Specifying Variability in Service Contracts

Davide Basile
Istituto di Scienza e Tecnologia dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche, Pisa, Italy
d.basile@isti.cnr.it

Felicita Di Giandomenico
Istituto di Scienza e Tecnologia dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche, Pisa, Italy
f.digiandomenico@isti.cnr.it

Stefania Gnesi
Istituto di Scienza e Tecnologia dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche, Pisa, Italy
s.gnesi@isti.cnr.it

Pierpaolo Degano
Dipartimento di Informatica
Università di Pisa, Italy
degano@unipi.it

Gian-Luigi Ferrari
Dipartimento di Informatica
Università di Pisa, Italy
giangi@unipi.it

ABSTRACT

In Service Oriented Computing (SOC) contracts characterise the behavioural conformance of a composition of services and guarantee that the composition does not lead to spurious results. Variability features can enable services to adapt to customer requirements and to changes in the context in which they execute.

We extend a recently introduced formal model of service contracts to specify variability mechanisms in a composition of services. Necessary and permitted service requests can be defined and triggered to increase adaptability. The compositional rules of the original formalism are enriched to fulfil all necessary requirements and the maximal number of permitted ones.

CCS Concepts

•Software and its engineering → **Software product lines;** *Formal software verification;* **•Information systems** → **Service discovery and interfaces;** **•Computing methodologies** → *Computational control theory;* **•Applied computing** → Electronic commerce;

Keywords

Services, Variability, Control Theory

1. INTRODUCTION

Nowadays software applications are formed by a heterogeneity of interacting entities connected through the Internet that can be updated to deliver new functionalities and adapt to changes in their operating environment. Service-oriented computing (SOC) [13] is a paradigm for building distributed interoperable applications by assembling fine-grained computational units, called services. Services are loosely coupled, reusable and platform-independent, that are built with little or no knowledge about clients and other services involved in their operating environment. Services can be composed to create largely distributed applications that can be delivered to either end-user applications or other services.

Vamos '17, February 01 - 03, Eindhoven, Netherlands

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4811-9/17/02...$15.00

DOI: http://dx.doi.org/10.1145/3023956.3023965

Service contracts [3] have been introduced in SOC to provide a behavioural description of services in terms of their obligations (or *offers*) and requirements (or *requests*). They are used to formally characterise a notion of *agreement* among the parties, that is a composition of services satisfying all service requirements through service obligations. Flexibility is fundamental to guarantee the adaptation of services to updates in the composed application. Adaptive mechanisms are used for the activation and deactivation of functionalities that are triggered by updates in the contract agreement. An agreement among contracts should guarantee the fulfilment of all *necessary* requirements and negotiate the maximum numbers of *permitted* requirements that can be fulfilled without spoiling the composition of services. Contracts adapt to the overall agreement by renouncing to those permitted requirements not satisfiable.

Service coordination dictates how services interact to realise the composite application. In particular, in an *orchestration* of services a distinguished component, called the orchestrator, drives their interactions to enforce only the behaviours in agreement. In a choreographed approach, services realise autonomously (i.e. without a central coordinator) the negotiated agreement.

Contract automata have been introduced in [7] as a formal model for service contracts; they represent either single services (called *principals*) or compositions of several services and are based both on orchestrated and choreographed coordination [4].

In this paper we introduce an adaptive formal model of contracts, within the orchestration approach. We extend contract automata with variability mechanisms for modelling adaptivity in contract composition, and we call them *modal service contract automata* (MSCA). Through MSCA a service exposes its offers and both necessary and permitted requirements. We describe different operators for composing MSCA resembling a static and a dynamic orchestration policy, and a notion of modal refinement of contracts useful for modelling adaptivity of services. A novel property of *modal agreement* modelling the fulfilment of all requests is described in language-theoretic terms, and a technique for synthesising an orchestration of services enjoying modal agreement is introduced. In particular, necessary requirements are naturally interpreted as uncontrollable actions in Control Theory [11], and the orchestrator is indeed the most permissive controller. The orchestration guarantees to negotiate an agreement among the parties such that all necessary requirements and the maximal number of permitted requirements can be eventually fulfilled. All the theory developed in this paper has been implemented in a prototypical tool available at https://github.com/davidebasile/CAT.

Structure of the paper. We start by illustrating our approach through a motivating example in Section 2. The formalisation of modal service contracts is described in Section 3, and a technique for synthesising a well-behaving orchestration of services is introduced in Section 4. In Section 5 we briefly discuss a prototypical tool developed for mechanising the theory presented in this paper, and Section 6 and Section 7 discuss related work, conclusion and future work, respectively.

2. MOTIVATING EXAMPLE

To illustrate our approach and help intuition, we consider a simple hotel reservation system. The system is composed of different hotel services and client services, that are displayed in Figure 1. Each service performs offer and request actions, that are depicted respectively as overlined and non-overlined labels on arcs; moreover the requests are divided into permitted (dotted arcs) and necessary. The goal of each service is to reach an accepting (final) state where its requests are matched by corresponding offers of other services. In particular, in order to reach an agreement among services, we require that all necessary requests that can *eventually* be fired and all the permitted requests that have been fired are matched. We are interested in generating an *orchestration* of services in agreement, that we recall to be a composition of services that interact with each other through a distinguished service, the orchestrator, which at run-time enforces the services agreement by regulating the evolution of the computation.

In Figure 1a the contract of Client1 is displayed; it starts by requiring to book a room, this request is modelled as *necessary*. When the room is selected, the client can decide to pay either by cash ($\overline{cashr}$) or credit card ($\overline{card}$). Assuming that the client is travelling for work, it *may* require a receipt from the hotel in order to be refunded from its organization. In particular, in case it will pay by credit card this request is *permitted*; indeed it can certify the payment through its bank service. Otherwise, if the client pays by cash then the receipt request is *necessary*, because this is the only way to prove that the payment has been made. Finally, in case the selected room is not single, the client *may* require a locker for securing its personal properties. In this case, the payment can only be performed by cash ($\overline{cashl}$, where l stands for locker), and hence a receipt *must* be provided before proceeding to the room payment.

The contract of Client2 is depicted in Figure 1b. This contract is similar to the one of Client1, except that Client2 will only pay by credit card and will not require a locker. The contract of Client3 is almost equal to Client1 and hence is not displayed: the only exception is in the locker request which is *necessary*, because Client3 is transporting high-value items.

In Figure 1c the contract of Hotel Service is depicted. This service starts by offering a room ($\overline{room}$) with a special offer of free breakfast ($\overline{freebrk}$) only for the first client. The hotel accepts payment either by credit card or cash and can offer a locker service ($\overline{locker}$). However, the hotel will not provide receipts in case of payment by cash. When a transaction is completed (i.e. a final state is reached), the hotel can start a new interaction with a different client: this time the free breakfast offer will not be available. Note that both requests of payment (card or cash) are permitted, but at least one of them must be satisfied in order to successfully terminate (that is the hotel service reaches an accepting state).

We will also consider a second hotel service Hotel Service 2 very similar to the first one (this service contract is not depicted), with the only exception that the payments by cash ($cashr$,$cashl$, here r stands for room) are *necessary*. Indeed, this second hotel service is experiencing issues with the credit card payment, and requires the clients to be always available to pay by cash, if necessary.

The composition of Client1 and Hotel Service in agreement (i.e. their orchestration) is displayed in Figure 1d. In the proposed agreement only the card payment is allowed, because if Client1 proceeds to pay through cash, its necessary request of receipt will not be fulfilled by Hotel Service. Similarly, the permitted locker request of Client1 is not admitted for the same reason. Note that we are only interested in ensuring that all requests are matched: even if Client1 does not require a breakfast, the presence of a free breakfast unmatched offer does not spoil the agreement property.

