Orchestration Synthesis for Real-time Service Contracts

Davide Basile[1,2], Maurice H. ter Beek[2], Axel Legay[3], and Louis-Marie Traonouez[3]

[1] University of Florence, Italy
[2] ISTI–CNR, Pisa, Italy
{davide.basile,maurice.terbeek}@isti.cnr.it
[3] Inria Rennes, France
{axel.legay,louis-marie.traonouez}@inria.fr

Abstract. Service contracts offer a way to define the desired behavioural compliance of a composition of services, characterised by the fulfilment of all requirements (e.g. service requests) by obligations (e.g. service offers). Depending on their granularity, requirements may vary according to their criticality and contain real-time aspects (e.g. service expiration time). Synthesis of safe orchestrations, the standard method to refine spurious compositions into compliant ones, is of paramount importance. Ideally, safe orchestrations solve competition among matching requests/offers, respecting criticalities and time constraints, in the best possible way.
The contribution of this paper is (i) the introduction of timed service contract automata, a novel formalisation of service contracts with (ii) real-time constraints and (iii) service requests with varying levels of criticality, and a means to compute their (iv) composition and (v) safe orchestration. Orchestration is based on the synthesis of the most permissive controller from supervisory control theory, computed using the concept of zones from timed games. An intuitive example illustrates the contribution.

1 Introduction

Service computing is concerned with the creation, publication, discovery and orchestration of services [1]. A typical application is an orchestration of services created and published by different organisations that are dynamically discovered. In the recent Service Computing Manifesto [2], *service design* is listed as one of the four emerging research challenges in service computing for the next 10 years.

Formal models of service contracts are surveyed in [3]. These offer specification frameworks to formalise the externally observable behaviour of services in terms of obligations (i.e. *offers*) and requirements (i.e. *requests*) to be matched. Contracts that are fulfilled characterise *agreement* among services as an *orchestration* (i.e. composition) 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 [4].

In this paper, we include notions of time in one such model, viz. (service) contract automata [5]. Such an 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 offers 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. The (verifiable) notion of *agreement* characterises safe executions of services (i.e. all requests are matched by corresponding offers).

In [6], service contract automata were equipped with modalities distinguishing *necessary* and *permitted* requests to mimick the uncontrollable and controllable actions, respectively, as known from Supervisory Control Theory (SCT) [7].

Contribution. We introduce *timed service contract automata* (TSCA) by endowing service contract automata with *real-time constraints*. TSCA also allow to specify different types of necessary requests, called *urgent*, *greedy* and *lazy*, with decreasing levels of criticality as in [8], which 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 handle this in a synthesis algorithm for TSCA, a 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. Our synthesis algorithm mixes and extends techniques from SCT with notions from timed games [9, 10].

A TSCA orchestration thus solves multi-party competitions on service actions and on the associated timing constraints, a natural scenario in service computing. 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 becoming unsafe. This neatly delimits the fragments (i.e. executions) of service compositions allowed in safe orchestrations (cf. Fig. 4 discussed in Sect. 4). By changing the timing constraints or criticality levels, designers can fine-tune such fragments according to their specific needs.

We summarise our contribution: (i) we introduce TSCA, a new formalisation of service contracts with (ii) real-time constraints and (iii) service requests with varying criticality levels, and a means to compute TSCA (iv) composition and (v) safe orchestration. We are not aware of other formalisms for service contracts or component-based software engineering with native support for these features. We illustrate its functioning with a TSCA model of a Hotel reservation system.

Related Work. Formalisms for service contracts and session types are surveyed in [11]: all of them lack an explicit notion of time and different levels of criticality.

Component-based formalisms like *Interface automata* [12] and *(timed) (I/O) automata* [13–15] cannot model contracts that compete for the same service offer or request, a key feature of TSCA, and also do not allow different criticality levels. Modal I/O automata [16] distinguish may and must modalities, thus admitting some actions to be more critical than others, but the other differences remain. The accidentally homonym *contract automata* of [17] were introduced to model generic natural language legal contracts between two parties: they are not compositional and do not focus on synthesising orchestrations of services in agreement.

Finally, the synthesis algorithm for TSCA (introduced in Sect. 3) resembles a *timed game*, but differs from classical timed game algorithms [9, 10]: it 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 is the maximal winning strategy, which is computable since only finite traces are allowed [18] and all services terminate by definition. The orchestrator enforces only fair executions.

2 Modelling Real-time Service Contracts

Contract automata were introduced to describe and compose service contracts [5]. 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. The complement of a finite set S is denoted by $\overline{S}$; the empty set by $\varnothing$. For a vector $\boldsymbol{v} = (e_1, \dots, e_n)$ of *rank* $n \geq 1$, denoted by r_v, its ith element is denoted $\boldsymbol{v}_{(i)}$, $1 \leq i \leq r_v$. Concatenation of m vectors $\boldsymbol{v}_i$ is denoted by $\boldsymbol{v}_1 \cdots \boldsymbol{v}_m$.

The set of basic actions of a contract automaton is defined as $\Sigma = \mathsf{R} \cup \mathsf{O} \cup \{\bullet\}$, where $\mathsf{R} = \{a, b, \dots\}$ is the set of *requests*, $\mathsf{O} = \{\overline{a}, \overline{b}, \dots\}$ 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 $\boldsymbol{a}$ 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 $\boldsymbol{a}_{(i)}$ is an offer and $\boldsymbol{a}_{(j)}$ is the complementary request or vice versa; all the other entries of $\boldsymbol{a}$ contain the symbol $\bullet$ (meaning that the corresponding principals remain idle). Let $\bullet^m$ denote a vector $(\bullet, \dots, \bullet)$ of rank m.

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

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

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

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

In [6], the contract automata of [5] were equipped with action variability via *necessary* ($\Box$) and *permitted* ($\Diamond$) modalities that can be used to classify requests (and matches), while all offers are by definition permitted. Permitted requests and offers reflect optional behaviour and can thus be discarded in compositions.

Table 1: Classification of (basic) actions of timed service contract automata

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

2.1 Timed Service Contract Automata

In this paper, the set of necessary requests of the service contract automata of [6] is partitioned in *urgent*, *greedy* and *lazy* requests as in [8]. These must be matched to reach agreement, thus adding a layer of 'timed' variability: a means to specify 'when' certain (service) requests must be matched in a composition (contract). Table 1 depicts the different types of actions considered in this paper.

