Bridging the Gap Between Supervisory Control and Coordination of Services: Synthesis of Orchestrations and Choreographies

Davide Basile, Maurice H. ter Beek, Rosario Pugliese

Bridging the Gap between Supervisory Control and Coordination of Services: Synthesis of Orchestrations and Choreographies

Davide Basile[1][0000−0002−7196−6609], Maurice H. ter Beek[2][0000−0002−2930−6367], and Rosario Pugliese[1][0000−0002−1419−1405]

[1] University of Florence, Florence, Italy
{davide.basile,rosario.pugliese}@unifi.it
[2] ISTI–CNR, Pisa, Italy, maurice.terbeek@isti.cnr.it

Abstract. We explore the frontiers between coordination and control systems by discussing a number of contributions to bridging the gap between supervisory control theory and coordination of services. In particular, we illustrate how the classical synthesis algorithm from supervisory control theory to obtain the so-called most permissive controller can be modified to synthesise orchestrations and choreographies of service contracts formalised as contract automata. The key ingredient to make this possible is a novel notion of controllability. Finally, we present an abstract parametric synthesis algorithm and show that it generalises the classical synthesis as well as the orchestration and choreography syntheses.

Keywords: Service Contracts · Contract Automata · Controller Synthesis · Orchestration · Choreography

1 Introduction

Coordination of services describes how control and data exchanges are coordinated in distributed service-based applications and systems. Their principled design is identified as one of the primary research challenges for the next 10 years, and the recent Service Computing Manifesto [22] points out that "Service systems have so far been built without an adequate rigorous foundation that would enable reasoning about them" and, moreover, that "The design of service systems should build upon a formal model of services".

Two widely adopted approaches to the coordination of services are *orchestration* and *choreography*. Intuitively, an orchestration yields the description of a distributed workflow from "one party's perspective" [45], whereas a choreography describes the behaviour of the involved parties from a "global viewpoint" [38]. In an orchestrated model, the service components are coordinated by a special component, the *orchestrator*, which, by interacting with them, dictates the workflow at runtime. In a choreographed model, instead, the service components autonomously execute and interact with each other on the basis of a local control flow expected to comply with their role as specified by the global viewpoint. Ideally, a choreographed model is more efficient due to the absence of the overhead of communications with the orchestrator. Any choreography can be trivially transformed into an orchestration of services, by adding an idle orchestrator.

Similarly, by explicitly adding an orchestrator and its interactions with the service components, and hence the relative overhead, it is possible to transform an orchestration of services into a choreography.

In [13], two orchestrated and choreographed automata-based models of services, called *contract automata*[3] and *communicating finite state machines*, respectively, were studied and related. The goal is to compose the automata so that each service is capable of reaching an accepting (final) state by synchronous/asynchronous one-to-one interactions with the other services in the composition. The main difference relies on the fact that communicating machines name the recipient service of each interaction upfront and use FIFO buffers to interact with each other, whereas contract automata are oblivious of their partners and an orchestration is synthesised to drive their interactions. In particular, the model of contract automata was further developed in, e.g., [11,12,14].

The orchestration synthesis was borrowed from the synthesis of the most permissive controller (mpc) from Supervisory Control Theory [47,24] (SCT), whose aim is to coordinate an ensemble of (local) components into a (global) system that functions correctly. In the context of contract automata, this amounts to refine the composition of service contracts into its largest sub-portion whose behaviour is non-blocking and safe (a notion of service compliance). The adaptation of the mpc synthesis for synthesising an orchestration of services required the introduction of a novel notion of semi-controllability. Basically, the assumption of the presence of an unpredictable environment was dropped in favour of a milder notion of predictable necessary service requests to be fulfilled.

In this paper, we report on the efforts to relate the mpc synthesis and the orchestration synthesis of contract automata through a homogeneous formalisation. The need for semi-controllability is showcased with intuitive examples and its expressiveness is evaluated with respect to standard SCT notions of controllable and uncontrollable actions. Moreover, a novel choreography synthesis algorithm is introduced as a refined version of the orchestration synthesis. Finally, we show that all synthesis algorithms presented in this paper are generalised into a single abstract synthesis algorithm from which each can be obtained through a different instantiation.

The paper is organised as follows. Section 2 contains background notions and results concerning contract automata and SCT. Section 3 and Section 4 introduce the synthesis of orchestrations and the novel synthesis of choreographies in the setting of (modal service) contract automata. Section 5 demonstrates that all the previously introduced syntheses algorithms are instantiations of a more abstract, parametric synthesis algorithm. Section 6 discusses related work, while Section 7 concludes the paper and provides some hints for future work.

2 Background

In this section, we provide the background needed to appreciate our contributions on the crossroads of supervisory control theory and coordination of services formalised as modal service contract automata.

[3] Not to be confused with the contract automata of [7] meant to formalise legal contracts among two parties expressed in natural language.

2.1 Contract Automata

A Contract Automaton (CA) represents either a single service (in which case it is called a *principal*) or a multi-party composition of services performing actions. The number of principals of a CA is called its *rank*. The states of a CA are vectors of states of principals. In the following, notation $\vec{v}$ stands for a vector and $\vec{v}_{(i)}$ is the ith element. The transitions of CA are labelled with *actions*, which are vectors of elements in the set of *basic actions* $\mathsf{L} = \mathsf{R} \cup \mathsf{O} \cup \{\bullet\}$, with $\mathsf{R} \cap \mathsf{O} = \emptyset$ and $\bullet \notin \mathsf{R} \cup \mathsf{O}$. Intuitively, R is the set of *requests* (depicted as non-overlined labels on arcs, e.g. *ins*), O is the set of *offers* (depicted as overlined labels on arcs, e.g. $\overline{ins}$), and $\bullet$ is a distinguished symbol representing the *idle* action. An *action* is then a vector $\vec{a}$ of basic actions where there is either a single offer, or a single request, or a single pair of request-offer that match, i.e. there exist i and j such that $\vec{a}_{(i)}$ is an offer and $\vec{a}_{(j)}$ is the complementary request; all other elements of the vector are the symbol $\bullet$. Such action is called *request*, *offer*, or *match*, respectively. A transition is also deemed request/offer/match according to its labelling action. The goal of each principal is to reach an accepting (*final*) state such that all its requests and offers are matched. In [14], CA were equipped with *modalities*, i.e. *necessary* ($\Box$) and *permitted* ($\Diamond$) requests, respectively, and the formalism was called Modal Service Contract Automata (MSCA), formally defined below.