The orchestration of Client1,Hotel Service and Client2 is displayed in Figure 1e. In this example Client2 joins the composition after Client1, accordingly its request will be served after the one of Client1. A successful state can be reached only if all requests are matched: this requires one unfolding of the loop of Hotel Service.

If we add Client3 to the composition, then it *must* always be able to require a locker and the corresponding receipt. The Hotel Service *may* provide a locker but not the receipt. In this case the orchestrator cannot prevent Client3 from requiring a locker, and no intermediate accepting states are possible. Indeed, no agreement is possible and the orchestration will be *empty*. Similarly, if we swap Hotel Service with Hotel Service2, its necessary request of cash payment will not be fulfilled because no receipts are provided, and the resulting orchestration will be empty.

In the next sections we will present a novel formal model of service contracts capable of expressing the variability requirements discussed in this example, and an algorithm for composing services and synthesising, if possible, their orchestration in agreement.

3. FORMAL MODEL

We start by introducing some useful notation borrowed from [4, 7]. Let $\Sigma = \mathbb{R} \cup \mathbb{O} \cup \{\star\}$ be the alphabet of *basic actions*, made of *requests* $\mathbb{R} = \{a,b,c,\dots\}$ and *offers* $\mathbb{O} = \{\overline{a},\overline{b},\overline{c},\dots\}$ where $\mathbb{R} \cap \mathbb{O} = \emptyset$, and $\star \notin \mathbb{R} \cup \mathbb{O}$ is a distinguished element representing the *idle* move. We define the involution $co(\bullet) : \Sigma \mapsto \Sigma$ such that $co(\mathbb{R}) = \mathbb{O}$, $co(\mathbb{O}) = \mathbb{R}$, $co(\star) = \star$.

Let $\vec{v} = (a_1,...,a_n)$ be a vector of *rank* $n \geq 1$, in symbols r_v, and let $\vec{v}_{(i)}$ denote the i-th element with $1 \leq i \leq r_v$. We write $\vec{v}_1\vec{v}_2\dots\vec{v}_m$ for the concatenation of m vectors $\vec{v}_i$, while $|\vec{v}| = n$ is the rank (length) of $\vec{v}$ and $\vec{v}^n$ is the vector obtained by n concatenations of $\vec{v}$.

The alphabet of a contract automaton consists of vectors, each element of which intuitively records the activity, i.e. the occurrence of a basic action of a single principal in the contract. In a vector $\vec{v}$ there is either a single offer or a single request, or a single pair of request-offer that matches, i.e. there exists exactly i, j such that $\vec{v}_{(i)}$ is an offer and $\vec{v}_{(j)}$ is the complementary request or vice-versa; all the other elements of the vector contain the symbol $\star$, meaning that the corresponding principals stay idle. In the following let $\star^m$ denote a vector of rank m, all elements of which are $\star$. Formally:

DEFINITION 1 (ACTIONS). *Given a vector $\vec{a} \in \Sigma^n$, if*

- $\vec{a} = \star^{n_1}\alpha\star^{n_2}, n_1, n_2 \geq 0$, *then $\vec{a}$ is a* request (action) on α *if* $\alpha \in \mathbb{R}$, *and is an* offer (action) on α *if* $\alpha \in \mathbb{O}$
- $\vec{a} = \star^{n_1}\alpha\star^{n_2}co(\alpha)\star^{n_3}, n_1, n_2, n_3 \geq 0$, *then $\vec{a}$ is a* match (action) on α, *where* $\alpha \in \mathbb{R} \cup \mathbb{O}$.

Two actions $\vec{a}$ and $\vec{b}$ are complementary, *in symbols $\vec{a} \bowtie \vec{b}$ if and only if the following conditions hold: (i) $\exists \alpha \in \mathbb{R} \cup \mathbb{O}$: $\vec{a}$ is either a request or an offer on α; (ii) $\vec{a}$ is an offer on $\alpha \implies \vec{b}$ is a request on $co(\alpha)$ and (iii) $\vec{a}$ is a request on $\alpha \implies \vec{b}$ is an offer on $co(\alpha)$.*

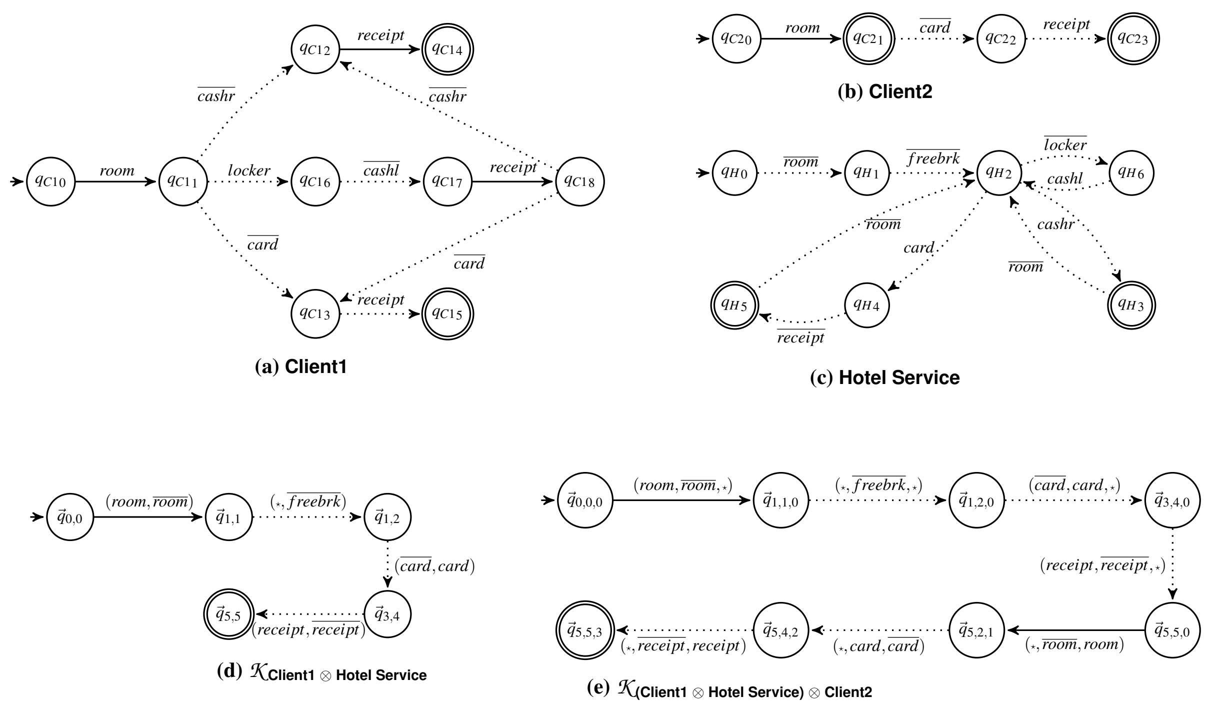
room
receipt
$\overline{cashr}$
$\overline{cashr}$
locker
$\overline{cashl}$
receipt
$\overline{card}$
$\overline{card}$
receipt
(a) Client1
room
$\overline{card}$
receipt
(b) Client2
$\overline{room}$
$\overline{freebrk}$
$\overline{locker}$
cashl
$\overline{room}$
card
cashr
$\overline{room}$
$\overline{receipt}$
(c) Hotel Service
$(room,\overline{room})$
$(\star,\overline{freebrk})$
$(\overline{card},card)$
$(receipt,\overline{receipt})$
(d) $\mathcal{K}_{\text{Client1} \otimes \text{Hotel Service}}$
$(room,\overline{room},\star)$
$(\star,\overline{freebrk},\star)$
$(\overline{card},card,\star)$
$(receipt,\overline{receipt},\star)$
$(\star,\overline{room},room)$
$(\star,card,\overline{card})$
$(\star,\overline{receipt},receipt)$
(e) $\mathcal{K}_{(\text{Client1} \otimes \text{Hotel Service}) \otimes \text{Client2}}$

Figure 1: The hotel reservation service

The actions and the states of contract automata are vectors of basic actions and of states of principals, respectively. In the following we will define modal service contract automata (MSCA) and their operators of composition, by extending those of [7].