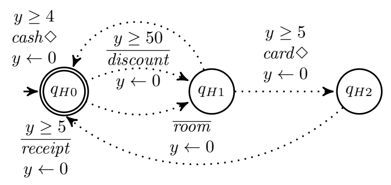
We borrow notation concerning clocks from [10]. Let X be a finite set of real-valued variables called clocks. Let $C(X)$ denote the set of constraints φ generated by the grammar $\varphi ::= x \sim k \mid x - y \sim k \mid \varphi \wedge \varphi$, where $k \in \mathbb{Z}$, $x, y \in X$ and $\sim \in \{<, \leq, =, >, \geq\}$. 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 $Y \subseteq X$, let $v[Y]$ denote 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 $\mathcal{A}$ (TSCA for short) of rank $n \geq 1$ is a tuple $\langle Q, \boldsymbol{q_0}, A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell}, A^o, X, T, F \rangle$, in which*

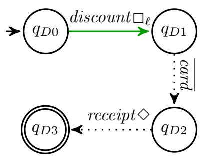
- *$Q = Q_1 \times \cdots \times Q_n$ is the product of finite sets of states*
- *$\boldsymbol{q_0} \in Q$ is the initial state*
- *$A^{\Diamond}, A^{\Box_u}, A^{\Box_g}, A^{\Box_\ell} \subseteq \mathsf{R}$ are (pairwise disjoint) sets of permitted, urgent, greedy and lazy requests, respectively, and 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 $t = (\boldsymbol{q}, g, \boldsymbol{a}, Y, \boldsymbol{q'}) \in T$, the following holds:*
  - *$\boldsymbol{a}$ is either a request or an offer or a match*
  - *$\forall i \in 1 \ldots n : \boldsymbol{a}_{(i)} = \bullet$ implies $\boldsymbol{q}_{(i)} = \boldsymbol{q'}_{(i)}$*
  - *$t \in T^{\Diamond}$ iff $\boldsymbol{a}$ is either a request or a match on $a \in A^{\Diamond}$ or an offer on $\overline{a} \in A^o$; otherwise $t \in T^{\Box}$*
- *$F \subseteq Q$ is the set of final states*

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

For brevity, unless stated differently, in the sequel we assume a fixed TSCA $\mathcal{A} = \langle Q_{\mathcal{A}}, \boldsymbol{q_0}_{\mathcal{A}}, A^{\Diamond}_{\mathcal{A}}, A^{\Box_u}_{\mathcal{A}}, A^{\Box_g}_{\mathcal{A}}, A^{\Box_\ell}_{\mathcal{A}}, A^o_{\mathcal{A}}, X_{\mathcal{A}}, T_{\mathcal{A}}, F_{\mathcal{A}} \rangle$ of rank n. Subscript $\mathcal{A}$ 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, offer or match, if its label is such.



(a) Hotel



(b) DiscountClient

Fig. 1: TSCA: (a) hotel booking system; (b) discount client

Pictorially, offer actions are overlined while request actions are not. Moreover, permitted actions label dotted transitions and are suffixed by $\Diamond$, whereas urgent, greedy and lazy necessary actions label red, orange and green transitions and are suffixed by $\Box_u$, $\Box_g$ and $\Box_\ell$, respectively (cf. Table 1).[4]

Example 1. Fig. 1 shows two TSCA. The one in Fig. 1a depicts a hotel booking system offering two room types (normal and discount) and requests payment from clients. The discount room is only available upon waiting at least 50 time units (t.u. for short). Then the hotel requests payment, either in cash (which takes at least 4 t.u.) or by card (at least 5 t.u.). In the latter case only, the hotel offers a receipt after at least 5 t.u. The TSCA in Fig. 1b depicts a hotel client, who requests a discount room, offers to pay by card and requests a receipt.

2.2 Semantics

A TSCA recognises a trace language over actions and their modalities. Let $\mathcal{A}$ 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 tuple $(w, \boldsymbol{q}, v) \in (A \cup \{\bigcirc\})^* \times Q \times \mathbb{R}^X_{\geq 0}$ consisting of a recognised trace, a state and a valuation of clocks. Recognised traces are such that from a configuration $(w, \boldsymbol{q}, v)$, 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 $(w, \boldsymbol{q}, v) \xrightarrow{\boldsymbol{a}\bigcirc} (w', \boldsymbol{q}', v')$ is fired iff $w = \boldsymbol{a}\bigcirc\, w'$ and $(\boldsymbol{q}, g, \boldsymbol{a}, Y, \boldsymbol{q}') \in T^{\bigcirc}$, where $v \models g$ and $v' = v[Y]$ or else, for some $\delta \geq 0$, we have $(w, \boldsymbol{q}, v) \xrightarrow{\delta} (w, \boldsymbol{q}, v')$ if $v' = v + \delta$. Time progress δ is a silent action in languages recognised by TSCA.

The semantics of a TSCA $\mathcal{A}$ is a labelled transition system $TS_{\mathcal{A}} = (\mathbb{C}, c_0, \rightarrow)$, where $\mathbb{C} = (A \cup \{\bigcirc\})^* \times Q \times \mathbb{R}^X_{\geq 0}$ is the set of configurations, $c_0 = (w, \boldsymbol{q}_0, \boldsymbol{0})$ is the initial configuration, for some $w \in (A \cup \{\bigcirc\})^*$, and the set of transition labels is $(A\{\bigcirc\}) \cup \mathbb{R}_{\geq 0}$. A *run* of $\mathcal{A}$ 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, modalities can be attached to basic actions or to their action vector (e.g. $(a\Box_\ell, \overline{a}) \equiv (a, \overline{a})\Box_\ell$). We may write $(\boldsymbol{q}, v)$ whenever w is immaterial, $(\boldsymbol{q}, v) \xrightarrow{\boldsymbol{a}\bigcirc}$ whenever $(\boldsymbol{q}', v')$ is immaterial and $(w, \boldsymbol{q}, v) \rightarrow (w', \boldsymbol{q}', v')$ whenever $\boldsymbol{a}\bigcirc$ or δ are immaterial. Let $\rightarrow^*$ denote the reflexive and transitive closure of $\rightarrow$. The language of $\mathcal{A}$ is $\mathscr{L}(\mathcal{A}) = \{\, w \mid (w, \boldsymbol{q_0}, \boldsymbol{0}) \rightarrow^* (\varepsilon, \boldsymbol{q}, v),\ \boldsymbol{q} \in F\,\}$.

