{\mathscr{A}}(C) \,\}$$

in which

$$\begin{aligned} unc_{\mathscr{A}}(C) = \{\, (\mathbf{q},v) \xrightarrow{\mathbf{a}\Box} \mid\ & (\mathbf{q},v) \in \mathbb{C} \wedge \\ & (\mathbf{a} \textit{ is urgent} \vee \mathbf{a} \textit{ is a greedy match} \vee \\ & (\nexists (\mathbf{q}_2,v) \in Reachable_{\mathscr{A}}(C), (\mathbf{q}_2',v') \in Successful_{\mathscr{A}}(C). \\ & (\mathbf{q}_2',v') \in Post_{\mathscr{A},\mathbf{a}'}(\mathbf{q}_2,v)^{\nearrow} \wedge\ \mathbf{a}_{(i)} = \mathbf{a}'_{(i)} = a \in \mathsf{R} \\ & \qquad \wedge\ \mathbf{a}' \textit{ is a match} \wedge\ \mathbf{q}_{(i)} = \mathbf{q}_{2(i)}))\,\} \end{aligned}$$

Proof By definition, an uncontrollable transition has an action that is either urgent, a greedy match, an uncontrollable greedy or lazy request, or an uncontrollable lazy match. As action $\mathbf{a}$ is classified as necessary ($\Box$), it remains to show that

$$\begin{aligned} & (\nexists (\mathbf{q}_2,v) \in Reachable_{\mathscr{A}}(C), (\mathbf{q}_2',v') \in Successful_{\mathscr{A}}(C). \\ & (\mathbf{q}_2',v') \in Post_{\mathscr{A},\mathbf{a}'}(\mathbf{q}_2,v)^{\nearrow} \wedge\ \mathbf{a}_{(i)} = \mathbf{a}'_{(i)} = a \in \mathsf{R} \\ & \qquad \wedge\ \mathbf{a}' \text{ is a match} \wedge\ \mathbf{q}_{(i)} = \mathbf{q}_{2(i)}) \end{aligned}$$

is a predicate that identifies exactly those transitions that are an uncontrollable greedy or lazy request or an uncontrollable lazy match (given C). This follows trivially from Definition 6. □

By combining the above results, a safe orchestration of TSCA could now be implemented using available libraries for timed games [23, 28] that offer primitive operations on zones (i.e. $\cup$, $\cap$, $\setminus$, $^{\nearrow}$, and $^{\swarrow}$).

4 Running Example Revisted

We now continue our running example with an additional PriviledgedClient, depicted in Fig. 4. This priviledged client optionally asks for a discount room via a permitted request, but after 8 t.u. have passed (in the model's initial state) she urgently requests a normal room. All state invariants of the model are true. Consider the following composition

$$(\mathsf{Hotel} \otimes \mathsf{DiscountClient}) \otimes \mathsf{PriviledgedClient}$$

In orchestration f^* of this composition, the discount request of the DiscountClient could be matched before one of the requests of PriviledgedClient.

However, this interaction is prevented in f^*. To see this, let $\mathbf{a} = (\overline{discount}, discount, \bullet)\Box_{\ell}$, let $\mathbf{b} = (\bullet, \bullet, room)\Box_{u}$, let

$$t1 = ((q_{H0}, q_{D0}, q_{P0}), y \geq 50, \mathbf{a}, y \leftarrow 0, (q_{H1}, q_{D1}, q_{P0}))$$

and let

$$t2 = ((q_{H1}, q_{D1}, q_{P0}), x \geq 8, \mathbf{b}, \varnothing, (q_{H1}, q_{D1}, q_{P1}))$$

Note that $t1$ is not enabled by f^*, since otherwise we could reach a configuration c_2 in uncontrollable disagreement via

$$c_0 \xrightarrow{\delta=50} c_1 \xrightarrow{\mathbf{a}} c_2 \xrightarrow{\delta=0} c_2 \xrightarrow{\mathbf{b}}$$