Definition 2 (Modal Service Contract Automata). *Assume as given a finite set of states $\mathfrak{Q} = \{q_1, q_2, \ldots\}$. Then a* modal service contract automaton *$\mathcal{A}$, MSCA for short, of rank n is a tuple $\langle Q, \vec{q}_0, A^r, A^o, T^\diamond, T^\square, F\rangle$, where*

- *$Q = Q_1 \times \ldots \times Q_n \subseteq \mathfrak{Q}^n$;*
- *$\vec{q}_0 \in Q$ is the initial state;*
- *$A^r \subseteq \mathbb{R}, A^o \subseteq \mathbb{O}$ are finite sets (of requests and offers, respectively);*
- *$F \subseteq Q$ is the set of final states;*
- *$T^\diamond, \subseteq Q \times A \times Q$ and $T^\square \subseteq Q \times A \times Q$ are sets of respectively* may (permitted) *and* must (necessary) *transitions, where $A = (A^r \cup A^o \cup \{\star\})^n$;*
- *if $t = (\vec{q}, \vec{a}, \vec{q}\,') \in T$ where $T = T^\diamond \cup T^\square$ (abusing the notation $T^{\diamond\,\cup\,\square}$), then both the following conditions hold:*
  - *$\vec{a}$ is either a request or an offer or a match;*
  - *if $\vec{a}$ is an offer then $t \notin T^\square$;*
  - *$\forall i \in 1 \ldots n$. if $\vec{a}_{(i)} = \star$ then it must be $\vec{q}_{(i)} = \vec{q}\,'_{(i)}$.*

A principal *MSCA (or simply* principal*) has rank 1 and it is such that $A^r \cap co(A^o) = \emptyset$.*

To avoid cumbersome repetitions, from now onwards when not stated differently we assume a fixed modal service contract automaton $\mathcal{A} = \langle Q, \vec{q}_0, A^r, A^o, T^\diamond, T^\square, F\rangle$ of rank n, possibly indexed by a finite set I. Note that only requests and matches can be marked as necessary: we assume that a service contract can always withdraw its offers, because they are not necessary for reaching an agreement (see Section 4). The example in Section 2 can explain the intuition behind this design choice. If free breakfast is a necessary offer in `Hotel Service`, then we have the unrealistic scenario where the hotel contract rejects all clients' contracts. Indeed, no agreement is reached because no client is requiring free breakfast to match the offer, although they all are willing to pay for a room.

Modal service contract automata recognise traces of actions and modalities, through the language defined below.

Definition 3. *Let $\mathcal{A}$ be an MSCA. A step $(w, \vec{q}) \xrightarrow{\vec{a}\circ} (w', \vec{q}\,')$ with $\circ \in \{\diamond, \square\}$ occurs if and only if $w = \vec{a}\circ w', w' \in (A \cup \{\diamond, \square\})^*$ and $(\vec{q}, \vec{a}, \vec{q}\,') \in T^\circ$. Let $\rightarrow^*$ be the reflexive, transitive closure of the transition relation $\rightarrow$. The language of $\mathcal{A}$ is denoted as $\mathscr{L}(\mathcal{A}) = \{w \mid (w, \vec{q}_0) \xrightarrow{w}{}^* (\varepsilon, \vec{q}), \vec{q} \in F\}$. A step is denoted as $\vec{q} \xrightarrow{\vec{a}\circ}$ when w, w' and $\vec{q}\,'$ are immaterial and $(w, \vec{q}) \rightarrow (w', \vec{q}\,')$ when $\vec{a}\circ$ is immaterial.*

We now introduce our first (non-associative) operation of composition; recall that we implicitly assume the set of labels of an MSCA of rank m to be $A \subseteq (A^r \cup A^o \cup \{\star\})^m$.

Definition 4 (Product Composition). *Let $\mathcal{A}_i$ be MSCA of rank r_i. The* product composition *$\bigotimes_{i \in 1\ldots n} \mathcal{A}_i$ is the MSCA $\mathcal{A}$ of rank $m = \sum_{i \in 1 \ldots n} r_i$, where:*

- *$Q = Q_1 \times \ldots \times Q_n$, where $\vec{q}_0 = \vec{q}_{0_1} \cdots \vec{q}_{0_n}$*
- *$A^r = \bigcup_{i \in 1 \cdots n} A^r_i$, $A^o = \bigcup_{i \in 1 \cdots n} A^o_i$*
- *$F = \{\vec{q}_1 \ldots \vec{q}_n \mid \vec{q}_1 \ldots \vec{q}_n \in Q, \vec{q}_i \in F_i, i \in 1 \ldots n\}$*
- *let $\circ \in \{\diamond, \square\}$, T° is the least subset of $Q \times A \times Q$ s.t. $(\vec{q}, \vec{c}, \vec{q}\,') \in T^\circ$ iff, when $\vec{q} = \vec{q}_1 \ldots \vec{q}_n \in Q$,*

– *either there are* $1 \leq i < j \leq n$ *s.t.* $(\vec{q}_i, \vec{a}_i, \vec{q}'_i) \in T_i^{\circ}$, $(\vec{q}_j, \vec{a}_j, \vec{q}'_j) \in T_j^{\circ\,\cup\,\diamond}$, $\vec{a}_i \bowtie \vec{a}_j$ *and*

$$\begin{cases} \vec{c} = {}_{\star}{}^{u}\vec{a}_{i\star}{}^{v}\vec{a}_{j\star}{}^{z} \text{ with } u = r_1 + \ldots + r_{i-1}, \\ v = r_{i+1} + \ldots + r_{j-1}, |\vec{c}| = m \\ \text{and} \\ \vec{q}' = \vec{q}_1 \ldots \vec{q}_{i-1}\ \vec{q}'_i\ \vec{q}_{i+1} \ldots\ \vec{q}_{j-1}\ \vec{q}'_j\ \vec{q}_{j+1} \ldots \vec{q}_n \end{cases}$$

– *or there is* $1 \leq i \leq n$ *s.t.* $(\vec{q}_i, \vec{a}_i, \vec{q}'_i) \in T_i^{\circ}$ *and*

$$\begin{cases} \vec{c} = {}_{\star}{}^{u}\vec{a}_{i\star}{}^{v} \text{ with } u = r_1 + \ldots + r_{i-1}, \\ v = r_{i+1} + \ldots + r_n, |\vec{c}| = m, \\ \vec{q}' = \vec{q}_1 \ldots \vec{q}_{i-1}\ \vec{q}'_i\ \vec{q}_{i+1} \ldots \vec{q}_n \\ \text{and } \forall j \neq i, 1 \leq j \leq n, (\vec{q}_j, \vec{a}_j, \vec{q}'_j) \in T_j^{\circ\,\cup\,\diamond} \\ \text{it does not hold that } \vec{a}_i \bowtie \vec{a}_j \end{cases}$$

Note that the first case of the definition of transitions is for the matching of actions of two principals automata, while the other considers the action of a single component.