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

Behavioural analysis is based on exploring a (finite) *simulation graph*, whose nodes are *symbolic configurations*, defined as pairs $(\boldsymbol{q}, Z)$, where $\boldsymbol{q} \in Q$ and Z is a zone of $\mathbb{R}^X_{\geq 0}$. Let $C \subseteq \mathbb{C}$ be a set of configurations and let $\boldsymbol{a} \in A$. Then we define the $\boldsymbol{a}$-successor of X by $Post_{\mathcal{A},\boldsymbol{a}}(C) = \{\, c' \mid \exists\, c \in C : c \xrightarrow{\boldsymbol{a}\circ} c' \,\}$ and the $\boldsymbol{a}$-predecessor $Pred_{\mathcal{A},\boldsymbol{a}}(C) = \{\, c \mid \exists\, c' \in C : c \xrightarrow{\boldsymbol{a}\circ} c' \,\}$. We moreover define the *match/offer predecessor* as $moPred_{\mathcal{A}}(C) = \bigcup_{\boldsymbol{a}\ match\ or\ offer} Pred_{\mathcal{A},\boldsymbol{a}}(C)$.

The timed successors and timed predecessors of C are defined by $C^{\nearrow} = \{\, (\boldsymbol{q}, v + \delta) \mid (\boldsymbol{q}, v) \in C,\ \delta \in \mathbb{R}_{\geq 0} \,\}$ and $C^{\swarrow} = \{\, (\boldsymbol{q}, v - \delta) \mid (\boldsymbol{q}, v) \in C,\ \delta \in \mathbb{R}_{\geq 0} \,\}$, respectively. Let $\rightarrow$ be the transition relation defined on symbolic configurations by $(\boldsymbol{q}, Z) \xrightarrow{\boldsymbol{a}\bigcirc} (\boldsymbol{q}', Z')$ if $(\boldsymbol{q}, g, \boldsymbol{a}, Y, \boldsymbol{q}') \in T^{\bigcirc}$ and $Z' = ((Z \cap [g])[Y])^{\nearrow}$.

2.3 Composition

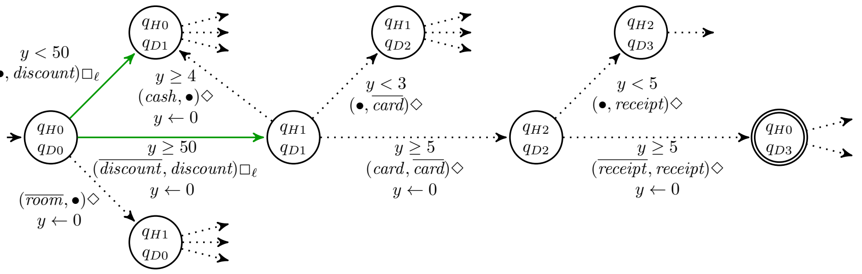
A set of TSCA is *composable* iff their sets of clocks are pairwise disjoint.

Definition 3 (Composable TSCA). *A set* $\{\, \mathcal{A}_i \mid i \in 1 \ldots n \,\}$ *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 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, then only their match is allowed wheras 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* $\mathcal{A}_i$ *be composable TSCA of rank* r_i, $i \in 1 \ldots n$. *The* composition $\bigotimes_{i \in 1 \ldots n} \mathcal{A}_i$ *is the TSCA* $\mathcal{A}$ *of rank* $m = \sum_{i \in 1 \ldots n} r_i$, *where*

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



q_{H0} q_{D0}
q_{H0} q_{D1}
q_{H1} q_{D0}
q_{H1} q_{D1}
q_{H1} q_{D2}
q_{H2} q_{D2}
q_{H2} q_{D3}
q_{H0} q_{D3}
$y < 50$ $(\bullet, discount)\Box_\ell$
$y \geq 4$ $(cash, \bullet)\Diamond$ $y \leftarrow 0$
$y \geq 50$ $(\overline{discount}, discount)\Box_\ell$ $y \leftarrow 0$
$(\overline{room}, \bullet)\Diamond$ $y \leftarrow 0$
$y < 3$ $(\bullet, \overline{card})\Diamond$
$y \geq 5$ $(card, \overline{card})\Diamond$ $y \leftarrow 0$
$y < 5$ $(\bullet, receipt)\Diamond$
$y \geq 5$ $(\overline{receipt}, receipt)\Diamond$ $y \leftarrow 0$

Fig. 2: Excerpt of composition $\mathsf{Hotel} \otimes \mathsf{DiscountClient}$ of the two TSCA in Fig. 1

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 $\boldsymbol{a}_i \bowtie \boldsymbol{a}_j$. If $(\boldsymbol{q}_j, g_j, \boldsymbol{a}_j, Y_j, \boldsymbol{q}'_j) \in T^\Box$, then the resulting match transition is marked necessary (i.e. $(\boldsymbol{q}, g, \boldsymbol{a}, Y, \boldsymbol{q}') \in T^\Box$), with $g = g_i \wedge g_j$ the conjunction of the guards. If both $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ are permitted, then so is their resulting match transition t. All principals not involved in t remain idle. In case $\boldsymbol{a}_i \bowtie \boldsymbol{a}_j$ as before, but only one guard (i.e. either g_i or g_j) is satisfied, then only the interleaving is possible and 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 if no complementary actions can be executed from the composed source state (i.e. $\boldsymbol{q}$). Now one operand executes its transition $t = (\boldsymbol{q}_i, g_i, \boldsymbol{a}_i, Y_i, \boldsymbol{q}'_i)$ and all others remain idle: 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 $\boldsymbol{a}_i \bowtie \boldsymbol{a}_j$ excludes pre-existing match transitions of the operands to generate new matches.

Example 2. Fig. 2 shows excerpts of the TSCA composition of the hotel and client TSCA of Fig. 1. The more relevant part is depicted, viz. whose semantics is an orchestration (from initial to final state). Note that request $discount\,\Box_\ell$ can either be matched with the offer $\overline{discount}$ if $y \geq 50$ or not matched if $y < 50$.