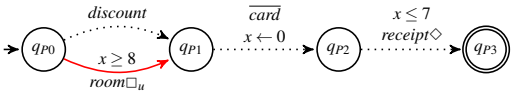

Fig. 4: TSCA model PriviledgedClient

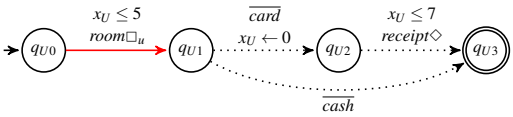

Fig. 5: TSCA model BusinessClientU

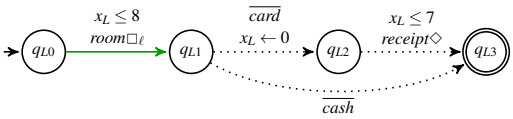

Fig. 6: TSCA model BusinessClientL

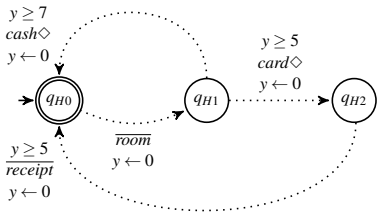

Fig. 7: TSCA model Hotel2

In c_2, the uncontrollable transition t_2 is enabled, but urgent request $\mathbf{b}$ is not matched, thus violating agreement. In f^*, the first enabled transition is

$$((q_{H0},q_{D0},q_{P0}),x \geq 8,\mathbf{c},y \leftarrow 0,(q_{H1},q_{D0},q_{P1}))$$

where $\mathbf{c} = (\overline{room},\bullet,room)\Box_u$, i.e. the offer $(\overline{room},\bullet)$ from state (q_{H0},q_{C0}) is synchronised with the request $room\Box_u$.

Hence, PriviledgedClient interacts with Hotel prior to DiscountClient, who is served successively. This is only possible as both the lazy request $(\bullet,discount)\Box_\ell$ and the lazy match $(\overline{discount},discount)\Box_\ell$ of Hotel $\otimes$ DiscountClient are semi-controllable and are delayed in the orchestration of (Hotel $\otimes$ DiscountClient) $\otimes$ PriviledgedClient.

Consequently we consider the TSCA models depicted in Figs. 5–7, which are variants of the previous contracts (cf. also Figs. 1 and 2 in Section 2). In particular, the BusinessClientU requests urgently a room within 5 t.u., the BusinessClientL requests lazily a room within 8 t.u, and the variant Hotel2 offers only a normal room (i.e. no discount room) and the invariant $y \leq 5$ has been removed from its state q_{H1}. In fact, all state invariants of the models in Figs. 5–7 are true.

First, we have a look at the following composition

$$(\text{Hotel2} \otimes \text{BusinessClientL}) \otimes \text{BusinessClientU}$$

whose orchestration is empty (i.e. there is no agreement). In the initial state of Hotel2 $\otimes$ BusinessClientL, the room offer is available only after 8 t.u., otherwise it is matched by BusinessClientL's lazy room request. As BusinessClientU's urgent room request must be matched within 5 t.u., it cannot be matched prior to BusinessClientL's lazy room request. This is a violation, so the initial configuration is in uncontrollable disagreement.

Next, we have a look at the following composition

$$(\text{Hotel2} \otimes \text{BusinessClientU}) \otimes \text{BusinessClientL}$$

and consider its orchestration f^*. A part of the behaviour allowed by f^* is depicted in Fig. 8 in the fragment marked with ✓. In this figure, a transition is executed as soon as it is enabled.

In this case, the BusinessClientU performs the transaction with the hotel first. For card payments, the minimum time required to reach state $\mathbf{q} = (q_{H0},q_{U3},q_{L0})$ is $5+5=10$ t.u., with clocks valuation $v = (y = 0, x_U = 5, x_L = 10)$. In $(\mathbf{q},v)$ (the top leftmost configuration in Fig. 8), the (lazy) necessary room request of the BusinessClientL can no longer be satisfied as it should have been matched within 8 t.u., thus violating agreement (since every necessary request should be matched). Hence f^* forbids card payments made by the BusinessClientU. Moreover, note that also the two previous configurations (contained in the fragment marked with ϟ in Fig. 8) are forbidden in orchestration f^*, because they are in uncontrollable disagreement.

If, however, the BusinessClientU performs a cash payment, then the minimum time required to reach state $\mathbf{q}$ is 7 t.u., with clocks valuation $v' = (y = 0, x_U = 7, x_L = 7)$. Indeed, in configuration $(\mathbf{q},v')$ (the central rightmost configuration in the fragment marked with ✓ in Fig. 8) the lazy room request of the BusinessClientL can be matched by the room offer of Hotel2, and successively the orchestration enables this client to pay either by cash or by card. Therefore, to satisfy the BusinessClientL's lazy room request, in the resulting safe orchestration the BusinessClientU is only allowed to perform cash payments.