We use $\circ$ as a placeholder for both necessary ($\Box$) and permitted ($\diamond$) actions. In the product composition, permitted transitions ($\circ = \diamond$) are only generated from permitted transitions of their operands, while necessary transitions ($\circ = \Box$) are generated from both necessary and permitted transitions; i.e. when a necessary request matches a permitted offer the resulting match transition will be marked as necessary. Intuitively, the product composition interleaves the actions of all principals, with the only restriction that if two principals are ready to fire two complementary actions then only the match of them will be allowed and not their interleaving.

Example 1. In Figure1a, Figure 1b and Figure 1c three principals introduced in Section 2 are displayed. Two sub-portions of two different compositions are depicted in Figure 1d and Figure 1e. In Figure 1e the outgoing transition $\vec{q}_{0,0,0}\xrightarrow{(room,\overline{room},\star)\Box}$ is an example of a necessary match between the necessary request of the first principal and the permitted offer of the second principal. Moreover, considering the product composition (Client1 $\otimes$ Hotel Service) $\otimes$ Client2, the transitions $\vec{q}_{0,0,0}\xrightarrow{(room,\star,\star)\Box}$ or $\vec{q}_{0,0,0}\xrightarrow{(\star,\overline{room},\star)\diamond}$ are not allowed.

The next operator retrieves the principals involved in an MSCA obtained through the product introduced above, and identifies their original *may* and *must* transitions.

DEFINITION 5 (PROJECTION). *Let $\mathcal{A}$ be an MSCA, then the* projection *on the i-th principal is*
$\prod^i(\mathcal{A}) = \langle \prod^i(Q), \vec{q}_{0(i)}, \prod^i(A^r), \prod^i(A^o), \prod^i(T^{\diamond}), \prod^i(T^{\Box}), \prod^i(F) \rangle$
where $i \in 1 \ldots n$ *and:*

$$\begin{aligned} &\textstyle\prod^i(Q) = \{\vec{q}_{(i)} \mid \vec{q} \in Q\} \quad \prod^i(F) = \{\vec{q}_{(i)} \mid \vec{q} \in F\} \\ &\textstyle\prod^i(T^{\diamond}) = \{(\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q'}_{(i)}) \mid ((\vec{q}, \vec{a}, \vec{q'}) \in T^{\diamond} \wedge \vec{a}_{(i)} \notin \star) \\ &\qquad\qquad\qquad \vee ((\vec{q}, \vec{a}, \vec{q'}) \in T^{\Box} \wedge \vec{a}_{(i)} \in \mathbb{O})\} \\ &\textstyle\prod^i(T^{\Box}) = \{(\vec{q}_{(i)}, \vec{a}_{(i)}, \vec{q'}_{(i)}) \mid ((\vec{q}, \vec{a}, \vec{q'}) \in T^{\Box} \wedge \vec{a}_{(i)} \in \mathbb{R})\} \\ &\textstyle\prod^i(A^r) = \{a \mid a \in \mathbb{R}, (q, a, q') \in \prod^i(T)\} \\ &\textstyle\prod^i(A^o) = \{\overline{a} \mid \overline{a} \in \mathbb{O}, (q, \overline{a}, q') \in \prod^i(T)\} \end{aligned}$$

We now relate the two operators of product composition and projection by proving that principals are preserved by them.

PROPERTY 1. *Let $\mathcal{A}_i, i \in I$ be principal MSCAs, then*

$$\prod^j(\bigotimes_{i \in I} \mathcal{A}_i) = \mathcal{A}_j, j \in I$$

Example 2. Consider the MSCAs $\mathcal{K}_{\text{(Client1 ⊗ Hotel Service) ⊗ Client2}}$ in Figure 1e, Client1 in Figure 1a and Client2 in Figure 1b. We have that $\prod^3(\mathcal{K}_{\text{(Client1 ⊗ Hotel Service) ⊗ Client2}}) =$ Client2. Moreover, $\prod^1(\mathcal{K}_{\text{(Client1 ⊗ Hotel Service) ⊗ Client2}}) \neq$ Client1, while in general $\prod^1($ (Client1 $\otimes$ Hotel Service) $\otimes$ Client2$) =$ Client1 holds.

Our second (associative) composition operation first extracts from its operands the principals they are composed of through projection, and then reassembles them through product composition.

DEFINITION 6 (A-PRODUCT COMPOSITION). *Let $\mathcal{A}_1, \mathcal{A}_2$ be two MSCA of rank n and m, respectively, and let $I = \{\prod^i(\mathcal{A}_1) \mid 0 < i \leq n\} \cup \{\prod^j(\mathcal{A}_2) \mid 0 < j \leq m\}$. Then the* a-product composition *of $\mathcal{A}_1$ and $\mathcal{A}_2$ is $\mathcal{A}_1 \boxtimes \mathcal{A}_2 = \bigotimes_{\mathcal{A}_i \in I} \mathcal{A}_i$.*

The (non-associative $\otimes$ and associative $\boxtimes$) operators of composition model two different coordination policies among services. Through $\otimes$ a *static* composition is obtained: matched offers are not rearranged when new contracts join the composition. On the converse, a *dynamic* composition is obtained through $\boxtimes$: new services joining composite services can intercept already matched actions, as explained in the example below. Moreover, the dynamic composition is related to the static composition through projection.

Example 3. Let (Client1 $\otimes$ Hotel Service) $\boxtimes$ Client2 be a possible composition of the principals in Figure 1. In this composition the transition $\vec{q}_{0,0,0}\xrightarrow{(\star,\overline{room},room)\Box}$ is allowed, while it is not in the composition (Client1 $\otimes$ Hotel Service) $\otimes$ Client2.

From now onwards we assume that every MSCA $\mathcal{A}$ of rank $r_{\mathcal{A}} > 1$ is composed by MSCAs using the operators of composition described in this section. We introduce the refinement relation between MSCAs. Intuitively, an MSCA $\mathcal{A}_p$ refines an MSCA $\mathcal{A}$ when all the necessary transitions of $\mathcal{A}$ are maintained in $\mathcal{A}_p$ and only a subset of the permitted transitions of $\mathcal{A}$ are available in $\mathcal{A}_p$.

Firstly, we need the auxiliary notion of dangling state: an unreachable state or a state from which no final state can be reached.

DEFINITION 7 (DANGLING STATE). *Let $\mathcal{A}$ be an MSCA, then $\vec{q} \in Q$ is* dangling, *and belongs to the set Dangling($\mathcal{A}$), if $\nexists w.(w, \vec{q}_0) \rightarrow^* (\varepsilon, \vec{q}) \vee \forall \vec{q}_f \in F, \nexists w.(w, \vec{q}) \rightarrow^* (\varepsilon, \vec{q}_f)$.*

DEFINITION 8 (REFINEMENT OF MSCA). *An MSCA $\mathcal{A}_p$ is a* refinement *of an MSCA $\mathcal{A}$ written $\mathcal{A}_p \preceq \mathcal{A}$, if and only if there exists a refinement relation $\mathcal{R} \subseteq Q \times Q_p$ such that $A_p^r \subseteq A^r, A_p^o \subseteq A^o$, $(\vec{q}_0, \vec{q}_{0p}) \in \mathcal{R}$ and for all $(\vec{q}, \vec{q}_p) \in \mathcal{R}$ the following holds:*

- $(\vec{q}, \vec{a}, \vec{q}') \in T^{\Box}$ *iff* $\exists \vec{q}'_p \in Q_p.(\vec{q}_p, \vec{a}, \vec{q}'_p) \in T_p^{\Box} \wedge (\vec{q}', \vec{q}'_p) \in \mathcal{R}$;
- *if* $(\vec{q}_p, \vec{a}, \vec{q}'_p) \in T^{\diamond}$ *then* $\exists \vec{q}' \in Q.(\vec{q}, \vec{a}, \vec{q}') \in T_p^{\diamond} \wedge (\vec{q}', \vec{q}'_p) \in \mathcal{R}$;
- $\vec{q} \notin Dangling(\mathcal{A}) \wedge \vec{q}_p \notin Dangling(\mathcal{A}_p)$.