2.4 Controllability

We now discuss the different types of actions of TSCA (cf. Table 1) in light of the orchestration synthesis algorithm we will present in Sect. 3. To begin with, we define dangling configurations, i.e. those that are either not reachable or from which no final state can be reached (i.e. not successful). The orchestration synthesis will be specified as a safety game, in which reachability of final states is satisfied through the dangling predicate. The definition makes use of a set C of 'bad' configurations that are not to be traversed. Recall that $\mathcal{A}$ is a fixed TSCA.

Definition 5 (Dangling configuration). *Let $\mathcal{A}$, $C \subseteq \mathbb{C}$ and $c = (\boldsymbol{q}, v) \in C$.*

We say that c is reachable *in $\mathcal{A}$ given C, denoted as $c \in Reachable_{\mathcal{A}}(C)$, iff $(\boldsymbol{q_0}, \boldsymbol{0}) \xrightarrow{w}{}^* c$ without traversing configurations $(\boldsymbol{q}_r, v_r) \in C$*

We say that c is successful *in $\mathcal{A}$ given C, denoted as $c \in Successful_{\mathcal{A}}(C)$, iff $c \xrightarrow{w}{}^* (\boldsymbol{q_f}, v') \in F$ without traversing configurations $(\boldsymbol{q}_r, v_r) \in C$*

The set of dangling configurations *in $\mathcal{A}$ given C is defined as $Dangling_{\mathcal{A}}(C) = \overline{Reachable_{\mathcal{A}}(C) \cap Successful_{\mathcal{A}}(C)}$*

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

Orchestration synthesis for (service) contract automata resembles that of the most permissive controller from SCT; in fact, [6] provided a correspondence between *controllable*/*uncontrollable* actions from SCT and permitted/necessary requests of contract automata. Intuitively, the aim of SCT is to synthesise a most permissive controller enforcing 'good' computations, i.e. runs reaching a final state without traversing any given *forbidden* state. To do so, SCT distinguishes *controllable* events (those the controller can disable) from *uncontrollable* events (those always enabled). Ideally, actions ruining a so-called safe orchestration of service contracts (a notion formally defined in Sect. 3, resembling a most permissive controller) should be removed by the synthesis algorithm. However, this can only be done for actions that are controllable in the orchestration.

We now characterise when a TSCA action (and the transition it labels) is (un)controllable. 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. It is also possible to require that a necessary action must be matched in at least one configuration from which it is fired. In the latter situation, it is possible to safely remove those requests (or matches) from the orchestration, as long as they 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 all offers are permitted. All permitted actions (offers, requests and matches) are fully controllable. 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, 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 is matched, it can no longer be disabled. In this case, 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 at least one match is available (i.e. lazy requests and lazy matches are semi-controllable).

Table 2: Controllability of request actions and match actions

action	**requests**	**matches**
urgent $\Box_u$	uncontrollable	uncontrollable
greedy $\Box_g$	semi-controllable	uncontrollable
lazy $\Box_\ell$	semi-controllable	semi-controllable
permitted $\Diamond$	controllable	controllable

In the rest 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. Note 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 TSCA $\mathcal{A}$ given C if there exists a (greedy or lazy) match transition t' in $\mathcal{A}$, 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.

Definition 6 (Semi-controllable transition). *Let $\mathcal{A}$ be a TSCA, let $C \subseteq \mathbb{C}$ and let $t = (\boldsymbol{q}_1, g_1, \boldsymbol{a}_1, Y_1, \boldsymbol{q}_1')$ be a transition of $\mathcal{A}$. Then 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 $\mathcal{A}$ given C.

We say that t is controllable *in $\mathcal{A}$ given C if $\exists\, t' = (\boldsymbol{q}_2, g_2, \boldsymbol{a}_2, Y_2, \boldsymbol{q}_2') \in T^{\Box}$, s.t. $\boldsymbol{a}_2$ is a match, $\exists\, v$ s.t. $(\boldsymbol{q}_2, v) \in \mathit{Reachable}_{\mathcal{A}}(C)$, $(\boldsymbol{q}_2', v') \in \mathit{Post}_{\mathcal{A},\boldsymbol{a}_2}((\boldsymbol{q}_2, v)^{\nearrow})$, $(\boldsymbol{q}_2', v') \in \mathit{Successful}_{\mathcal{A}}(C)$, $\boldsymbol{q_1}_{(i)} = \boldsymbol{q_2}_{(i)}$ and $\boldsymbol{a_1}_{(i)} = \boldsymbol{a_2}_{(i)} \in \mathsf{R} \cap (A^{\Box_g} \cup A^{\Box_\ell})$; otherwise t is* uncontrollable *in $\mathcal{A}$ given C.*

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

Example 3. In Fig. 2, all transitions are permitted, except for the lazy discount actions. The transition $(\bullet, \mathit{discount})\Box_\ell$ is thus a controllable lazy request, as the same request of DiscountClient is matched in the transition $(\overline{\mathit{discount}}, \mathit{discount})\Box_\ell$. In the resulting orchestration (cf. Sect. 4) this will be the only match available for such a necessary action.

We 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 synthesis of safe orchestrations of TSCA, considering both timing constraints and service requests with different levels of criticality. We carefully adapt the synthesis algorithm for (modal) service contract automata defined in [6], which was based on the synthesis of the most permissive controller from SCT. To respect the timing constraints, the synthesis algorithm of TSCA presented below is computed using the notion of zones from timed games [9, 10].

The algorithm we will propose differs from the ones presented in [9, 10] 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 framework, marked states are the final states of the composition of contracts, whereas bad states are those states that spoil an agreement among contracts (cf. Definitions 7 and 9 below).

We recall *(modal) agreement* and *safety* on languages of service contract automata [6]. 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.

Definition 7 (Agreement, safety). *Let $\mathcal{A}$ be a TSCA. A trace accepted by $\mathcal{A}$ is in* agreement *if it belongs to the set*

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

$\mathcal{A}$ is safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$; else* unsafe. *If $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \varnothing$, then $\mathcal{A}$* 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 evaluations 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: a strategy avoiding dangling configurations.