5 Discussion of Innovations

In this section, the innovations proposed in this paper are illustrated by means of a comparison of the TSCA formalism with Timed I/O Automata (TIOA) [22], which is a widely used formalism with tool support for the specifications of real-time systems and which comprehends several extensions (e.g. stochastic, hybrid). However, the comparisons remain valid with respect to similar formalisms, such as Interface Automata [1]. The innovations concern *compositionality*, *games*, and *controllability*. We will discuss these three innovations separately, since they are independent from one

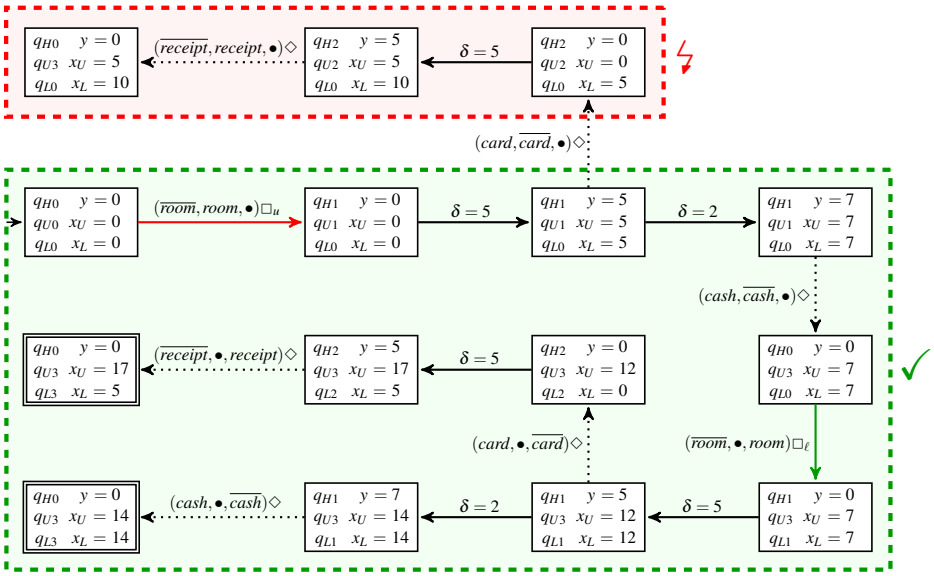

Fig. 8: Excerpt of $TS_{(\mathsf{Hotel2}\otimes\mathsf{BusinessClientU})\otimes\mathsf{BusinessClientL}}$, whose fragment marked with ✓ is allowed in the safe orchestration whereas the one marked with ϟ is not

another, but they have been seamlessly integrated into the framework of TSCA. The first innovation is inherited from the basic contract automata of [7], while the remaining two are specific to the TSCA introduced in this paper.

Compositionality First, we discuss composition. In TIOA, there are two levels of specification. One concerns the single components (e.g. $\mathtt{A}$, $\mathtt{B}$, $\mathtt{C}$) and the other their composition (e.g. $\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}$). Indeed, the composition operator of TIOA is *associative*, i.e. the order in which the components are composed is irrelevant, as the result will always be the same (i.e. $(\mathtt{A}\otimes\mathtt{B})\otimes\mathtt{C}=\mathtt{A}\otimes(\mathtt{B}\otimes\mathtt{C})=(\mathtt{A}\otimes\mathtt{C})\otimes\mathtt{B}=\mathtt{A}\otimes\mathtt{B}\otimes\mathtt{C}$).

TSCA, on the contrary, allow to model complex composition patterns based on the order in which service contracts are composed. This is due to their *non-associative* composition operator, according to which pre-existing matches of service offers and requests are not rearranged when new service contracts join the composition. We have seen an example of this in the example of Section 4, where the compositions $(\mathsf{Hotel2}\otimes\mathsf{BusinessClientL})\otimes\mathsf{BusinessClientU}$ and $(\mathsf{Hotel2}\otimes\mathsf{BusinessClientU})\otimes\mathsf{BusinessClientL}$ exhibit fundamentally different behaviour: in the former case the orchestration is empty, whereas in the latter case it is not (and part of the behaviour allowed by its safe orchestration f^* is depicted in Fig. 8).