Example 4. Consider the MSCAs $\mathcal{K}_{\text{(Client1 ⊗ Hotel Service) ⊗ Client2}}$ and Client1 in Figure 1e and Figure 1a respectively, let Client1$_p$ be the MSCA $\prod^1(\mathcal{K}_{\text{(Client1 ⊗ Hotel Service) ⊗ Client2}})$. The MSCA Client1$_p$ has the set of states $\{q_{C1_0}, q_{C1_1}, q_{C1_3}, q_{C1_5}\}$, the set of final states $\{q_{C1_5}\}$ and the language $\mathscr{L}$(Client1$_p$) $= \{room\Box\ \overline{card}\diamond\ receipt\diamond\}$. Finally, the relation Client1$_p \preceq$ Client1 holds.

In the next section a notion of well-formed composition of services and a technique for synthesising it are described.

4. SYNTHESIS OF ORCHESTRATION

In this section we introduce the notion of *modal agreement* (called agreement for brevity) and a technique for synthesising an orchestration of services in agreement. We start by introducing the notion of modal agreement as a property of the language recognised by an MSCA; intuitively a trace is in agreement if it is a concatenation of matches, offer actions and their modalities.

DEFINITION 9 (MODAL AGREEMENT). *A trace accepted by an MSCA is in* agreement *if it belongs to the set*

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

An MSCA is safe when all the traces of its language are in agreement, and admits agreement when at least one of its traces is in agreement. Formally:

DEFINITION 10 (MODAL SAFETY). *An MSCA $\mathcal{A}$ is* (modal) safe *if $\mathscr{L}(\mathcal{A}) \subseteq \mathfrak{A}$, otherwise it is* unsafe.
Additionally, if $\mathscr{L}(\mathcal{A}) \cap \mathfrak{A} \neq \emptyset$ then $\mathcal{A}$ admits (modal) agreement.

Example 5. Consider the MSCA $\mathcal{K}_{\mathsf{Client1} \otimes \mathsf{Hotel\ Service}}$ in Figure 1d. Its language is $\mathscr{L}(\mathcal{K}_{\mathsf{Client1} \otimes \mathsf{Hotel\ Service}}) = \{(room, \overline{room})\Box (\star, \overline{freebrk})\diamond(\overline{card}, card)\diamond(receipt, \overline{receipt})\diamond\}$. This MSCA is safe, because $\mathscr{L}(\mathcal{K}_{\mathsf{Client1} \otimes \mathsf{Hotel\ Service}}) \subseteq \mathfrak{A}$.

We now introduce an algorithm for synthesising an orchestration of MSCAs, that is the maximal sub-portion of an MSCA $\mathcal{A}$ that is safe. In particular, the orchestrator will be the *most permissive controller* (*mpc*) in the style of the supervisory control for discrete event systems [11]. A discrete event system is a finite state automaton, where *marked*(i.e. final) states represent the successful termination of a task, while *forbidden* states should never be traversed in "good" computations. Generally, the purpose of supervisory control theory is to synthesise a controller that enforces good computations. To do so, this theory distinguishes between *controllable* events (those the controller can disable) and *uncontrollable* events (those always enabled), besides partitioning events into *observable* and *unobservable* (obviously uncontrollable). If all events are observable then a most permissive controller exists that never blocks a good computation [11]. The purpose of contracts is to declare all the activities of a principal in terms of requests and offers. Therefore all the actions of a (composed) contract are observable.

Clearly, the behaviours that we want to enforce upon a given MSCA are exactly those traces in agreement, and so we assume that permitted requests and necessary requests that cannot be matched (called unmatchable) lead to a forbidden state.

The introduction of variability in MSCAs plays a crucial role in the synthesis of orchestration (see Theorem 1 below). Indeed, while in [7] all actions are controllable, the presence of necessary requests calls for a tailored notion of uncontrollable actions in the *mpc* and a new synthesis algorithm, as exemplified below.

Example 6. Assume that all actions (necessary and permitted) are controllable. The orchestration of `Client3` $\otimes$ `Hotel Service` would then admit agreement. In fact, it would be the same of the orchestration `Client1` $\otimes$ `Hotel Service`. Clearly, the *necessary* locker requirement of `Client3` is ignored by the orchestrator.

In order to avoid the situation described in Example 6 and fulfil the modalities imposed by an MSCA, we force a composition to be in agreement only if there exists a match for each necessary request. Therefore, we assume that the *unmatchable* necessary requests are *uncontrollable*, i.e. they cannot be disabled by the controller. Now the scenario in Example 6 does not hold any more, because the necessary locker request cannot be ignored by the orchestrator.

We only allow matchable necessary requests (see Definition 11) and permitted transitions to be controllable: they can be disabled by the controller without spoiling the overall agreement. Intuitively, if a necessary request r is matchable there exists a transition in the composition where r is matched (Definition 8 is updated by considering as optional the necessary matchable requests). For defining and computing the most permissive controller, we firstly need to formally introduce the notions of matched, unmatched (*must* request) transitions, and uncontrollable disagreement. A request transition t is matched in an MSCA $\mathcal{A}$ if there exists a necessary match transition t' in $\mathcal{A}$ where in both t and t' the same request is performed by the same principal that is in the same internal state, and the source state of t' is not dangling. If this is not the case, then t is unmatched. Intuitively, the request is matched somewhere in the MSCA. Formally:

DEFINITION 11 (MATCHED TRANSITION). *Let $\mathcal{A}$ be an MSCA and $t = (\vec{q}_1, \vec{a}_1, \vec{q}\,'_1)$ be a (must) request transition, then t is* matched *in $\mathcal{A}$ iff $\exists(\vec{q}_2, \vec{a}_2, \vec{q}\,'_2)$ match $\in T^{\Box}$ s.t. $\vec{q}_2 \notin Dangling(\mathcal{A})$, $\vec{q}_{1(i)} = \vec{q}_{2(i)}, \vec{a}_{1(i)} = \vec{a}_{2(i)} \in \mathbb{R}$; otherwise t is* unmatched *in $\mathcal{A}$.*

Example 7. Consider again the example in Section 2, and the product composition $\mathcal{A} = \mathsf{Client3} \otimes \mathsf{Hotel\ Service}$. For displaying purposes, we do not depict the whole product composition. A necessary match transition that is reachable in $\mathcal{A}$ through the trace $(room, \overline{room})\Box\ (\star, \overline{freebrk})\diamond$ is $t = (\ (q_{C3_1}, q_{H2}),\ (locker, \overline{locker}),\ (q_{C3_6}, q_{H6})\ )$. Moreover, through the trace $(room, \overline{room})\Box$ the necessary request transition $t' = (\ (q_{C3_1}, q_{H1}), (locker, \star), (q_{C3_6}, q_{H1})\ )$ is also reachable in $\mathcal{A}$. For the above reasons, the transition t' is matchable in $\mathcal{A}$, while t' is unmatchable in the modal service contract automaton $\mathcal{K}_{\mathsf{Client3} \otimes \mathsf{Hotel\ Service}}$ that is actually empty.

A state is in uncontrollable disagreement if the controller cannot avoid an unmatched request to be eventually fired.