Definition 8 (Strategy). *Let $\mathcal{A}$ be a TSCA. 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 to actions or delays s.t. given $(\boldsymbol{q_0}, \boldsymbol{0}) \xrightarrow{w}{}^{*} (\boldsymbol{q}, v)$, then $(\boldsymbol{q}, v) \xrightarrow{\lambda} (\boldsymbol{q}', v')$, for some $\lambda \in f(w)$, $(\boldsymbol{q}', v') \in \mathbb{C}$.*

Furthermore, f is well-formed *given $C \subseteq \mathbb{C}$ if never $(\boldsymbol{q}', v') \in Dangling_{\mathcal{A}}(C)$. The language recognised by $\mathcal{A}$ following the strategy f is denoted by $\mathscr{L}_f(\mathcal{A})$ and $f^{\mathbb{C}}$ denotes the strategy allowing to traverse all and only configurations in $\mathbb{C}$.*

We discuss further differences compared to timed games. A TSCA game can be seen as a 2-player game. A controller (i.e. orchestrator) fires controllable transitions to enforce agreement among contracts. An opponent fires 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, as traces recognised by TSCA languages are finite.

In timed games, strategies cannot prevent uncontrollable transitions from being fired. 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 fired without a corresponding offer (i.e. no match). In such configurations, the controller loses: 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 $\mathcal{A}$ be a TSCA and let $C \subseteq \mathbb{C}$. A transition $t = \boldsymbol{q}\xrightarrow{\boldsymbol{a}} \in T_{\mathcal{A}}$ is* forced *in a configuration $(\boldsymbol{q}, v)$ given C iff $(\boldsymbol{q}, v)\xrightarrow{\boldsymbol{a}}$ and (i) t is uncontrollable in $\mathcal{A}$ given C or (ii) $\boldsymbol{q} \notin F$ and no other $t' = \boldsymbol{q}\xrightarrow{\boldsymbol{a}'} \in T_{\mathcal{A}}$ is s.t. $(\boldsymbol{q}, v)\xrightarrow{\delta}(\boldsymbol{q}, v')\xrightarrow{\boldsymbol{a}'}$ for some delay δ.*

A configuration $(\boldsymbol{q}, v_1) \notin Dangling_{\mathcal{A}}(C)$ is in uncontrollable disagreement *in $\mathcal{A}$ given C iff $(\boldsymbol{q}, v_1)\xrightarrow{w}{}^*(\boldsymbol{q}_1, v_2)$ s.t. only timed or forced transitions are enabled and either (i) $w \notin \mathfrak{A}$ or (ii) some configuration in C was traversed or (iii) $\nexists\, w' \in \mathfrak{A}$ s.t. $(\boldsymbol{q}_1, v_2)\xrightarrow{w'}{}^*(\boldsymbol{q}_f, v_3)$ with $\boldsymbol{q}_f \in F_{\mathcal{A}}$ without traversing configurations in C.*

A safe orchestration of TSCA can be interpreted as a *winning strategy* in terms of timed games, and it is defined below. Basically, a winning strategy enforces agreement among contracts: no bad configurations will ever be traversed.

Definition 10 (Winning strategy). *Let $\mathcal{A}$ be a TSCA, f be a strategy given $C \subseteq \mathbb{C}$ and U its set of configurations in uncontrollable disagreement in $\mathcal{A}$ given C. Then f is a* winning strategy *given C if it is well-formed given C, it never traverses configurations in U and $\mathscr{L}_f(\mathcal{A}) \subseteq \mathfrak{A}$. A winning strategy f given C is* maximal *if there is no winning strategy f' given C s.t. $\mathscr{L}_f(\mathcal{A}) \subseteq \mathscr{L}_{f'}(\mathcal{A})$.*

Before defining the synthesis of a safe orchestration, we introduce some useful notions. Given a set of configurations $C \subseteq \mathbb{C}$ of a TSCA $\mathcal{A}$, the *uncontrollable predecessor* predicate $uPred_{\mathcal{A}}(C)$ is defined as all configurations from which some configuration in C is reachable by firing an uncontrollable transition. Formally:

$$uPred_{\mathcal{A}}(C) = \{\, c \mid \exists c' \in C,\ c\xrightarrow{\boldsymbol{a}\square}c' \text{ uncontrollable in } \mathcal{A} \text{ given } C \,\}$$

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 [10]. Intuitively, a configuration c is in $Pred_{\mathcal{A},t}(C_1, C_2)$ 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:

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

$$\text{where } Post_{\mathcal{A},[0,\delta]}(c) = \{\, c' \in \mathbb{C} \mid \exists\, t \in [0, \delta] \text{ s.t. } c\xrightarrow{t}c' \,\} \text{ and } \overline{C_2} = \mathbb{C} \setminus C_2$$

We can now specify the synthesis of a safe orchestration of TSCA. Let $\hat{\mathcal{A}}$ denote the TSCA obtained from $\mathcal{A}$ by replacing $T_{\mathcal{A}}$ with $T_{\hat{\mathcal{A}}} = \{\, t = \boldsymbol{q}\xrightarrow{\boldsymbol{a}\bigcirc} \mid t \in T_{\mathcal{A}}$ and $(\bigcirc \neq \Diamond \vee \boldsymbol{a}$ not request$)\,\}$, i.e. all permitted requests are pruned from $\mathcal{A}$.

Definition 11 (Safe orchestration synthesis). *Let $\mathcal{A}$ be a TSCA and let $\phi : 2^{\mathbb{C}} \to 2^{\mathbb{C}}$ be a monotone function on the cpo $(2^{\mathbb{C}}, \subseteq)$ s.t. $\phi(\mathbb{C}_{i-1}) = \mathbb{C}_i$, where $\mathbb{C}_0 = \{\, c \mid c \in \mathbb{C},\ c\xrightarrow{\boldsymbol{a}\square},\ \boldsymbol{a}$ uncontrollable request in $\hat{\mathcal{A}}$ given $\varnothing \,\}$ and*

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

Finally, let $\mathbb{C}^* = sup(\{\, \phi^n(\mathbb{C}_0) \mid n \in \mathbb{N} \,\})$ be the least fixed point of ϕ. Then the safe orchestration *of $\mathcal{A}$ is the strategy:*

$$f^* = \begin{cases} \bot & \text{if } (\boldsymbol{q}_0, \boldsymbol{0}) \in \mathbb{C}^* \\ f^{\overline{\mathbb{C}^*}} & \text{otherwise} \end{cases}$$

This definition is such that whenever the initial configuration belongs to $\mathbb{C}^*$, then the orchestration is *empty*: no strategy exists to enforce agreement among contracts while avoiding configurations in uncontrollable disagreement. Otherwise, $\overline{\mathbb{C}}^*$ identifies a winning strategy characterising a safe orchestration of contracts. This strategy allows as many transitions as possible without traversing configurations in $\mathbb{C}^*$. The controller can avoid principals reaching bad configurations in $\mathbb{C}^*$, while 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 in $\mathbb{C}^*$ (recall f^* is not a function). Note that f^* is computable due to finiteness of the symbolic configurations and monotonicity of the fixed-point computation [10]: it is the maximal well-formed winning strategy.