Definition 1 (MSCA [14]). *Given a finite set of states $Q = \{q_1, q_2, \ldots\}$, a* Modal Service Contract Automata (MSCA) *$\mathcal{A}$ of rank n is a septuple $\langle Q, \vec{q}_0, A^\Diamond, A^\Box, A^o, T, F\rangle$, with set of states $Q = Q_1 \times \ldots \times Q_n \subseteq Q^n$, initial state $\vec{q}_0 \in Q$, $A^\Diamond, A^\Box \subseteq \mathsf{R}$ (pairwise disjoint) finite sets of permitted and necessary requests, respectively, with set of requests $A^r = A^\Diamond \cup A^\Box$, set of offers $A^o \subseteq \mathsf{O}$, set of final states $F \subseteq Q$, set of transitions $T \subseteq Q \times A \times Q$, where $A \subseteq (A^r \cup A^o \cup \{\bullet\})^n$, partitioned into* permitted *transitions $T^\Diamond$ and* necessary *transitions $T^\Box$, s.t.: (i) given $t = (\vec{q}, \vec{a}, \vec{q}\,') \in T$, $\vec{a}$ is either a request or an offer or a match; (ii) $\forall i \in 1 \ldots n$, $\vec{a}_{(i)} = \bullet$ implies $\vec{q}_{(i)} = \vec{q}\,'_{(i)}$. (iii) $t \in T^\Diamond$ iff $\vec{a}$ is either a request or a match on $a \in A^\Diamond$ or an offer on $\overline{a} \in A^o$; otherwise $t \in T^\Box$.*

A *principal* is an MSCA of rank 1 such that $A^r \cap co(A^o) = \emptyset$, where the *involution* $co : \mathsf{L} \to \mathsf{L}$ establishing matching among requests and offers is such that (abusing notation) $co(\mathsf{R}) = \mathsf{O}$, $co(\mathsf{O}) = \mathsf{R}$, and $co(\bullet) = \bullet$. A *step* $(w, \vec{q}) \xrightarrow{\vec{a}} (w', \vec{q}')$ occurs iff $w = \vec{a}w'$, $w' \in A^*$, and $(\vec{q}, \vec{a}, \vec{q}') \in T$. Let $\to^*$ be the reflexive and transitive closure of $\to$. The *language* of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \{\, w \mid (w, \vec{q}_0) \xrightarrow{w}{}^* (\varepsilon, \vec{q}),\ \vec{q} \in F \,\}$. A step may be denoted $\vec{q} \xrightarrow{\vec{a}}$ if w, w', and $\vec{q}'$ are irrelevant, and $(w, \vec{q}) \to (w', \vec{q}')$ if $\vec{a}$ is. Unless stated differently, the MSCA $\mathcal{A} = \langle Q_\mathcal{A}, \vec{q}_{0\,\mathcal{A}}, A^\Diamond_\mathcal{A}, A^\Box_\mathcal{A}, A^o_\mathcal{A}, T_\mathcal{A}, F_\mathcal{A}\rangle$ of rank n is assumed to be given. Subscript $\mathcal{A}$ may be omitted if no confusion may arise.

Composition of services is rendered through the composition of their MSCA models by means of the *composition operator* $\otimes$. This operator basically interleaves or matches the transitions of the component MSCA, but, whenever two component MSCA are ready on their respective request/offer action, then the match is forced to happen. Moreover, a match involving a necessary request is itself necessary. In the resulting MSCA, states and actions are vectors of states and actions of the component MSCA, respectively. The composition is non-associative, i.e. pre-existing matches are not rearranged if a new MSCA joins the composition afterwards.

In a composition of MSCA, typically various properties are analysed. We are especially interested in *agreement* and *strong agreement* (a.k.a. in the literature as progress of interactions, deadlock freedom, compliance or conformance of contracts). In an MSCA in strong agreement, all requests and offers must be matched. Instead, the property of agreement only requires to match all requests. An MSCA admits (strong) agreement if it has a trace satisfying the corresponding property, and it is *safe* if all its traces are such.

2.2 Supervisory Control Theory

The aim of Supervisory Control Theory [47,24] (SCT) is to provide an algorithm to synthesise a finite state automaton model of a *supervisory controller* from given (component) finite state automata models of the uncontrolled system and its requirements. The synthesised supervisory controller, if successfully generated, is such that the controlled system, which is the composition (i.e. synchronous product) of the uncontrolled system and the supervisory controller, satisfies the requirements and is additionally *non-blocking*, *controllable*, and *maximally permissive*.

An automaton is *non-blocking* if from each state at least one of the so-called *marked states* (distinguished stable states representing completed 'tasks' [47]) can be reached without passing through so-called *forbidden states*, meaning that the system always has the possibility to return to an accepted stable state (e.g. a final state). The algorithm assumes that marked states and forbidden states are indicated for each component model. SCT distinguishes between *observable* and *unobservable*, *controllable* and *uncontrollable* actions, where unobservable actions are also uncontrollable. The supervisory controller is not permitted to directly block uncontrollable actions from occurring; the controller is only allowed to disable them by preventing controllable actions from occurring. Intuitively, controllable actions correspond to stimulating the system, while uncontrollable actions correspond to messages provided by the environment, like sensors, which may be neglected but cannot be denied from existing. Finally, the fact that the resulting supervisory controller is *maximally permissive* (or least restrictive) means that as much behaviour of the uncontrolled system as possible is still present in the controlled system without violating neither the requirements, nor controllability, nor the non-blocking condition.

From the seminal work of Ramadge and Wonham [47], we know that a unique maximally permissive supervisory controller exists, provided that all actions are observable. This is called the *most permissive controller* (*mpc*); it coordinates an ensemble of (local) components into a (global) system that works correctly. The synthesis algorithm suffers from the same state space explosion problem as model checking [35].

Intuitively, the synthesis algorithm for computing the mpc of a finite state automaton $\mathcal{A}$ works as follows. The mpc is computed through an iterative procedure that at each step i updates incrementally a set of states R_i containing the *bad* states, i.e. those states that cannot prevent a forbidden state to be eventually reached, and refines an automaton $\mathcal{K}_i$. The algorithm starts with an automaton $\mathcal{K}_0$ equal to $\mathcal{A}$ and a set R_0 containing all *dangling* states, where a state is dangling if it cannot be reached from the initial state or cannot reach a final state. At each step i the algorithm prunes in a backwards fashion transitions with target state in R_i or forbidden source state. The set R_i is updated by possibly adding source states of uncontrollable transitions of $\mathcal{A}$ with a bad target

state and dangling states. When no more updates are possible, the algorithm terminates. Since $\mathcal{A}$ is finite state and the set R_i can only increase at each step, termination is ensured. Now, suppose that at its termination the algorithm returns the pair $(\mathcal{K}_s, R_s)$. We have that the mpc is empty, if the initial state of $\mathcal{A}$ is in R_s; otherwise, the mpc is $\mathcal{K}_s$. We report below the standard synthesis algorithm, but we homogenise the notation and simplify the formulation, to align the algorithm with those presented in the next sections. For this purpose, we assume the standard mpc synthesis to operate on MSCA where necessary transitions ($T^{\Box}$) are uncontrollable whilst permitted transitions ($T^{\Diamond}$) are controllable. We use $\langle\,\rangle$ to denote the empty automaton. A state $q \in Q$ is *dangling* iff $\nexists\, w$ s.t. $q_0 \xrightarrow{w}{}^{*} q$ or $q \xrightarrow{w}{}^{*} q_f \in F$. $Dangling(\mathcal{A})$ denotes the set of dangling states of $\mathcal{A}$. Given two MSCA $\mathcal{A}$ and $\mathcal{A}'$, we say that $\mathcal{A}'$ is a *sub-automaton* of $\mathcal{A}$, written $\mathcal{A}' \subseteq \mathcal{A}$, whenever the components of $\mathcal{A}'$ are included in the corresponding ones of $\mathcal{A}$. Moreover, given two sets of states R and R', we let $(\mathcal{A}, R) \leq (\mathcal{A}', R')$ if $\mathcal{A}' \subseteq \mathcal{A}$ and $R \subseteq R'$. It is straightforward to show that $(MSCA \times 2^{Q}, \leq)$ is a complete partial order (cpo).