DEFINITION 12 (UNCONTROLLABLE DISAGREEMENT). *Let $\mathcal{A}$ be an MSCA, a transition $\vec{q} \xrightarrow{\vec{a}}$ is* forced *in $\mathcal{A}$ if $\vec{a}$ is either (i) a necessary match or unmatched request or (ii) the only outgoing action and $\vec{q} \notin F$. A state $\vec{q}$ of $\mathcal{A}$ is in* uncontrollable disagreement *if $\vec{q} \notin Dangling(\mathcal{A})$ and there exists a trace w such that $\vec{q} \xrightarrow{w}{}^* \vec{q}_1$ by only firing forced transitions, and $w \notin \mathfrak{A}$ or $\vec{q}_1 \in Dangling(\mathcal{A})$.*

Note that also final states can be in uncontrollable disagreement. The most permissive modal controller allows (1) all traces in agreement where (2) no states in uncontrollable disagreement are traversed, and blocks those traces not satisfying (1) or (2). Note that a controller of an MSCA $\mathcal{A}$ (obtained through the operators of Section 3) is also an MSCA, and that the most permissive controller is unique up to language equivalence.

DEFINITION 13 (MODAL CONTROLLER). *Let $\mathcal{A}$ and $\mathcal{K}$ be MSCAs, we call $\mathcal{K}$* modal controller *of $\mathcal{A}$ if and only if it holds that $\mathscr{L}(\mathcal{K}) \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A})$, $Dangling(\mathcal{K}) = \emptyset$, and there is no trace $w \in \mathscr{L}(\mathcal{K})$ such that $(w, \vec{q}_{0\,\mathcal{K}}) \xrightarrow{w}{}^* (\varepsilon, \vec{q}_{\mathcal{K}})$, $(w, \vec{q}_0) \xrightarrow{w}{}^* (\varepsilon, \vec{q})$ and $\vec{q}$ is in uncontrollable disagreement. A controller $\mathcal{K}$ of $\mathcal{A}$ is the* most permissive (modal) controller (mpc) *if and only if for all $\mathcal{K}'$ controller of $\mathcal{A}$, $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K})$ holds.*

We are now ready to describe the algorithm for computing the most permissive controller of an MSCA $\mathcal{A}$. The *mpc* is computed through an iterative procedure that at each step i updates incrementally a set of states R_i and revises an MSCA $\mathcal{K}_i$. The algorithm terminates when no more updates are possible.

Intuitively, the property of agreement requires that all requests are matched. Hence, we want to remove all possible unmatched requests. This is straightforward for permitted requests, because they are controllable, while for the necessary requests we can only remove those requests that are matched somewhere in the MSCA, and try to make the unmatched necessary requests unreachable. For this purpose, the sets R_i contain the "bad" states, that are those that cannot prevent an unmatched necessary request to be eventually fired (i.e. states in uncontrollable disagreement). Note that by pruning transitions, a matched request may become unmatched if all the available matches become unreachable.

The algorithm starts with an MSCA $\mathcal{K}_0$ obtained from $\mathcal{A}$ by pruning all permitted request transitions and all necessary request transitions that are matched in $\mathcal{A}$. The starting set R_0 contains all dangling states of $\mathcal{K}_0$ and all source states of *must* transitions of $\mathcal{A}$ that are unmatched in $\mathcal{K}_0$. At each step i the algorithm prunes in a backwards fashion *may* transitions with target state in R_i and *must* transitions with source state in R_i. The set R_i is updated by adding (1) source states of *must* transitions t of $\mathcal{K}_i$ not previously added with dangling target state, and (2) source states of *must* transitions t of $\mathcal{A}$ that are unmatched in $\mathcal{K}_i$. At termination of this procedure, if the initial state of $\mathcal{A}$ is in R_i then the most permissive controller is empty: $\mathcal{A}$ does not admit agreement. Otherwise, the revised MSCA $\mathcal{K}_i$ is the most permissive controller of $\mathcal{A}$.

Since the set R_i is finite and can only increase at each step, the termination of the algorithm is ensured. By an abuse of notation, in the following we denote with *MSCA* the set of all possible modal service contract automata. The algorithm for computing the most permissive controller is formally defined below.

THEOREM 1 (MPC SYNTHESIS). *Let $\mathcal{A}$ be an MSCA, $f : MSCA \times 2^Q \to MSCA \times 2^Q$ be a monotone function on the cpo $P = (2^Q, \subseteq)$. Moreover, let $\mathcal{K}_0 = \langle Q, \vec{q_0}, A^r, A^o, T^\diamond \setminus \{t \in T^\diamond \mid t$ is a request transition $\}, T^\Box \setminus \{t \in T^\Box \mid t$ is matched in $\mathcal{A}\}, F\rangle$,*
$R_0 = Dangling(\mathcal{K}_0) \cup \{\vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^\Box_{\mathcal{A}}$ is unmatched in $\mathcal{K}_0\}$,
$f(\mathcal{K}_{i-1}, R_{\,i-1}) = (\mathcal{K}_i, R_{\,i})$ where:

- $\mathcal{K}_i = \langle Q, \vec{q_0}, A^r, A^o,$
  $T^\diamond_{\mathcal{K}_{i-1}} \setminus \{(\vec{q}, \vec{a}, \vec{q}') \in T^\diamond_{\mathcal{K}_{i-1}} \mid \vec{q}' \in R_{\,i-1}\},$
  $T^\Box_{\mathcal{K}_{i-1}} \setminus \{(\vec{q}, \vec{a}, \vec{q}') \in T^\Box_{\mathcal{K}_{i-1}} \mid \vec{q} \in R_{\,i-1})\}, F\rangle$
- $R_{\,i} = R_{\,i-1} \cup \{\vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^\Box_{\mathcal{K}_i} . \vec{q} \notin R_{i-1}, \vec{q}' \in Dangling(\mathcal{K}_i)\}$
  $\cup \{\vec{q} \mid (\vec{q}, \vec{a}, \vec{q}') \in T^\Box_{\mathcal{A}}$ *is unmatched in* $\mathcal{K}_i\} \cup Dangling(\mathcal{K}_i)$;

and $(\mathcal{K}_n, R_n) = sup(\{f^n(\mathcal{K}_0, R_0) \mid n \in N\})$ is the least fixed point of f. Then, the mpc *of $\mathcal{A}$ is computed as:*

$$\textit{if } \vec{q_0} \in R_n \textit{ then } \mathcal{K}_{\mathcal{A}} = \langle\rangle, \textit{ else}$$

$$\mathcal{K}_{\mathcal{A}} = \langle Q \setminus Dangling(\mathcal{K}_n), \vec{q_0}, A^r, A^o,$$

$$T^\diamond_{\mathcal{K}_n}, T^\Box_{\mathcal{K}_n}, F \setminus Dangling(\mathcal{K}_n)\rangle$$

PROOF. We will prove that the algorithm always terminates, that $\mathcal{K}_{\mathcal{A}}$ is a controller of $\mathcal{A}$ and in particular it is the most permissive controller of $\mathcal{A}$.

We ensure that the algorithm always terminates, by proving the existence of the least fixed point of f. The function f is monotonic because it is the defined on the cpo P and at each iteration the set R_i can only increase, hence by Knaster-Tarski theorem the least fixed point of f exists.

We now prove that $\mathcal{K}_{\mathcal{A}}$ is a controller of $\mathcal{A}$, that is (1) $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathscr{L}(\mathcal{A}) \cap \mathfrak{A}$ and (2) $(w, \vec{q_0}_{\mathcal{K}}) \xrightarrow{w} (\varepsilon, \vec{q}_{\mathcal{K}})$, $(w, \vec{q_0}) \xrightarrow{w} (\varepsilon, \vec{q})$ with $\vec{q}$ in uncontrollable disagreement does not hold.

We start by proving (1). Since $\mathcal{K}_{\mathcal{A}}$ is derived from $\mathcal{A}$ by pruning transitions, trivially $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathscr{L}(\mathcal{A})$.