The view on compositionality adopted by TSCA allows to primitively model scope restriction in the style of the π-calculus [30]. If, e.g., $\mathtt{A}$ and $\mathtt{B}$ are privately interacting on an action a, in the composition $(\mathtt{A}\otimes\mathtt{B})\otimes\mathtt{C}$ the principal $\mathtt{C}$ is not allowed to interfere with $\mathtt{A}$ and $\mathtt{B}$ on such action.

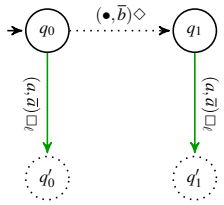

Fig. 9: Automaton $\mathscr{A}$

Games Second, we discuss the introduction of a new kind of timed game, for which we have formalised the synthesis of the most permissive strategy. Indeed, the TSCA strategy game combines the traditional reachability and safety games (cf., e.g., [3, 21] and the game-based specification theory for TIOA [22]) into one single setting, with the difference that only finite traces of execution are considered.

The approach of this paper is useful in any real-time most permissive controller synthesis problem, for which both forbidden and successful configurations are specified. We have proved that the TSCA strategy game can effectively be solved by using the existing operators on zones used for timed games, which can be efficiently computed thanks to existing algorithms available from the literature [23, 28].

We believe that it would not be extremely difficult to add this kind of timed games to the UPPAAL toolset [16, 17] (`http://www.uppaal.org/`) but, as far as we know, this has not been done yet. Since the UPPAAL code is not open source, we plan to investigate the issue in collaboration with the UPPAAL developers.

Controllability Third, we discuss the identification of a new type of modality, namely *semi-controllability*. We believe this to be our most important innovation and we illustrate it in detail with the help of a simple example.

The automaton $\mathscr{A}$ depicted in Figure 9 depicts (in an informal way) a composition $\mathtt{A} \otimes \mathtt{B}$ in which A and B can either synchronise on a semi-controllable action a in their initial state or after B has performed an internal action (say, b). We underline that such a composition can easily be obtained in other automata-based formalisms (such as, e.g., TIOA). Moreover, we have willingly left unspecified whether a bad or a successful configuration is reached after one of the two synchronisations depicted in Figure 9 has occurred. Indeed, as stated above, the notion of semi-controllability is independent from both the specific formalism being used and the requirement (e.g. agreement in case of TSCA) to be enforced.

To the best of our knowledge, there exists no synthesis algorithm in the literature capable of producing a controller (or strategy) which guarantees that *at least* one of these two synchronisation actually occurs. Indeed, in case action a were declared controllable, then the synthesis algorithm could possibly prune both synchronisations in $\mathscr{A}$. On the other hand, in case a were declared uncontrollable, then the synthesis algorithm cannot possibly prune any of the two synchronisations. This apparently stems from the fact that traditionally uncontrollable actions are typically related to an unpredictable environment. However, the interpretation of such actions as *necessary* service requests to be fulfilled in a service contract, as is the case in the setting of TSCA, implies that it suffices that in the synthesised controller at least one such synchronisations actually occurs. This is precisely what is modelled with what we have called *semi-controllable* actions in this paper.

In the remainder of this section, we evaluate the expressiveness of the TSCA formalism by means of an example. We show that the encoding of an automaton $\mathscr{A}$ with semi-controllable actions into an automaton $\mathscr{A}'$ without, such that the same synthesised controllers are obtained, results in an exponential blow-up of the state space. More precisely, the encoding is intended to preserve safety: the most permissive controllers of $\mathscr{A}$ and $\mathscr{A}'$ are equals.

The automaton $\mathscr{A}'$ in Figure 10 sketches the encoding mentioned above. Automaton $\mathscr{A}'$ was obtained by turning all semi-controllable transitions of $\mathscr{A}$ into uncontrollable transitions in $\mathscr{A}'$. We now provide some intuition for this result. Intuitively, if the synchronisation on a specific semi-controllable action a occurs in n different transitions in $\mathscr{A}$ (two in our example), then the encoding creates an automaton that is the union of $2^n - 1$ automata (three in our example), which are obtained by all possible combinations of pruning a subset of the n semi-controllable transitions of $\mathscr{A}$, minus the one in which all n semi-controllable transitions are pruned. In fact, without knowing a priori the set of forbidden and successful configurations, it is impossible to provide a more efficient encoding. This can be seen as follows.