The algorithm to compute the mpc is then defined in terms of the *least fixed point* of a monotone function on the cpo $(MSCA \times 2^{Q}, \leq)$.

Definition 2 (Standard synthesis, adapted from [47]). *Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = Dangling(\mathcal{K}_0)$. We let the* synthesis function *$f : MSCA \times 2^{Q} \to MSCA \times 2^{Q}$ be defined as follows:*

$$
\begin{aligned}
&f(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i),\ \textit{with}\\
&\qquad T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus \{\, (\vec{q}, \vec{a}, \vec{q}') \in T_{\mathcal{K}_{i-1}} \mid \vec{q}' \in R_{i-1} \vee \vec{q} \textit{ is forbidden} \,\}\\
&\qquad R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^{\Box}_{\mathcal{A}},\ \vec{q}' \in R_{i-1} \,\} \cup Dangling(\mathcal{K}_i)
\end{aligned}
$$

Theorem 1 (Standard mpc, adapted from [47]). *The synthesis function f is monotone on the cpo $(MSCA \times 2^{Q}, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = sup(\{\, f^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The mpc *of $\mathcal{A}$, denoted by $\mathcal{K}_{\mathcal{A}}$, is:*

$$
\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\,\rangle & \textit{if } \vec{q}_0 \in R_s \\ \langle Q \setminus Dangling(\mathcal{K}_s), \vec{q}_0, A^{\Diamond}, A^{\Box}, A^{o}, T_{\mathcal{K}_s}, F \setminus Dangling(\mathcal{K}_s) \rangle & \textit{otherwise} \end{cases}
$$

3 Synthesis of Orchestrations

In this section, we discuss how we revised the classical synthesis algorithm from SCT (cf. Theorem 1) to obtain the mpc and synthesise orchestrations of MSCA.

Differently from standard SCT, all transitions of MSCA are *observable*, since MSCA model the execution of services in terms of their requests and offers. Originally, MSCA were capable of expressing only permitted requirements, corresponding to actions that are controllable by the orchestrator. Hence, in the synthesis of the orchestration, all transitions labelled by actions violating the property to be enforced were pruned, and all dangling states were removed (cf. [11]).

While permitted requests of MSCA are in one-to-one correspondence with controllable actions, interestingly this is not the case for necessary requests and uncontrollable

actions. A necessary (request) action is indeed a weaker constraint than an uncontrollable one. This stems from the fact that traditionally uncontrollable actions relate 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 MSCA, implies that it suffices that in the synthesised orchestration at least one such synchronisation (i.e. match) actually occurs. This is precisely what is modelled by the notion of *semi-controllable* actions, anticipated in [14] and introduced in [10,9], discussed next.

The importance of this novel notion in the synthesis algorithm is showcased by an intuitive example. Consider the two MSCA interacting on the necessary service request a depicted in Fig. 1 (left and middle), and their possible composition $\mathcal{A}$ depicted in Fig. 1 (right). Note that $\mathcal{A}$ models two possibilities of fulfilling request a from the leftmost automaton by matching it with a service offer $\overline{a}$ of the middle one. Note that a similar composition can be obtained in other automata-based formalisms (such as, e.g., (timed) I/O automata [43,3,31]). Now assume that a must be matched with $\overline{a}$ to obtain an agreement (i.e. it is *necessary*), and that for some reason the *bad* state ✗ is to be avoided in favour of the *successful* state ✓, i.e. in some sense we would like to express that a must be matched at some point, rather than always. In most automata-based formalisms this is not allowed and the resulting mpc is empty. In the MSCA formalism, it is possible to orchestrate the composition of the two automata on the left in such a way that the result is the automaton $\mathcal{A}$ on the right, but *without the state* ✗ and its incident transition.

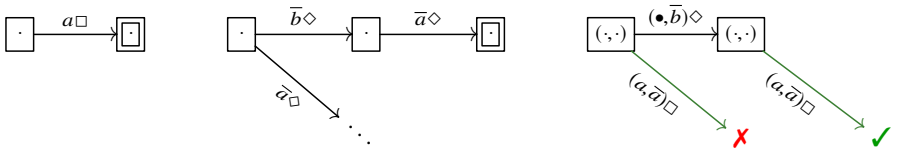

Fig. 1. Two MSCA (left and middle) and a possible composition $\mathcal{A}$ of them (right)

In fact, in the MSCA formalism, $\mathcal{A}$ depicts a composition in which the automata on the left can synchronise on a so-called semi-controllable action $a\Box$ either in their initial state or after the middle automaton has performed some other action $\overline{b}\Diamond$, ignoring in this case whether a bad or a successful state is reached in the end. Indeed, the notion of semi-controllability is independent from both the specific formalism being used and the requirement (e.g. agreement in case of MSCA) to be enforced.

As far as we know, we were the first to define a synthesis algorithm, in [10], that is capable of producing a controller that guarantees that *at least* one of these two synchronisations actually occurs. Indeed, in the standard synthesis algorithm (cf. Theorem 1), a can either be *controllable* and hence not necessary as we want, or *uncontrollable* thus requiring that a must *always* be matched, a stronger requirement than the one posed by declaring a as necessary.

To formalize the intuitions above[4], a semi-controllable transition t becomes controllable if in a given portion of $\mathcal{A}$ there exists a semi-controllable match transition t', with source and target states not dangling, such that in both t and t' the *same* principal, in the *same* local state, does the *same* request. Otherwise, t is uncontrollable.

[4] We refer the interested reader to [10,9] for a full account.

Definition 3 (Controllability). *Let $\mathcal{A}$ be an MSCA and let $t = (\vec{q}_1, \vec{a}_1, \vec{q}_1{}') \in T_{\mathcal{A}}$. Then:*

- *if $\vec{a}_1$ is an action on $a \in A^{\diamond} \cup A^{o}$, then t is* controllable *(in $\mathcal{A}$) and part of $T^{\diamond}$;*
- *if $\vec{a}_1$ is a request or match on $a \in A^{\square}$, then t is* semi-controllable *(in $\mathcal{A}$) and part of $T^{\square}$.*

Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists\, t' = (\vec{q}_2 \xrightarrow{\vec{a}_2} \vec{q}_2{}') \in T^{\square}_{\mathcal{A}'}$ in $\mathcal{A}'$ s.t. $\vec{a}_2$ is a match, $\vec{q}_2, \vec{q}_2{}' \notin Dangling(\mathcal{A}')$, $\vec{q}_{1(i)} = \vec{q}_{2(i)}$, and $\vec{a}_{1(i)} = \vec{a}_{2(i)} = a$, then t is controllable *in $\mathcal{A}'$ (via t'); otherwise, t is* uncontrollable *in $\mathcal{A}'$.*

The algorithm for synthesising an orchestration enforcing agreement of MSCA follows. The main adaptation of the mpc synthesis of Theorem 1 is that transitions are no longer declared uncontrollable, but instead they can be either controllable or semi-controllable. More importantly, a semi-controllable transition switches from controllable to uncontrollable only after it has been pruned in a previous iteration, in which case its source state becomes bad. Finally, in this case there are no forbidden states but rather forbidden transitions (i.e. requests, according to the property of agreement).