For proving $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subseteq \mathfrak{A}$ we have to show that no trace w' recognised by $\mathscr{L}(\mathcal{K}_{\mathcal{A}})$ contains a request $\vec{a}$. We observe that the algorithm only prunes and never adds transitions, and since in $\mathcal{K}_0$ all permitted requests are pruned, $\vec{a}$ cannot be a permitted request.

By contradiction assume that $\vec{a}$ is a necessary request and w' is recognised by $\mathcal{K}_{\mathcal{A}}$. We note that $\vec{a}$ cannot be matched otherwise it would have been pruned in $\mathcal{K}_0$, consequently $\vec{a}$ is an unmatched necessary request and by definition we have $\vec{q} \in R_n$ where $\vec{q} \xrightarrow{\vec{a}\Box}$. Let $Q_{w'} = \{\vec{q_0}, \vec{q_1}, \ldots, \vec{q_m}\}$ be the sequence of states visited by $\mathcal{K}_{\mathcal{A}}$ for recognising w', we have $\vec{q} \in Q_{w'}$. Moreover, by definition of R_i it must be that $Q_{w'} \subseteq R_n$, and in particular $\vec{q_0} \in R_n$ and we reach the contradiction $\mathcal{K}_{\mathcal{A}} = \langle\rangle$.

We now prove (2). The argument is similar to the previous point. Assume $(w, \vec{q_0}_{\mathcal{K}}) \xrightarrow{w} (\varepsilon, \vec{q}_{\mathcal{K}})$ and $(w, \vec{q_0}) \xrightarrow{w} (\varepsilon, \vec{q})$ with $\vec{q}$ in uncontrollable disagreement holds. Since $\mathcal{K}_{\mathcal{A}}$ is derived from $\mathcal{A}$, we have $\vec{q_0}_{\mathcal{K}} = \vec{q_0}$ and $\vec{q}_{\mathcal{K}} = \vec{q}$. Let $Q_w = \{\vec{q_0}, \vec{q_1}, \ldots, \vec{q}\}$ be the sequence of states visited by $\mathcal{K}_{\mathcal{A}}$ for recognising w. Since $\vec{q}$ is in uncontrollable disagreement and is reachable in $\mathcal{K}_{\mathcal{A}}$, by definition of R_i it must be that $Q_w \subseteq R_n$, and in particular $\vec{q_0} \in R_n$ and we reach the contradiction $\mathcal{K}_{\mathcal{A}} = \langle\rangle$.

It remains to prove that $\mathcal{K}_{\mathcal{A}}$ is the most permissive controller. By contradiction assume $\mathcal{K}'$ to be a controller of $\mathcal{A}$ such that it holds $\mathscr{L}(\mathcal{K}_{\mathcal{A}}) \subset \mathscr{L}(\mathcal{K}')$. Hence there must be a trace $w_2 \in \mathscr{L}(\mathcal{K}'), w_2 \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}})$, and let $Q_{w_2} = \{\vec{q_0}, \vec{q_1}, \ldots, \vec{q_m}\}$ be the sequence of states visited by $\mathcal{K}'$ for recognising w_2. We have that the set $Q_p = Q_{w_2} \setminus Q_{\mathcal{K}_{\mathcal{A}}}$ is non-empty. Moreover, $Q_p \subseteq R_n$, otherwise we would obtain $w_2 \in \mathscr{L}(\mathcal{K}_{\mathcal{A}})$. Let $\vec{q}_p \in Q_p$ be one of the states traversed by $\mathcal{K}'$ for recognising w_2, by definition of R_i and Definition 12 $\vec{q}_p$ is a state of $\mathcal{A}$ in uncontrollable disagreement, hence $\mathcal{K}'$ is not a controller of $\mathcal{A}$, contradiction. □

Example 8. Consider the MSCAs of the example in Section 2. The *mpc* of Client1 $\otimes$ Hotel Service is $\mathcal{K}_{\text{Client1} \otimes \text{Hotel Service}}$; and the *mpc* of (Client1 $\otimes$ Hotel Service) $\otimes$ Client2 is the automaton $\mathcal{K}_{(\text{Client1} \otimes \text{Hotel Service}) \otimes \text{Client2}}$, both computed through Theorem 1.

Let $\mathcal{A}$ = Client3 $\otimes$ Hotel Service, to illustrate the algorithm in Theorem 1 we describe the iterations for computing the corresponding *mpc*, that we recall to be empty (see Section 2).

Initially, in $\mathcal{K}_0$ all possible *may* requests of $\mathcal{A}$ are removed (recall that in $\mathcal{A}$ we have interleavings of actions of Client3 and Hotel Service), as well as all the matched *must* requests. In particular, the *must* requests $q_{C3_2} \xrightarrow{receipt\Box}$ and $q_{C3_7} \xrightarrow{receipt\Box}$ are matched in $\mathcal{A}$ when Hotel Service is in state q_{H4}, and the source states of these match transitions are not dangling in $\mathcal{A}$. For example $(w, (q_{C3_0}, q_{H0})) \rightarrow^* (\varepsilon, (q_{C3_4}, q_{H5}))$ with $w = (room, \overline{room})\Box$ $(\overline{cashr}, \star)\diamond$ $(\star, \overline{freebrk})\diamond$ $(\star, card)\diamond (receipt, \overline{receipt})\Box$.

The set R_0 contains all dangling states of $\mathcal{K}_0$. The sources of unmatched transitions of $\mathcal{A}$ in $\mathcal{K}_0$ are all the possible combinations of states where the first principal (Client3) is in states q_{C3_2} or q_{C3_7}. Indeed, these states have become dangling in $\mathcal{K}_0$.

In the first iteration, all *may* transitions of $\mathcal{K}_0$ leading to states in R_0 are removed in $\mathcal{K}_1$. In particular, the (dangling) matches in $\mathcal{K}_0$ that are removed are $((q_{C3_8}, q_{H2}), (\overline{cashr}, cashr), (q_{C3_2}, q_{H3}))$, $((q_{C3_6}, q_{H6}), (\overline{cashl}, cashl), (q_{C3_7}, q_{H2}))$ and the transition $((q_{C3_1}, q_{H2}), (\overline{cashr}, cashr), (q_{C3_2}, q_{H3}))$.

The *must* transitions with source state in R_0 are also removed, in particular they are: $((q_{C3_7}, q_{H4}), (receipt, \overline{receipt}), (q_{C3_8}, q_{H5}))$ and $((q_{C3_2}, q_{H4}), (receipt, \overline{receipt}), (q_{C3_4}, q_{H5}))$.

The set R_1 is updated with the state (q_{C3_1}, q_{H2}) and no new unmatched transitions are generated.

In the second iteration, there are one *may* transition and one *must* transition in $\mathcal{K}_1$ with respectively target and source state in R_1 and in particular (q_{C3_1}, q_{H2}) (the state added at the previous iteration). Consequently, the transitions to be removed from $\mathcal{K}_1$ are the *must* transition $((q_{C3_1}, q_{H2}), (locker, \overline{locker}), (q_{C3_6}, q_{H6}))$ and the *may* transition $((q_{C3_1}, q_{H1}), (\star, \overline{freebrk}), (q_{C3_1}, q_{H2}))$. This removal causes the state (q_{C3_0}, q_{H0}) to be added to R_2. Moreover, a newly generated unmatched transition of the MSCA $\mathcal{A}$ in $\mathcal{K}_2$ is $((q_{C3_1}, q_{H1}), (locker, \star), (q_{C3_6}, q_{H1}))$, because the only match of the locker request remained in $\mathcal{K}_1$ has been removed.