Assume, by contradiction, that there exists an encoding that results in a 'smaller' automaton $\mathscr{A}''$, in which one of the $2^n - 1$ combinations of pruned transitions (say, P) is discarded. It then suffices to specify as counterexample a property in $\mathscr{A}$ such that all source states of transitions in P are forbidden and all target states of the remaining semi-controllable transitions are successful. The synthesis of $\mathscr{A}$ against such a property would prune exactly the semi-controllable transitions in P. Thus, in the synthesis of $\mathscr{A}''$ such a most permissive controller would not be present.

6 Conclusions and Future Work

We have presented TSCA, a new formalism for specifying *service contracts* with *real-time constraints*, and a means to *synthesise safe orchestrations* in the presence of service requests with decreasing levels of criticality (viz. *urgent*, *greedy*, and *lazy*).

In compositions of TSCA, a match between a service offer and a service request has to satisfy the time constraints of

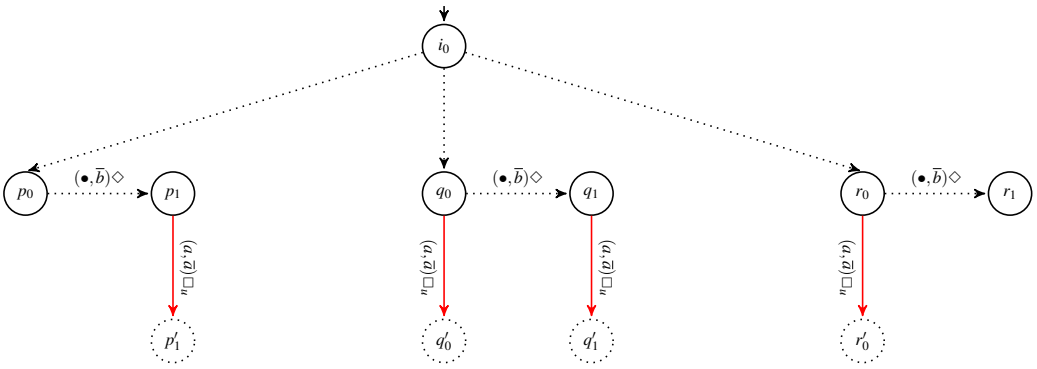

Fig. 10: Automaton $\mathscr{A}'$

both actions involved, otherwise the actions are interleaved. An agreement among service contracts is reached whenever all requests are matched by corresponding offers. To reach agreement, permitted service requests may be omitted if they are not satisfiable, while necessary service requests must be matched based on their level of criticality.

The synthesis of a safe TSCA orchestration is based on that of the most permissive controller from SCT and the concept of zones from timed games. The resulting synthesis can thus be considered both a safety and a reachability game, limited to finite executions. To properly deal with greedy and lazy requests, a novel type of *semi-controllable* action has been introduced in the orchestration synthesis. Such an action is basically a controllable action that may become uncontrollable in case specific non-local criteria are not met (i.e. the absence of agreement between requests and offers).

We plan to implement the presented theory in a prototypical tool by extending existing tools for (modal) service contract automata [8, 10, 12] and by reusing libraries from timed games that provide operations on zones [23, 28], to which our orchestration synthesis has been successfully reduced (cf. Theorem 2). Moreover, we would like to explore other types of requests than those currently available in TSCA. For instance, a service contract could declare a request to be necessary only if someone is willing to match it with a corresponding offer, otherwise it could renounce to such a request. This would be a service action that is either permitted or necessary in case it is a request or a match, respectively. Finally, we would also like to equip the formalism with weighted actions to be able to specify, e.g., the prices of hotel rooms or how much clients are willing to pay for their room. This would require to revisit the formalisation of service contracts and the synthesis of safe orchestrations, as well as the notion of agreement.

Acknowledgements We would like to thank Louis-Marie Traonouez for his contribution to this paper's conference publication. We would also like to thank the audience at VECoS 2019 for interesting questions and comments, which have led to the introduction of state invariants in our formalism, as presented in this paper. Finally, we would like to thank the anonymous reviewers for comments and suggestions that have improved the paper.