Definition 4 (MSCA orchestration synthesis, adapted from [14]). *Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = Dangling(\mathcal{K}_0)$. We let the* orchestration synthesis function *$f_o : MSCA \times 2^{Q} \to MSCA \times 2^{Q}$ be defined as follows:*

$$
\begin{aligned}
&f_o(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}\\
&\quad T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus \{\, (\vec{q} \to \vec{q}\,') = t \in T_{\mathcal{K}_{i-1}} \mid (\vec{q}\,' \in R_{i-1} \ \vee\ t \text{ is a request}) \,\}\\
&\quad R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to) \in T^{\square}_{\mathcal{A}} \text{ is uncontrollable in } \mathcal{K}_i \,\} \cup Dangling(\mathcal{K}_i)
\end{aligned}
$$

Theorem 2 (MSCA orchestration mpc, adapted from [14]). *The orchestration synthesis function f_o is monotone on the cpo $(MSCA \times 2^{Q}, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = sup(\{\, f_o^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The (orchestration) mpc $\mathcal{K}_{\mathcal{A}}$ of $\mathcal{A}$ is:

$$\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\,\rangle & \text{if } \vec{q}_0 \in R_s \\ \langle Q \setminus R_s, \vec{q}_0, A^{\diamond}, A^{\square}, A^{o}, T_{\mathcal{K}_s} \setminus T', F \setminus R_s \rangle & \text{otherwise} \end{cases}$$

where $T' = \{\, t = \vec{q} \to \in \mathcal{K}_s \mid t \text{ is controllable in } \mathcal{K}_s,\ \vec{q} \in R_s \,\}$.

Semi-controllability. We now show, by means of an example, that the encoding of an automaton $\mathcal{A}$ with semi-controllable actions into an automaton $\mathcal{A}'$ 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 mpc of $\mathcal{A}$ equals that of $\mathcal{A}'$. The encoding is sketched in Fig. 2: the automaton $\mathcal{A}'$ is obtained by turning all semi-controllable transitions of the automaton $\mathcal{A}$ from Fig. 1 (right) into uncontrollable transitions in $\mathcal{A}'$. The intuition for this construction is as follows. If the synchronisation on a specific semi-controllable action a occurs in n different transitions in $\mathcal{A}$ (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 $\mathcal{A}$, minus the one in which

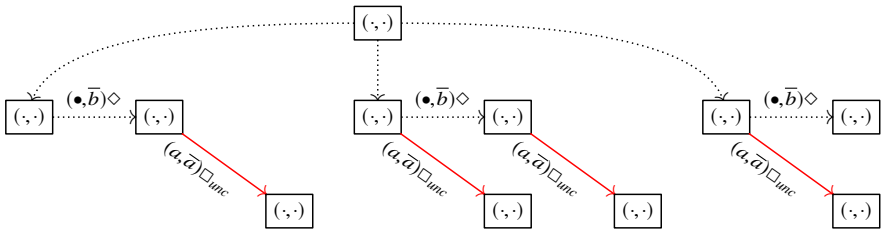

Fig. 2. Automaton $\mathcal{A}'$ uses uncontrollable transitions to encode automaton $\mathcal{A}$ from Fig. 1 (right)

all n semi-controllable transitions are pruned. In fact, without knowing a priori the set of forbidden and successful states, it is impossible to provide a more efficient encoding.

We explain why this is the case. Assume, by contradiction, that there exists an encoding that results in a 'smaller' automaton $\mathcal{A}''$, in which one of the $2^n - 1$ combinations of pruned transitions (say, P) is discarded. It then suffices to specify as a counterexample a property in $\mathcal{A}$ 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 $\mathcal{A}$ against such a property would prune exactly the semi-controllable transitions in P. Thus, in the synthesis of $\mathcal{A}''$ such an mpc would not be present, a contradiction.

4 Synthesis of Choreographies

In the previous section we have seen that the orchestration of MSCA is rendered as a particular mpc. The orchestrator is however implicit, in the sense that its interactions with the principals are hidden. Basically, one could assume that before interacting, each principal expects a message from the orchestrator and answers with an acknowledgement after the interaction terminates. The main intuition behind switching from an orchestrated to a choreographic coordination of contracts is that there is no longer the need for such 'hidden' interactions. Ideally, the principals moving autonomously are able to accomplish the behaviour foreseen by the synthesis, which in this case acts as a global type. Differently from the traditional choreographic approach, where the starting point is a global type, in MSCA the global type is synthesised automatically.

The requirements for ensuring that the synthesised mpc is a (form of) choreography were studied in [13,41]. Roughly, they amount to the so-called *branching condition* requiring that principals perform their offers/outputs independently of the other principals in the composition. To formalise it, we let $snd(\vec{a}) = i$ when $\vec{a}$ is a match action or an offer action and $\vec{a}_{(i)} \in \mathsf{O}$.

Definition 5 (Branching condition [13]). *An MSCA $\mathcal{A}$ satisfies the* branching condition *iff the following holds for each pair of states $\vec{q}_1$, $\vec{q}_2$ reachable in $\mathcal{A}$:*

$$\forall \vec{a} \text{ match action } . \ (\vec{q}_1 \xrightarrow{\vec{a}} \wedge\ snd(\vec{a}) = i \wedge \vec{q}_{1(i)} = \vec{q}_{2(i)}) \text{ implies } \vec{q}_2 \xrightarrow{\vec{a}}.$$

The branching condition is related to a phenomenon known as 'state sharing' in other coordination models (cf., e.g., [18]) according to which system components can

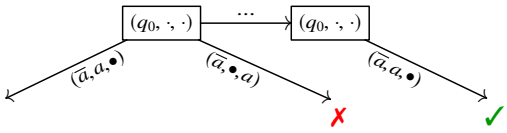

Fig. 3. Fragment of a possible service composition

influence potential synchronisations through their local (component) states even if they are not involved in the actual global (system) transition.

In [13] it is proved that the mpc corresponds to a well-behaving choreography if and only if it satisfies the branching condition and is strongly safe. Notably, in case the two conditions are not satisfied, that paper does not provide any algorithm for automatically synthesising a choreography, rather the contracts have to be manually amended. Instead, in the remainder of this section, we introduce an algorithm for automatically synthesising a well-behaving choreography.

The property to be enforced during the synthesis is strong agreement: all offers and requests have to be matched, because all messages have to be read (i.e. offers matched). Moreover, in the case of choreography, service contract requests are always permitted whereas service contract offers can be necessary. That is, their roles are swapped with respect to the case of orchestration.

In principle, the synthesis could trivially introduce a coordinator component and its interactions to coordinate the principals. However, this would reduce the choreography to a centralised coordination of contracts. To prevent this, the synthesis can only remove and never add behaviour. Hence, a choreography can only be synthesised if all principals are capable of interacting on their own without resorting to a central coordinator.

Similarly to orchestration synthesis, indicating transitions as either controllable or uncontrollable does not suffice for synthesising a choreography. Moreover, the notion of semi-controllability introduced for the orchestration case does not suffice for expressing necessary offers. Indeed, orchestration synthesis does not ensure the branching condition to be satisfied by the synthesised automaton, as the following example shows.