Theorem 1 (Maximal winning strategy). *Let $\mathcal{A}$ be a TSCA and f^* be the strategy 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 $\mathcal{A}$ given $\mathbb{C}^*$.*

Example 4. Recall the composition $\mathsf{Hotel} \otimes \mathsf{DiscountClient}$ in Fig. 2. We can apply the synthesis algorithm to compute its safe orchestration f^*. In f^*, the request transition $(\bullet, \mathit{discount}\Box_\ell)$ is removed because it is controllable (cf. Example 3). The language recognised by f^* is the singleton $\mathscr{L}_{f^*}(\mathsf{Hotel} \otimes \mathsf{DiscountClient}) = \{(\overline{\mathit{discount}}, \mathit{discount})\Box_\ell(\mathit{card}, \overline{\mathit{card}})\Diamond(\overline{\mathit{receipt}}, \mathit{receipt})\Diamond\}$.

In [10] (Theorem 4) the computation of $\mathit{Pred}_{\mathcal{A},t}$ is reduced to the following basic operations on zones: $\mathit{Pred}_{\mathcal{A},t}(C_1, C_2) = (C_1^{\swarrow} \setminus C_2^{\swarrow}) \cup ((C_1 \cap C_2^{\swarrow}) \setminus C_2)^{\swarrow}$. Similarly, we now provide procedures for computing the newly introduced sets $\mathit{moPred}_{\mathcal{A}}$, $\mathit{uPred}_{\mathcal{A}}$ and $\mathit{Dangling}_{\mathcal{A}}$ using basic operations on zones. Together these provide an effective procedure for computing $\mathbb{C}^*$ (hence a safe orchestration). The set $\mathit{moPred}_{\mathcal{A}}$ can be computed from $\mathit{Pred}_{\mathcal{A}}$ by only considering discrete steps that are not requests. Conversely, both $\mathit{uPred}_{\mathcal{A}}$ and $\mathit{Dangling}_{\mathcal{A}}$ require visiting the symbolic configurations of $\mathcal{A}$, and can be computed as follows.

Theorem 2 (Compute dangling configuration). *Let $\mathcal{A}$ be a TSCA, $C \subseteq \mathbb{C}$ and ϕ be as 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* reachable configurations *in $\mathcal{A}$ given C are computed as $\mathit{Reachable}_{\mathcal{A}}(C) = \mathbb{C}^*$, where $\mathbb{C}_0 = (\boldsymbol{q}_0, \boldsymbol{0})^{\nearrow} \setminus C^{\nearrow}$ and $\mathbb{C}_i = \bigcup_{\boldsymbol{a}}(\mathit{Post}_{\mathcal{A},\boldsymbol{a}}(\mathbb{C}_{i-1})^{\nearrow} \setminus C^{\nearrow}) \cup \mathbb{C}_{i-1}$*
2. *The* successful configurations *in $\mathcal{A}$ given C are computed as $\mathit{Successful}_{\mathcal{A}}(C) = \mathbb{C}^*$, where $\mathbb{C}_0 = \{\, (\boldsymbol{q}_f, v) \mid \boldsymbol{q}_f \in F_{\mathcal{A}}$ and $v \in \mathbb{R}_{\geq 0}^{X_{\mathcal{A}}} \,\} \setminus C$ and $\mathbb{C}_i = \mathit{Pred}_{\mathcal{A},t}(\mathbb{C}_{i-1} \cup (\mathit{Pred}_{\mathcal{A}}(\mathbb{C}_{i-1}) \setminus C), C) \cup \mathbb{C}_{i-1}$*
3. *The* dangling configurations *in $\mathcal{A}$ given C are computed as $\mathit{Dangling}_{\mathcal{A}}(C) = \overline{\mathit{Successful}_{\mathcal{A}}(C \cup \overline{\mathit{Reachable}_{\mathcal{A}}(C)})}$*

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): it is then possible to ignore unreachable successful configurations. We thus determined an effective procedure to compute $\mathit{Dangling}_{\mathcal{A}}(C)$ that uses basic operations on zones. Finally, we define a procedure for computing the set of uncontrollable predecessors using Theorem 2.

Lemma 1 (Compute uncontrollable predecessors). *Let $\mathcal{A}$ be a TSCA and $C \subseteq \mathbb{C}$. Then the set of* uncontrollable predecessors *of C in $\mathcal{A}$ is computed as*

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

where $unc_{\mathcal{A}}(C) = \{\, (\boldsymbol{q}, v) \xrightarrow{\boldsymbol{a}\Box} \mid (\boldsymbol{q}, v) \in \mathbb{C} \land (\boldsymbol{a}$ *urgent* $\lor\ \boldsymbol{a}$ *greedy match* $\lor$ $(\nexists (\boldsymbol{q}_2, v) \in Reachable_{\mathcal{A}}(C), (\boldsymbol{q}'_2, v') \in Successful_{\mathcal{A}}(C).(\boldsymbol{q}'_2, v') \in Post_{\mathcal{A},\boldsymbol{a}'}(\boldsymbol{q}_2, v)^{\nearrow}$ $\land\ \boldsymbol{a}_{(i)} = \boldsymbol{a}'_{(i)} = a \in \mathsf{R} \land \boldsymbol{a}'$ *match* $\land\ \boldsymbol{q}_{(i)} = \boldsymbol{q}_{2(i)}))\,\}$

With our results, safe TSCA orchestrations can be implemented using libraries for timed games [19, 20] with primitive zone operations (i.e. $\cup$, $\cap$, $\setminus$, $^{\nearrow}$ and $^{\swarrow}$).