References

1. de Alfaro L, Henzinger TA (2001) Interface Automata. In: Proceedings 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'01), ACM, pp 109–120, DOI 10.1145/503209.503226
2. Alur R, Dill DL (1994) A Theory of Timed Automata. Theoret Comput Sci 126(2):183–235, DOI 10.1016/0304-3975(94)90010-8
3. Asarin E, Maler O, Pnueli A, Sifakis J (1998) Controller Synthesis for Timed Automata. IFAC Proceedings Volumes 31(18):447–452, DOI 10.1016/S1474-6670(17)42032-5
4. Azzopardi S, Pace GJ, Schapachnik F, Schneider G (2016) Contract automata: An operational view of contracts between interactive parties. Artif Intell Law 24(3):203–243, DOI 10.1007/s10506-016-9185-2
5. Bartoletti M, Cimoli T, Zunino R (2015) Compliance in Behavioural Contracts: A Brief Survey. In: Bodei C, Ferrari GL, Priami C (eds) Programming Languages with Applications to Biology and Security, Springer, LNCS, vol 9465, pp 103–121, DOI 10.1007/978-3-319-25527-9_9
6. Basile D, Degano P, Ferrari GL (2014) A formal framework for secure and complying services. J Supercomput 69(1):43–52, DOI 10.1007/s11227-014-1211-0
7. Basile D, Degano P, Ferrari GL (2016) Automata for Specifying and Orchestrating Service Contracts. Log Meth Comput Sci 12(4:6):1–51, DOI 10.2168/LMCS-12(4:6)2016
8. Basile D, Degano P, Ferrari GL, Tuosto E (2016) Playing with Our CAT and Communication-Centric Applications. In: Albert E, Lanese I (eds) Proceedings 36th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE'16), Springer, LNCS, vol 9688, pp 62–73, DOI 10.1007/978-3-319-39570-8_5
9. Basile D, ter Beek MH, Di Giandomenico F, Gnesi S (2017) Orchestration of Dynamic Service Product Lines with Featured Modal Contract Automata. In: Proceedings 21st International Systems and Software Product Line Conference (SPLC'17), ACM, vol 2, pp 117–122, DOI 10.1145/3109729.3109741
10. Basile D, Di Giandomenico F, Gnesi S (2017) FMCAT: Supporting Dynamic Service-based Product Lines. In: Proceedings 21st International Systems and Software Product Line Conference (SPLC'17), ACM, vol 2, pp 3–8, DOI 10.1145/3109729.3109760
11. Basile D, Di Giandomenico F, Gnesi S, Degano P, Ferrari GL (2017) Specifying Variability in Service Contracts. In: Proceedings 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'17), ACM, pp 20–27, DOI 10.1145/3023956.3023965
12. Basile D, ter Beek MH, Gnesi S (2018) Modelling and Analysis with Featured Modal Contract Automata. In: Proceedings 22nd International Systems and Software Product Line Conference (SPLC'18), ACM, vol 2, pp 11–16, DOI 10.1145/3236405.3236408
13. Basile D, ter Beek MH, Legay A, Traonouez LM (2018) Orchestration Synthesis for Real-Time Service Contracts. In: Atig MF, Bensalem S, Bliudze S, Monsuez