In Fig. 3, a fragment of a service composition is shown. Two global states are depicted, and in both the first service, say *Alice*, is in its initial local state (say, q_0). *Alice* performs an output (i.e. offer) $\overline{a}$ that can be directed to either *Bob* (second service) or *Carol* (third service), from the initial global state, or only to *Bob* from the other state. It is possible to reach either a successful (✓) or a bad (✗) state, left unspecified for the moment. Notably, the output of *Alice* is neither controllable nor uncontrollable nor semi-controllable by the synthesis.

Now assume that the $\overline{a}$ is controllable and from the initial global state both interactions eventually lead to a bad state (✗). In this case, those transitions are pruned by the synthesis, and the resulting automaton is wrongly approved. Indeed, *Alice* has no mean to understand when her output $\overline{a}$ is enabled, because she has not changed state. The branching condition, which is necessary for obtaining a well-behaving choreography, would be violated. Note that this would happen also if $\overline{a}$ were semi-controllable. In fact, to satisfy the branching condition, the synthesis should remove all outputs $\overline{a}$.

Conversely, assume that the $\overline{a}$ is uncontrollable and that it is possible from the initial global state to reach a successful state (✓) if the message $\overline{a}$ is received by *Bob*. In this

case, it would not be possible to prune the transition from the initial state leading to ✗, because it is also uncontrollable. The synthesis would thus be empty, a wrong rejection, because a choreography exists in which *Alice* autonomously interacts with *Bob*.

In conclusion, a necessary action is rendered neither as uncontrollable nor semi-controllable and permitted actions require extra pruning operations during the synthesis. A novel notion of semi-controllability for a necessary action is required that is weaker than uncontrollable but stronger than the semi-controllable notion used in the synthesis of orchestration. Basically, for the choreography synthesis, a (semi-controllable) necessary transition $t = (\vec{q} \xrightarrow{\vec{a}_1}) \in T^{\Box}$ is detected to be uncontrollable iff no necessary transition $t' = (\vec{q} \xrightarrow{\vec{a}_2}) \in T^{\Box}$ exists from the same source state such that in both t and t' the same offer is provided by the same principal, but possibly with different receivers. Formally:

Definition 6. *Let $\mathcal{A}$ be an MSCA and let $t = (\vec{q}, \vec{a}_1, \vec{q}_1{}') \in T_{\mathcal{A}}$. Then:*

- *if $\vec{a}_1$ is an action on $a \in A^{\diamond}$, then t is* controllable *(in $\mathcal{A}$);*
- *if $\vec{a}_1$ is an offer or a match on $a \in A^{\Box}$, then t is* semi-controllable *(in $\mathcal{A}$).*

Moreover, given $\mathcal{A}' \subseteq \mathcal{A}$, if t is semi-controllable and $\exists\, t' = (\vec{q}, \vec{a}_2, \vec{q}_2{}') \in T^{\Box}_{\mathcal{A}'}$ s.t. $\vec{a}_2$ is a match, $\vec{q}, \vec{q}_2{}' \notin Dangling(\mathcal{A}')$, and $snd(\vec{a}) = i$ and $\vec{a}_{1(i)} = \vec{a}_{2(i)} = \overline{a}$, then t is controllable *in $\mathcal{A}'$ (via t'); otherwise, t is* uncontrollable *in $\mathcal{A}'$.*

Hence, again a necessary transition is a particular type of transition that switches from being controllable to uncontrollable in case a condition on the global automaton is not met. Note that this condition is stronger than the one required for the case of orchestration (semi-controllability), because for the case of choreography transitions t and t' in Definition 6 share the source state. Moreover, also in this case it can be shown that the encoding of this type of semi-controllable transition into an uncontrollable one would result in an exponential growth of the state space of the model.

Similarly to the orchestration synthesis in Definition 4, when a semi-controllable transition previously removed by the synthesis switches from controllable to uncontrollable, its source state is detected to be bad. Apart from the different notion of semi-controllability, another difference with respect to the orchestration synthesis is that each time a controllable transition is pruned, all other transitions violating the branching condition must also be removed. Finally, according to the property of strong agreement, both request and offer transitions are forbidden. The formalisation is provided below.

Definition 7 (MSCA choreography synthesis). *Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = Dangling(\mathcal{K}_0)$. We let the* choreography synthesis function *$f_c : MSCA \times 2^Q \to MSCA \times 2^Q$ be defined as follows:*

$$
\begin{aligned}
&f_c(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}\\
&\quad T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus (\{\, (\vec{q} \to \vec{q}') = t \in T_{\mathcal{K}_{i-1}} \mid \vec{q}' \in R_{i-1} \vee t \text{ is a request or an offer} \,\}\\
&\qquad\qquad \cup \{\, (\vec{q}_1 \xrightarrow{\vec{a}}) = t \in T_{\mathcal{K}_{i-1}} \mid \exists\, \vec{q}_2 : (snd(\vec{a}) = i \wedge \vec{q}_{1(i)} = \vec{q}_{2(i)})\\
&\qquad\qquad\qquad \wedge (\vec{q}_2 \xrightarrow{\vec{a}}) \notin T_{\mathcal{K}_{i-1}} \,\})\\
&\quad R_i = R_{i-1} \cup \{\, \vec{q} \mid (\vec{q} \to) \in T_{\mathcal{A}} \text{ is uncontrollable in } \mathcal{K}_i \,\} \cup Dangling(\mathcal{K}_i)
\end{aligned}
$$

Theorem 3 (MSCA choreography mpc). *The choreography synthesis function f_c is monotone on the cpo $(MSCA \times 2^{Q}, \leq)$ and its* least fixed point *is:*

$$(\mathcal{K}_s, R_s) = sup(\{\, f_c^n(\mathcal{K}_0, R_0) \mid n \in \mathbb{N} \,\})$$

The (choreography) mpc $\mathcal{K}_{\mathcal{A}}$ of $\mathcal{A}$ is:

$$\mathcal{K}_{\mathcal{A}} = \begin{cases} \langle\ \rangle & \text{if } \vec{q}_0 \in R_s \\ \langle Q \setminus R_s, \vec{q}_0, A^{\Diamond}, A^{\Box}, A^{o}, T_{\mathcal{K}_s} \setminus T', F \setminus R_s \rangle & \text{otherwise} \end{cases}$$

where $T' = \{\, t = \vec{q} \rightarrow \in \mathcal{K}_s \mid t$ is controllable in $\mathcal{K}_s,\ \vec{q} \in R_s \,\}$.
Moreover, $\mathcal{K}_{\mathcal{A}}$ satisfies the branching condition.

Returning to the example in Fig. 3, the wrongly accepted case is removed because, during the synthesis, the operation of pruning the transitions leading to bad states causes the removal of the remaining transition. Thus, the obtained choreography is empty. Similarly, the wrongly rejected case is not possible because, assuming that the output from the initial state is necessary, this necessary action is not rendered as uncontrollable as long as the output is matched by some other principal from the same initial state.

5 Abstract Synthesis

We have presented in the previous three sections three slightly different synthesis algorithms. As previously stated, in order to bridge the gap between standard synthesis and orchestration and choreography syntheses, the controllable and uncontrollable actions from SCT are related to permitted and necessary modalities, respectively, of MSCA.