4 Running Example Revisted

We continue our running example with a PriviledgedClient, depicted in Fig. 3a, optionally asking for a discount room via a permitted request, but after 8 t.u. (in its initial state) urgently requests a normal room. In orchestration f^* of composition (Hotel $\otimes$ DiscountClient) $\otimes$ PriviledgedClient, the discount request of DiscountClient could be matched before one of the requests of PriviledgedClient. But, this interaction is prevented in f^*. Let $\boldsymbol{a} = (\overline{discount}, discount, \bullet)\Box_\ell$, $\boldsymbol{b} = (\bullet, \bullet, room)\Box_u$, $t1 = ((q_{H0}, q_{D0}, q_{P0}), y \geq 50, \boldsymbol{a}, y \leftarrow 0, (q_{H1}, q_{D1}, q_{P0}))$ and $t2 = ((q_{H1}, q_{D1}, q_{P0}), x \geq 8, \boldsymbol{b}, \varnothing, (q_{H1}, q_{D1}, q_{P1}))$. Now $t1$ is not enabled by f^* or else we can reach a configuration c_2 in uncontrollable disagreement via $c_0 \xrightarrow{\delta = 50} c_1 \xrightarrow{\boldsymbol{a}} c_2 \xrightarrow{\delta = 0} c_2 \xrightarrow{\boldsymbol{b}}$. In c_2, the uncontrollable transition t_2 is enabled, but urgent request $\boldsymbol{b}$ is not matched, thus violating agreement. The first transition enabled in f^* is $((q_{H0}, q_{D0}, q_{P0}), x \geq 8, (\overline{room}, \bullet, room)\Box_u, y \leftarrow 0, (q_{H1}, q_{D0}, q_{P1}))$.

Thus, PriviledgedClient interacts with Hotel prior to DiscountClient, who is served successively. This is only possible as both lazy request $(\bullet, discount)\Box_\ell$ and 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.

Next consider the TSCA of Figs. 3b–d, variants of the previous contracts: BusinessClientU requests urgently a room within 5 t.u., BusinessClientL requests lazily a room within 8 t.u, while Hotel2 offers only a normal room (no discount).

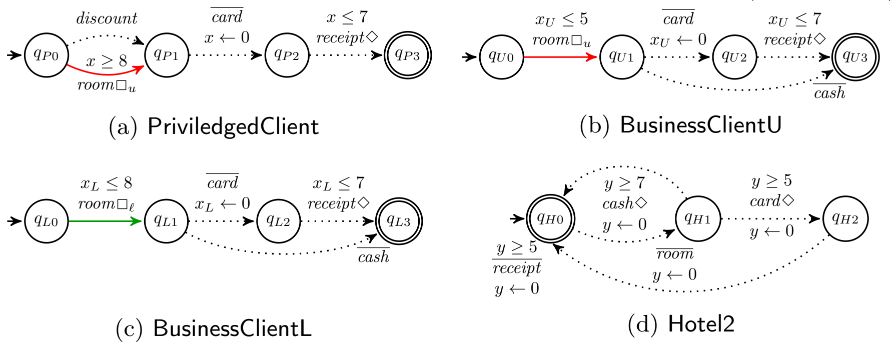
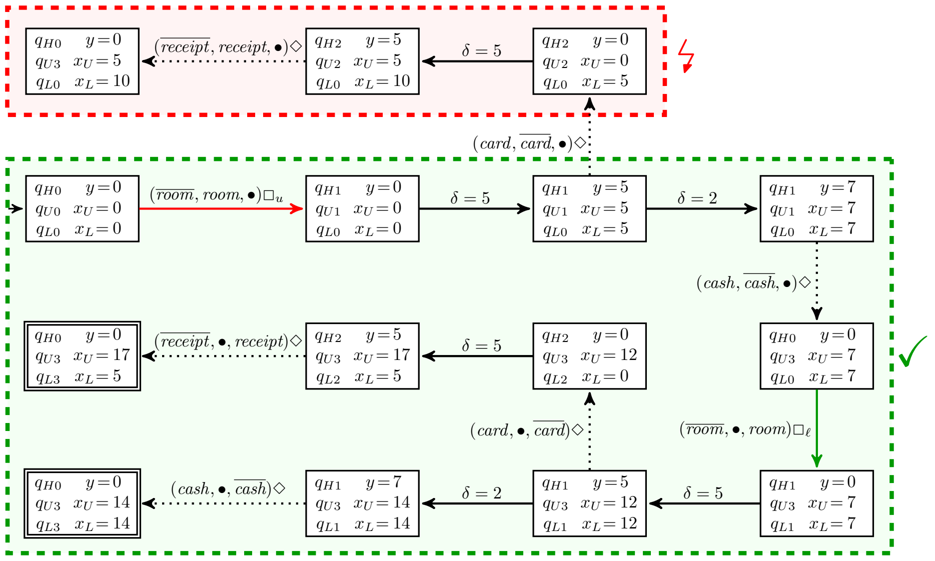


Fig. 3: TSCA: (a) priviledged, (b) urgent (c) lazy business clients, (d) hotel2

First look at the (Hotel2 ⊗ BusinessClientL) ⊗ BusinessClientU orchestration. It is empty (i.e. no agreement). In the initial state of Hotel2 ⊗ 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: a violation, so the initial configuration is in uncontrollable disagreement.

Next look at (Hotel2 ⊗ BusinessClientU) ⊗ BusinessClientL's orchestration f^*. Part of the behaviour allowed by f^* is depicted in Fig. 4 in the fragment marked with ✓ (in this figure, a transition is fired as soon as it is enabled). Now BusinessClientU performs the transaction with the hotel first. In case of card payments, the minimum time required to reach state $\boldsymbol{q} = (q_{H0}, q_{U3}, q_{L0})$ is 5+5=10 t.u., with clocks evaluation $v = (y = 0, x_U = 5, x_L = 10)$. In $(\boldsymbol{q}, v)$ (the top leftmost configuration in Fig. 4), the (lazy) necessary room request of BusinessClientL can no longer be satisfied as it should have been matched within 8 t.u., so violating agreement. Thus f^* forbids card payments of BusinessClientU. Note that also the two previous configurations (contained in the fragment marked with ϟ in Fig. 4) are forbidden in f^*, as they are in uncontrollable disagreement.