B (eds) Proceedings 12th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS'18), Springer, LNCS, vol 11181, pp 31–47, DOI 10.1007/978-3-030-00359-3_3
14. ter Beek MH, Bucchiarone A, Gnesi S (2007) Web Service Composition Approaches: From Industrial Standards to Formal Methods. In: Proceedings 2nd International Conference on Internet and Web Applications and Services (ICIW'07), IEEE, DOI 10.1109/ICIW.2007.71
15. ter Beek MH, Fantechi A, Gnesi S, Mazzanti F (2016) Modelling and analysing variability in product families: Model checking of modal transition systems with variability constraints. J Log Algebr Meth Program 85(2):287–315, DOI 10.1016/j.jlamp.2015.11.006
16. Behrmann G, David A, Larsen KG, Håkansson J, Pettersson P, Yi W, Hendriks M (2006) UPPAAL 4.0. In: Proceedings 3rd International Conference on the Quantitative Evaluation of SysTems (QEST'06), IEEE, pp 125–126, DOI 10.1109/QEST.2006.59
17. Behrmann G, Cougnard A, David A, Fleury E, Larsen KG, Lime D (2007) UPPAAL-Tiga: Time for Playing Games! In: Damm W, Hermanns H (eds) Proceedings 19th International Conference on Computer Aided Verification (CAV'07), Springer, LNCS, vol 4590, pp 121–125, DOI 10.1007/978-3-540-73368-3_14
18. Bouguettaya A, Singh M, Huhns M, Sheng QZ, Dong H, Yu Q, Neiat AG, Mistry S, Benatallah B, Medjahed B, Ouzzani M, Casati F, Liu X, Wang H, Georgakopoulos D, Chen L, Nepal S, Malik Z, Erradi A, Wang Y, Blake B, Dustdar S, Leymann F, Papazoglou M (2017) A Service Computing Manifesto: The Next 10 Years. Commun ACM 60(4):64–72, DOI 10.1145/2983528
19. Bouyer P, Markey N, Sankur O (2012) Robust Reachability in Timed Automata: A Game-Based Approach. In: Czumaj A, Mehlhorn K, Pitts AM, Wattenhofer R (eds) Proceedings 39th International Colloquium on Automata, Languages, and Programming (ICALP'12), Springer, LNCS, vol 7392, pp 128–140, DOI 10.1007/978-3-642-31585-5
20. Cassandras CG, Lafortune S (2006) Introduction to Discrete Event Systems. Springer, New York, NY, USA, DOI 10.1007/978-0-387-68612-7
21. Cassez F, David A, Fleury E, Larsen KG, Lime D (2005) Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In: Abadi M, de Alfaro L (eds) Proceedings 16th International Conference on Concurrency Theory (CONCUR'05), Springer, LNCS, vol 3653, pp 66–80, DOI 10.1007/11539452_9
22. David A, Larsen KG, Legay A, Nyman U, Wąsowski A (2010) Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In: Proceedings 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10), ACM, pp 91–100, DOI 10.1145/1755952.1755967
23. David A *et al* (2017) UPPAAL DBM Library. URL `http://people.cs.aau.dk/~adavid/UDBM/`
24. Georgakopoulos D, Papazoglou MP (eds) (2008) Service-oriented Computing. MIT Press, Cambridge, MA, USA, URL `https://mitpress.mit.edu/books/service-oriented-computing`
25. Hüttel H, Lanese I, Vasconcelos VT, Caires L, Carbone M, Deniélou PM, Mostrous D, Padovani L, Ravara A, Tuosto E, Torres Vieira H, Zavattaro G (2016) Foundations of Session Types and Behavioural Contracts. ACM Comput Surv 49(1):3:1–3:36, DOI 10.1145/2873052
26. Křetínský J (2017) 30 Years of Modal Transition Systems: Survey of Extensions and Analysis. In: Aceto L, Bacci G, Bacci G, Ingólfsdóttir A, Legay A, Mardare R (eds) Models, Algorithms, Logics and Tools, LNCS, vol 10460, Springer, pp 36–74, DOI 10.1007/978-3-319-63121-9_3
27. Larsen KG, Nyman U, Wąsowski A (2007) Modal I/O Automata for Interface and Product Line Theories. In: De Nicola R (ed) Proceedings 16th European Symposium on Programming (ESOP'07), Springer, LNCS, vol 4421, pp 64–79, DOI 10.1007/978-3-540-71316-6_6
28. Legay A, Traonouez LM (2013) PyEcdar: Towards Open Source Implementation for Timed Systems. In: Hung DV, Ogawa M (eds) Proceedings 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13), Springer, LNCS, vol 8172, pp 460–463, DOI 10.1007/978-3-319-02444-8_35, URL `https://project.inria.fr/pyecdar/`
29. Lynch NA, Tuttle MR (1989) An Introduction to Input/Output Automata. CWI Quarterly 2(3):219–246, URL `https://ir.cwi.nl/pub/18164/18164A.pdf`, also available as MIT Technical Memo MIT/LCS/TM-373
30. Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, New York, NY, USA
31. Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230, DOI 10.1137/0325013
32. Yi Q, Liu X, Bouguettaya A, Medjahed B (2008) Deploying and managing Web services: issues, solutions, and directions. VLDB J 17(3):735–572, DOI 10.1007/s00778-006-0020-3