The main intuition is that the SCT assumption of an unpredictable environment responsible for the uncontrollable transitions is not realistic in the case of coordination of services whose behaviour is known and observable. As a result, necessary actions are not in correspondence with uncontrollable actions, but rather require the introduction of a milder notion of controllability. The condition under which a controllable transition becomes uncontrollable varies depending on the particular synthesis algorithm (orchestration or choreography). Conversely, in the standard mpc synthesis such information is local, i.e. a transition is declared to be uncontrollable.

In this section, we discuss an abstract synthesis algorithm that generalises the previous algorithms by abstracting away the conditions under which a transition is pruned or a state is deemed bad. These two conditions, called *pruning predicate* (ϕ_p) and *forbidden predicate* (ϕ_f) are parameters to be instantiated by the corresponding instance of the synthesis algorithm (e.g. orchestration or choreography). Predicate ϕ_p is used for selecting the transitions to be pruned. Depending on the specific instance, non-local information about the automaton or the set of bad states is needed by ϕ_p. Therefore, ϕ_p takes as input the current transition to be checked, the automaton, and the set of bad states. If ϕ_p evaluates to true, then the corresponding transition will be pruned. Predicate ϕ_f is used for deciding whether a state becomes bad. The input parameters are the same as ϕ_p. However, ϕ_f only inspects necessary transitions ($T^{\Box}$). If ϕ_f evaluates to true, then the source state is deemed bad and added to the set R_i. The abstract synthesis algorithm is formally defined below.

Definition 8 (Abstract synthesis). *Let $\mathcal{A}$ be an MSCA, and let $\mathcal{K}_0 = \mathcal{A}$ and $R_0 = Dangling(\mathcal{K}_0)$. Given two predicates $\phi_p, \phi_f : T \times MSCA \times Q \to \mathsf{Bool}$, we let the* abstract synthesis function *$f_{(\phi_p,\phi_f)} : MSCA \times 2^Q \to MSCA \times 2^Q$ be defined as follows:*

$$f_{(\phi_p,\phi_f)}(\mathcal{K}_{i-1}, R_{i-1}) = (\mathcal{K}_i, R_i), \text{ with}$$
$$T_{\mathcal{K}_i} = T_{\mathcal{K}_{i-1}} \setminus \{ (\vec{q} \xrightarrow{\vec{a}}) = t \in T_{\mathcal{K}_{i-1}} \mid \phi_p(t, \mathcal{K}_{i-1}, R_{i-1}) = true \}$$
$$R_i = R_{i-1} \cup \{ \vec{q} \mid (\vec{q} \to) = t \in T^{\square}_{\mathcal{A}},\ \phi_f(t, \mathcal{K}_{i-1}, R_{i-1}) = true \} \cup Dangling(\mathcal{K}_i)$$

As in the previous cases, the mpc relative to the pair (ϕ_p, ϕ_f) is obtained by computing the least fixed point $(\mathcal{K}_s, \mathcal{R}_s)$ of $f_{(\phi_p,\phi_f)}$ and removing the states R_s from $\mathcal{K}_s$.

In the following, we show how to instantiate the abstract synthesis function to the standard synthesis function, to the orchestration synthesis function, or to the choreography synthesis function, and prove their correspondences.

Theorem 4 (Abstract mpc synthesis). *The standard synthesis function of Definition 2 coincides with the instantiation of the abstract synthesis function of Definition 8 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\phi_p^{mpc}(t, \mathcal{K}, R) = (\vec{q}' \in R) \vee (\vec{q} \text{ is forbidden})$$
$$\phi_f^{mpc}(t, \mathcal{K}, R) = (\vec{q}' \in R)$$

Note that in Theorem 4 the predicates do not use any non-local information related to the parameter $\mathcal{K}$. For both orchestration and choreography two different semi-controllability conditions are used to decide whether a state has become forbidden. These conditions are translated into the corresponding forbidden predicates.

Theorem 5 (Abstract orchestration synthesis). *The orchestration synthesis function of Definition 4 coincides with the instantiation of the abstract synthesis function of Definition 8 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\phi_p^{orc}(t, \mathcal{K}, R) = (t \text{ is a request }) \vee (\vec{q}' \in R)$$
$$\phi_f^{orc}(t, \mathcal{K}, R) = \nexists (\vec{q}_2 \xrightarrow{\vec{a}_2} \vec{q}_2{}') \in T^{\square}_{\mathcal{K}} : (\vec{a}_2 \text{ is a match }) \wedge (\vec{q}_2, \vec{q}_2{}' \notin Dangling(\mathcal{K})) \wedge (\vec{q}_{(i)} = \vec{q}_{2(i)}) \wedge (\vec{a}_{(i)} = \vec{a}_{2(i)} = a)$$

The pruning predicate of Theorem 5 does not use any information coming from the global automaton $\mathcal{K}$, whereas this is no longer the case for the forbidden predicate that indeed specifies the semi-controllability condition for the necessary transitions of an orchestration (cf. Definition 3).

Theorem 6 (Abstract choreography synthesis). *The choreography synthesis function of Definition 7 coincides with the instantiation of the abstract synthesis function of Definition 8 where, for a generic transition $t = (\vec{q}, \vec{a}, \vec{q}')$, predicates ϕ_p and ϕ_f are defined as follows:*

$$\phi_p^{cor}(t,\mathcal{K},R) = (t \textit{ is a request or an offer }) \vee (\vec{q}' \in R) \\ \vee (\exists\, \vec{q}_2 \in Q_{\mathcal{K}} \,:\, (snd(\vec{a}) = i) \wedge (\vec{q}_{(i)} = \vec{q}_{2(i)}) \wedge (\vec{q}_2 \xrightarrow{\vec{a}} \notin T_{\mathcal{K}}))$$

$$\phi_f^{cor}(t,\mathcal{K},R) = \nexists\, (\vec{q} \xrightarrow{\vec{a}_2} \vec{q}_2{}') \in T_{\mathcal{K}}^{\square} \,:\, (\vec{a}_2 \textit{ is a match }) \wedge (\vec{q}, \vec{q}_2{}' \notin \textit{Dangling}(\mathcal{K})) \\ \wedge (\vec{a}_{(i)} = \vec{a}_{2(i)} = \overline{a})$$

Notably, in Theorem 6 both predicates require global information on the whole automaton. Similarly to Theorem 5, the forbidden predicate codifies the semi-controllability condition of Definition 6. Moreover, the pruning predicate removes all transitions violating the branching condition (cf. Definition 5).

We believe that the synthesis algorithms are related. In particular, we conjecture that via a partial order of predicates, appropriately defined as $(\phi_{p_1}, \phi_{f_1}) \leq (\phi_{p_2}, \phi_{f_2})$ iff $(\phi_{p_1} \rightarrow \phi_{p_2}) \wedge (\phi_{f_1} \rightarrow \phi_{f_2})$, it can be proved that $(\phi_p^{orc}, \phi_f^{orc}) \leq (\phi_p^{mpc}, \phi_f^{mpc})$ and therefore $\mathcal{K}_{\mathcal{A}}^{mpc} \subseteq \mathcal{K}_{\mathcal{A}}^{orc}$. More generally, the cpo of predicates permits to perform abstraction of syntheses, in the sense that the lesser the pair of predicates the greater the corresponding synthesised automaton. This can be useful to perform partial syntheses and skip unnecessary checks or even potentially undecidable computations. For instance, given an MSCA $\mathcal{A}$, from $\mathcal{K}_{\mathcal{A}}^{orc} = \langle\ \rangle$, we can conclude $\mathcal{K}_{\mathcal{A}}^{mpc} = \langle\ \rangle$ without actually computing it, which could potentially require more computational effort.