If, however, BusinessClientU pays cash, then the minimum time required to reach state $\boldsymbol{q}$ is 7 t.u., with clocks evaluation $v' = (y = 0, x_U = 7, x_L = 7)$. Indeed, in configuration $(\boldsymbol{q}, v')$ (the central rightmost configuration in the fragment marked with ✓ in Fig. 4) the lazy room request of 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 BusinessClientL's lazy room request, in the resulting safe orchestration BusinessClientU is only allowed to pay with cash.



q_{H0} $y=0$ q_{U3} $x_U=5$ q_{L0} $x_L=10$
$(\overline{receipt}, receipt, \bullet)\Diamond$
q_{H2} $y=5$ q_{U2} $x_U=5$ q_{L0} $x_L=10$
$\delta=5$
q_{H2} $y=0$ q_{U2} $x_U=0$ q_{L0} $x_L=5$
ϟ
$(card, \overline{card}, \bullet)\Diamond$
q_{H0} $y=0$ q_{U0} $x_U=0$ q_{L0} $x_L=0$
$(\overline{room}, room, \bullet)\Box_u$
q_{H1} $y=0$ q_{U1} $x_U=0$ q_{L0} $x_L=0$
$\delta=5$
q_{H1} $y=5$ q_{U1} $x_U=5$ q_{L0} $x_L=5$
$\delta=2$
q_{H1} $y=7$ q_{U1} $x_U=7$ q_{L0} $x_L=7$
$(cash, \overline{cash}, \bullet)\Diamond$
q_{H0} $y=0$ q_{U3} $x_U=17$ q_{L3} $x_L=5$
$(\overline{receipt}, \bullet, receipt)\Diamond$
q_{H2} $y=5$ q_{U3} $x_U=17$ q_{L2} $x_L=5$
$\delta=5$
q_{H2} $y=0$ q_{U3} $x_U=12$ q_{L2} $x_L=0$
q_{H0} $y=0$ q_{U3} $x_U=7$ q_{L0} $x_L=7$
$(card, \bullet, \overline{card})\Diamond$
$(\overline{room}, \bullet, room)\Box_\ell$
✓
q_{H0} $y=0$ q_{U3} $x_U=14$ q_{L3} $x_L=14$
$(cash, \bullet, \overline{cash})\Diamond$
q_{H1} $y=7$ q_{U3} $x_U=14$ q_{L1} $x_L=14$
$\delta=2$
q_{H1} $y=5$ q_{U3} $x_U=12$ q_{L1} $x_L=12$
$\delta=5$
q_{H1} $y=0$ q_{U3} $x_U=7$ q_{L1} $x_L=7$

Fig. 4: Excerpt of $TS_{(\text{Hotel2} \otimes \text{BusinessClientU}) \otimes \text{BusinessClientL}}$, whose fragment marked with ✓ is allowed in the safe orchestration whereas the one marked with ϟ is not

5 Conclusions and Future Work

We have presented TSCA, a new formalism for specifying *service contracts* with *real-time constraints*, and for *synthesising* their *safe orchestration* in the presence of service requests with different levels of criticality (viz. *urgent*, *greedy* and *lazy*).

We plan to implement the theory in a prototype, extending tools for contract automata [21–23] and reusing libraries from timed games for operations on zones [19, 20], to which orchestration synthesis has been reduced (cf. Theorem 2). We would also like to equip the formalism with weighted actions, e.g. to specify the prices of hotel rooms or how much clients are willing to pay for their room.

References

1. D. Georgakopoulos and M.P. Papazoglou. *Service-oriented Computing.* MIT, 2008.
2. A. Bouguettaya *et al.* A Service Computing Manifesto: The Next 10 Years. *Commun. ACM*, 60(4):64–72, April 2017.
3. M. Bartoletti, T. Cimoli, and R. Zunino. Compliance in Behavioural Contracts: A Brief Survey. In *Programming Languages with Applications to Biology and Security*, *LNCS* 9465, pp. 103–121, 2015.
4. D. Basile, P. Degano, and G.L. Ferrari. A formal framework for secure and complying services. *J. Supercomput.*, 69(1):43–52, 2014.
5. D. Basile, P. Degano, and G.L. Ferrari. Automata for Specifying and Orchestrating Service Contracts. *Log. Meth. Comput. Sci.*, 12(4:6):1–51, 2016.
6. D. Basile, F. Di Giandomenico, S. Gnesi, P. Degano, and G.L. Ferrari. Specifying Variability in Service Contracts. In *VaMoS'17*, pp. 20–27. ACM, 2017.
7. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987.
8. D. Basile, M.H. ter Beek, F. Di Giandomenico, and S. Gnesi. Orchestration of Dynamic Service Product Lines with Featured Modal Contract Automata. In *SPLC*, pp. 117–122. ACM, 2017.
9. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller Synthesis for Timed Automata. *IFAC Proceedings Volumes*, 31(18):447–452, 1998.
10. F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, *LNCS* 3653, pp. 66–80, 2005.
11. H. Hüttel *et al.* Foundations of Session Types and Behavioural Contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.
12. L. de Alfaro and T.A. Henzinger. Interface Automata. In *ESEC/FSE*, pp. 109–120. ACM, 2001.
13. N.A. Lynch and M.R. Tuttle. An Introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989.
14. R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoret. Comput. Sci.*, 126(2):183–235, 1994.
15. A. David, K.G. Larsen, A. Legay, U. Nyman, and A. Wąsowski. Timed I/O Automata. In *HSCC*, pp. 91–100. ACM, 2010.
16. K.G. Larsen, U. Nyman, and A. Wąsowski. Modal I/O Automata for Interface and Product Line Theories. In *ESOP*, *LNCS* 4421, pp. 64–79, 2007.
17. S. Azzopardi, G.J. Pace, F. Schapachnik, and G. Schneider. Contract automata. *Artif. Intell. Law*, 24(3):203–243, 2016.

18. P. Bouyer, N. Markey, and O. Sankur. Robust Reachability in Timed Automata: A Game-Based Approach. In *ICALP*, *LNCS* 7392, pp. 128–140, 2012.
19. A. David *et al.* UPPAAL DBM Library, 2017.
20. A. Legay and L.-M. Traonouez. PyEcdar: Towards Open Source Implementation for Timed Systems. In *ATVA*, *LNCS* 8172, pp. 460–463, 2013.
21. D. Basile, P. Degano, G.L. Ferrari, and E. Tuosto. Playing with Our CAT and Communication-Centric Applications. In *FORTE*, *LNCS* 9688, pp. 62–73, 2016.
22. D. Basile, F. Di Giandomenico, and S. Gnesi. FMCAT: Supporting Dynamic Service-based Product Lines. In *SPLC*, pp. 3–8. ACM, 2017.
23. D. Basile, M.H. ter Beek, and S. Gnesi. Modelling and Analysis with Featured Modal Contract Automata. In *SPLC*. ACM, 2018.