6 Related Work

Our contributions to bridging the gap between SCT and coordination of services concern adaptations of the classical synthesis algorithm from SCT in order to synthesise orchestrations and choreographies of service contracts formalised as MSCA. In the literature, there exist many formalisms for modelling and analysing (service) contracts, ranging from behavioural type systems, including behavioural contracts [27,1,40] and session types [23,36,32,26,44], to automata-based formalisms, including interface automata [2] and (timed) (I/O) automata [43,3,31]. Foundational models for service contracts and session types are surveyed in [17,8,37].

The MSCA formalism used in this paper differs fundamentally from these models, which typically study notions of contract compliance involving only two parties, since MSCA primitively support *multi-party* compliance of contracts that *compete* on offering or requesting the same service. Furthermore, the above models do not consider modalities of services whereas MSCA provide primitive support for *permitted* and *necessary* service actions, resulting in the introduction of a novel notion of *semi-controllability* in the context of SCT. Modal Transition Systems (MTS) and their extensions [39], as adopted for instance in Software Product Line Engineering (SPLE [46,4]), like modal I/O automata [42] and MTS with variability constraints [19], do natively distinguish may and must modalities, but the other differences remain. In particular, they cannot explicitly handle dynamic composition by allowing new services that join composite services to intercept already matched actions.

We are only aware of two other applications of SCT to MTS. In [30], there is no direct relation between may/must and controllable/uncontrollable, and the modal automaton

(i.e. MTS with final states) is seen as a predicate that is satisfied if the plant automaton (i.e. the system to be refined against the predicate) is a sort of alternate refinement of the predicate. Similarly, in [33], the control objectives (i.e. the predicate) is a modal automaton, non-blockingness is not considered, and another modal automaton describes which actions are controllable and which are uncontrollable in the plant automaton. In this paper, the predicate is an invariant (i.e. forbidden states and forbidden transitions are given), the modal automaton (i.e. MSCA) is the plant, and a necessary transition induces different notions of controllability according to the adopted coordination paradigm.

SCT was first applied to SPLE in [20] by showing how the CIF 3 toolset [16] can automatically synthesise a single (global, family) model representing an automaton for each of the valid products of a product line from (i) a feature constraint with attributes (e.g. cost), (ii) behavioural component models associated with the features, and (iii) additional behavioural requirements like state invariants, action orderings, and guards on actions (reminiscent of the Featured Transition Systems of [28]). The resulting CIF 3 model satisfies all feature-related constraints as well as all given behavioural requirements. Since CIF 3 allows the export of such models in a format accepted by the mCRL2 model checker [29], the latter can be used to verify arbitrary behavioural properties expressed in the modal μ-calculus with data or its feature-oriented variant of [21]. An important advantage is that both CIF 3 and mCRL2 can be used off-the-shelf, meaning that no additional tools are required. Differently from our approach, all actions are controllable and orchestration is not considered, whilst a prototype tool supporting orchestration synthesis for contract automata is presented in [12].

The only approach by others to bridge the gap between SCT and coordination of services that we are aware of is that of [6], where services are formalised as so-called Service Labelled Transition Systems (SLTS), which are a kind of guarded automata with data. To this aim, SCT is adapted to deal with conditions and variables as well as with a means to enforce services based on runtime information. However, service composition through SLTS is based on the standard synchronous product, whilst the contract composition expresses competing contracts. More importantly, in [6], input actions are considered uncontrollable whilst output actions are controllable, in the standard view of a service interacting with the environment. Our contribution induces novel notions of controllability to express necessary requirements that are semi-controllable. The standard controller synthesis algorithm is used in [34] to synthesise adapters between services. These adapters act like proxies and are used to enforce properties such as deadlock-freedom. Compared to our work, the interactions between services are driven by their contracts rather than by adapters. The standard controller synthesis algorithm cannot be applied for synthesising a correct composition of contracts.

We conclude this section by describing two further extensions of MSCA, developed for different purposes, and for which we also defined adapted synthesis algorithms. In [9], we present Featured Modal Contract Automata (FMCA). Technically, we extend MSCA with a variability mechanism concerning structural constraints that operate on the service contract, used to define different configurations. This reflects the fact that services are typically reused in configurations that vary over time and need to dynamically adapt to changing environments [48]. Configurations are characterised by which service actions are mandatory and which forbidden. The valid configurations are those respecting all

structural constraints. We follow the well-established paradigm of SPLE, which aims at efficiently managing a product line (family) of highly (re)configurable systems to allow for mass customisation [46,4]. To compactly represent a product line, i.e. the set of valid product configurations, we use a so-called feature constraint, a propositional formula φ whose atoms are features [15], and we identify features as service actions (offers as well as requests). A valid product then distinguishes a set of mandatory and a set of forbidden actions. Consequently, we define an algorithm to compute the FMCA $\mathcal{K}_{\mathcal{A}_p}$ as the mpc for a valid product p of an FMCA $\mathcal{A}$. The main adaptation of the synthesis algorithm for MSCA is to consider as bad states also those that cannot prevent a forbidden action to be eventually executed and to discard the transitions labelled with actions forbidden by p. Moreover, if some action that is mandatory in p is unavailable in the automaton that results from the fixed point iteration, then the mpc is empty. In [10], we introduced Timed Service Contract Automata (TSCA) as an extension of the FMCA from [9] with real-time constraints. Formally, a configuration of a TSCA is a triple consisting of a recognised trace, a state, and a valuation of clocks. The (finite) behaviour recognised by a TSCA are traces of alternating time and discrete transitions, i.e. in a given configuration either time progresses (a silent action in the languages recognised by TSCA) or a discrete step to a new configuration is performed. Consequently, we define an algorithm to compute the orchestration synthesis of TSCA. To respect the timing constraints, we use the notion of zones from timed games [5,25]. The resulting synthesis algorithm resembles a timed game, but it differs from classical timed game algorithms [5,25,31] by combining two separate games, viz. *reachability* games (to ensure that marked states must be reachable) and *safety* games (to ensure that forbidden states are never traversed). 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).

7 Conclusion

In this paper, we have presented recent efforts in bridging the gap between the most permissive controller synthesis from Supervisory Control Theory with synthesis algorithms of orchestrations and choreographies of a formal model of service contracts called modal service contract automata. We have introduced a new algorithm capable of synthesising a safe non-blocking composition of service contracts that is directly translatable into a choreographed formalism. We have also introduced an abstract synthesis algorithm that generalises the synthesis of the choreography, as well as that of the orchestration and that of the most permissive controller.

The properties to be enforced in the algorithms that we have presented are all invariants specified through either forbidden states or forbidden transitions. Future work is needed to investigate the abstract syntheses under other non-invariant properties. Finally, further work is necessary to formally demonstrate that the different synthesis algorithms are related, as conjectured at the end of Section 5.

References

1. Acciai, L., Boreale, M., Zavattaro, G.: Behavioural contracts with request-response operations. Sci. Comp. Program. **78**(2), 248–267 (2013)

2. de Alfaro, L., Henzinger, T.: Interface Automata. In: ESEC/FSE. pp. 109–120. ACM (2001)
3. Alur, R., Dill, D.: A Theory of Timed Automata. Theoret. Comp. Sci. **126**(2), 183–235 (1994)
4. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer (2013)
5. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller Synthesis for Timed Automata. IFAC Proc. Vol. **31**(18), 447–452 (1998)
6. Atampore, F., Dingel, J., Rudie, K.: Automated Service Composition Via Supervisory Control Theory. In: WODES. pp. 28–35. IEEE (2016)
7. Azzopardi, S., Pace, G.J., Schapachnik, F., Schneider, G.: Contract automata: An operational view of contracts between interactive parties. Artif. Intell. Law **24**(3), 203–243 (2016)
8. Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in Behavioural Contracts: A Brief Survey. In: Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 103–121. Springer (2015)
9. Basile, D., ter Beek, M.H., Degano, P., Legay, A., Ferrari, G.L., Gnesi, S., Di Giandomenico, F.: Controller synthesis of service contracts with variability. Science of Computer Programming (2019), submitted
10. Basile, D., ter Beek, M.H., Legay, A., Traonouez, L..M.: Orchestration Synthesis for Real-Time Service Contracts. In: VECoS. LNCS, vol. 11181, pp. 31–47. Springer (2018)
11. Basile, D., Degano, P., Ferrari, G.L.: Automata for Specifying and Orchestrating Service Contracts. Log. Meth. Comp. Sci. **12**(4:6), 1–51 (2016)
12. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Playing with Our CAT and Communication-Centric Applications. In: FORTE. LNCS, vol. 9688, pp. 62–73. Springer (2016)
13. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Relating two automata-based models of orchestration and choreography. J. Log. Algebr. Meth. Program. **85**(3), 425–446 (2016)
14. Basile, D., Di Giandomenico, F., Gnesi, S., Degano, P., Ferrari, G.L.: Specifying Variability in Service Contracts. In: VaMoS. pp. 20–27. ACM (2017)
15. Batory, D.S.: Feature Models, Grammars, and Propositional Formulas. In: SPLC. LNCS, vol. 3714, pp. 7–20. Springer (2005)
16. van Beek, D.A., Fokkink, W.J., Hendriks, D., Hofkamp, A., Markovski, J., van de Mortel-Fronczak, J.M., Reniers, M.A.: CIF 3: Model-Based Engineering of Supervisory Controllers. In: TACAS. LNCS, vol. 8413, pp. 575–580. Springer (2014)
17. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Web Service Composition Approaches: From Industrial Standards to Formal Methods. In: ICIW. IEEE (2007)
18. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication Requirements for Team Automata. In: COORDINATION. LNCS, vol. 10319, pp. 256–277. Springer (2017)
19. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: 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 (2016)
20. ter Beek, M.H., Reniers, M.A., de Vink, E.P.: Supervisory Controller Synthesis for Product Lines Using CIF 3. In: ISoLA. LNCS, vol. 9952, pp. 856–873. Springer (2016)
21. ter Beek, M.H., de Vink, E.P., Willemse, T.A.C.: Family-Based Model Checking with mCRL2. In: FASE. LNCS, vol. 10202, pp. 387–405. Springer (2017)
22. Bouguettaya, A., Singh, M., Huhns, M., Sheng, Q.Z., Dong, H., Yu, Q., Neiat, A.G., 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.: A Service Computing Manifesto: The Next 10 Years. Commun. ACM **60**(4), 64–72 (2017)
23. Bruni, R., Lanese, I., Melgratti, H.C., Tuosto, E.: Multiparty Sessions in SOC. In: COORDINATION. LNCS, vol. 5052, pp. 67–82. Springer (2008)
24. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer (2006)

25. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In: CONCUR. LNCS, vol. 3653, pp. 66–80. Springer (2005)
26. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On Global Types and Multi-Party Sessions. Log. Meth. Comp. Sci. **8**(1:24), 1–45 (2012)
27. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. ACM Trans. Program. Lang. Syst. **31**(5), 19:1–19:61 (2009)
28. Classen, A., Cordy, M., Schobbens, P..Y., Heymans, P., Legay, A., Raskin, J..F.: Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. IEEE Trans. Softw. Eng. **39**(8), 1069–1089 (2013)
29. Cranen, S., Groote, J.F., Keiren, J.J.A., Stappers, F.P.M., de Vink, E.P., Wesselink, W., Willemse, T.A.C.: An Overview of the mCRL2 Toolset and Its Recent Advances. In: TACAS. LNCS, vol. 7795, pp. 199–213. Springer (2013)
30. Darondeau, P., Dubreil, J., Marchand, H.: Supervisory Control for Modal Specifications of Services. IFAC Proc. Vol. **43**(12), 418–425 (2010)
31. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In: HSCC. pp. 91–100. ACM (2010)
32. Dezani-Ciancaglini, M., de'Liguoro, U.: Sessions and Session Types: An Overview. In: WS-FM. LNCS, vol. 6194, pp. 1–28. Springer (2010)
33. Feuillade, G., Pinchinat, S.: Modal Specifications for the Control Theory of Discrete Event Systems. Discrete Event Dyn. Syst. **17**(2), 211–232 (2007)
34. Gierds, C., Mooij, A.J., Wolf, K.: Reducing Adapter Synthesis to Controller Synthesis. IEEE Trans. Services Computing **5**(1), 72–85 (2012)
35. Gohari, P., Wonham, W.M.: On the complexity of supervisory control design in the RW framework. IEEE Trans. Syst., Man, Cybern. B, Cybern. **30**(5), 643–652 (2000)
36. Honda, K., Yoshida, N., Carbone, M.: Multiparty Asynchronous Session Types. In: POPL. pp. 273–284. ACM (2008)
37. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P..M., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Torres Vieira, H., Zavattaro, G.: Foundations of Session Types and Behavioural Contracts. ACM Comput. Surv. **49**(1), 3:1–3:36 (2016)
38. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., Barreto, C.: Web Services Choreography Description Language v1.0. `https://www.w3.org/TR/ws-cdl-10/` (2005)
39. Křetínský, J.: 30 Years of Modal Transition Systems: Survey of Extensions and Analysis. In: Models, Algorithms, Logics and Tools, LNCS, vol. 10460, pp. 36–74. Springer (2017)
40. Laneve, C., Padovani, L.: An algebraic theory for web service contracts. Form. Asp. Comp. **27**(4), 613–640 (2015)
41. Lange, J., Tuosto, E., Yoshida, N.: From Communicating Machines to Graphical Choreographies. In: POPL. pp. 221–232. ACM (2015)
42. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: ESOP. LNCS, vol. 4421, pp. 64–79. Springer (2007)
43. Lynch, N., Tuttle, M.: An Introduction to Input/Output Automata. CWI Q. **2**, 219–246 (1989)
44. Michaux, J., Najm, E., Fantechi, A.: Session types for safe Web service orchestration. J. Log. Algebr. Program. **82**(8), 282–310 (2013)
45. Peltz, C.: Web Services Orchestration and Choreography. IEEE Comp. **36**(10), 46–52 (2003)
46. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
47. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987)
48. Yi, Q., Liu, X., Bouguettaya, A., Medjahed, B.: Deploying and managing Web services: issues, solutions, and directions. VLDB J. **17**(3), 735–572 (2008)