The third (last) iteration removes the outgoing *must* transition $((q_{C3_0}, q_{H0}), (room, \overline{room}), (q_{C3_1}, q_{H1}))$, because the source state has been added to R_2 in the previous iteration.

No more updates are possible for $\mathcal{K}_3$ and the algorithm returns an empty *mpc* because the initial state of $\mathcal{K}_3$, i.e. (q_{C3_0}, q_{H0}), is in R_3 (it has been added during the second iteration).

Note that if the synthesis succeeds (i.e. the *mpc* is not empty) then all the (combinations of) variation points that guarantee agreement are exactly the (combinations of) permitted requests of principals in the *mpc*. Conversely, the permitted actions disabled by the *mpc* are the variation points to be removed for guaranteeing agreement. The following theorem is based on the notion of modal refinement in Definition 8 and it states that the most permissive controller of an MSCA $\mathcal{A}$ produces the largest refinement of the principals in $\mathcal{A}$ such that an agreement among the parties is possible. Intuitively, if a permitted action does not spoil the overall agreement then it will be available in the composition of services.

THEOREM 2 (LARGEST REFINEMENT). *Let $\mathcal{A} = \otimes_{i \in I} \mathcal{A}_i$ be a product composition of principal MSCAs $\mathcal{A}_i$, let $\mathcal{K}_{\mathcal{A}}$ be its non-empty mpc computed through Theorem 1 and let $\Pi_i(\mathcal{K}_{\mathcal{A}}) = \mathcal{A}_{p_i}$ be its projections on i-th principals. The following holds:*

$$\forall i \in I \, . \, \mathcal{A}_{p_i} \preceq \mathcal{A}_i \tag{1}$$

$$\forall \mathcal{K}' \text{ controller of } \mathcal{A}, \forall i . \Pi_i(\mathcal{K}') = \mathcal{A}_{p_i'} \preceq \mathcal{A}_{p_i} \tag{2}$$

PROOF. We start by proving (1). By Theorem 1 we know that $Q_{mpc} \subseteq Q$, $Q_{mpc} \cap Dangling(\mathcal{A}) = \emptyset$ and $T^{\diamond}_{mpc} \subseteq T^{\diamond}$. Given a generic principal $\mathcal{A}_i$, it remains to prove that all reachable necessary transitions of $\mathcal{A}_i$ are available in $\mathcal{A}_{p_i}$. By contradiction, let q be a reachable state of both $\mathcal{A}_i$ and $\mathcal{A}_{p_i}$, and $t = (q, a, q')$ be such that $t \in T^{\square}_{\mathcal{A}_i}, t \notin T^{\square}_{\mathcal{A}_{p_i}}$. Let $Q' = \{\vec{q} \mid \vec{q} \in Q_{\mathcal{A}}, \vec{q}_{(i)} = q\} \cap Q_{mpc}$. We have two cases: if there exists $\vec{q} \in Q'$ in uncontrollable disagreement then $\mathcal{K}_{\mathcal{A}}$ is not a controller of $\mathcal{A}$, contradiction. Otherwise, there exists $(\vec{q}, \vec{a}, \vec{q}') \in T^{\square}_{\mathcal{A}}$ with $\vec{a}_{(i)} = a$ and $\vec{q}'$ not in uncontrollable disagreement, and since $\mathcal{K}_{\mathcal{A}}$ is the most permissive controller by Theorem 1 it must be $t \in T^{\square}_{\mathcal{A}_{p_i}}$, contradiction.

For proving item (2) by contradiction assume that for some i we have $\mathcal{A}_{p_i} \prec \mathcal{A}_{p_i'} = \Pi_i(\mathcal{K}')$. It follows that there exists a trace $w \in \mathscr{L}(\mathcal{K}') \subseteq \mathfrak{A} \cap \mathscr{L}(\mathcal{A}), w \notin \mathscr{L}(\mathcal{K}_{\mathcal{A}})$, and no states in uncontrollable disagreement are traversed by $\mathcal{K}'$ for recognising w. Since $\mathcal{K}_{\mathcal{A}}$ is the most permissive controller of $\mathcal{A}$, we have $\mathscr{L}(\mathcal{K}') \subseteq \mathscr{L}(\mathcal{K}_{\mathcal{A}})$, contradiction. $\square$

The theory presented in this paper has been implemented in a prototypical tool called MSCA tool, that is open-source and available online. The MSCA tool is described in the next section.

5. IMPLEMENTATION

In this section we briefly introduce the prototypical tool that has been implemented for mechanising the algorithms proposed in this paper and proving the effectiveness of our proposal. With the MSCA tool (MSCAT) it is possible to create, load and store MSCAs, and to compose several MSCAs with the two composition operators described in Section 3. The algorithm in Theorem 1 has been implemented in MSCAT and it is possible to compute the most permissive controller of a composition of MSCAs.

The MSCAT is based on the API of the Contract Automata Tool (CAT) [8], a tool available online for managing contract automata. In particular, the API of MSCAT extends those of CAT. It is implemented in Java and it is available online at https://github.com/davidebasile/CAT.

This tool is still under development and its functionalities can be invoked through command line. All the examples in this paper have been computed through MSCAT, and are available online. In particular, the MSCAs of the example in Section 2 can be found at https://github.com/davidebasile/CAT/tree/master/JaMata/MSCAexamples/hotel_reservation. For example, the controller $\mathcal{K}_{\texttt{Client1} \otimes \texttt{Hotel Service}}$ in Figure 1d that is stored in the file `K_C1xSH.data` is:

```
Rank: 2 ,  Number of states: [9, 7], Initial state:
[0, 0], Final states: [[4, 5][3, 5]], Transitions:
!([0, 0],[-1, 1],[1, 1]) ([1, 2],[3, -3],[3, 4])
([3, 4],[-5, 5],[5, 5])  ([1, 1],[0, 4],[1, 2])
```

This automaton has been computed automatically through the method `public MSCA mpc()` of the class `MSCA.java` in less than a second. The file contains the rank of the automaton, the total number of states of each principal, the initial state and the final states of each principal (the final states of the whole automaton are the combination of final states of the principals). The MSCA has four transitions, three of them are permitted and one is necessary. In particular the necessary transition is prefixed by `!`. The states and actions are encoded into integers, for example the encoding of the offer action $\overline{room}$ is the number 1, while the corresponding request is -1. The actions $freebrk, card, receipt$ are respectively encoded into $4, 3, 5$, and the idle move is encoded into zero. A more user-friendly GUI is under development. More information about MSCAT can be found in the documentation available online.

6. RELATED WORK

The MSCA formalism is an extension of a recent formal model of service contracts called contract automata (CA) [7]. By comparing MSCA and CA, the main difference is the possibility of expressing necessary and permitted requirements, a feature not available in CA. Indeed, in an orchestration of CAs all actions are controllable, that is necessary requirements cannot be expressed. Contract automata have been used for studying other important issues arising in a composition of service contracts. The problem of circular dependencies among contracts has been investigated through the property of *weak agreement* of CAs. This property intuitively admits traces where requests can be recorded as debits provided that in the future the corresponding offers will be honoured, and the verification techniques are based on optimisation of network flow problems [7]. The above properties have been studied for both competitive and collaborative contracts, stating that generally safety is preserved in collaborative contracts and not in competitive ones. This model has also been related to two intuitionistic logics introduced for modelling circular dependencies among contracts in [7]. Orchestrations of CAs have been related to an automata-based model of choreographies in [4], by identifying the conditions for dismissing the central orchestrator for both synchronous and asynchronous choreographies. For thorough comparisons with other formalisms for service contracts, we refer the reader to [7].

The problem of modelling and analysing variability in product families has been investigated in [9]. In product families, applications are derived from generic behavioural descriptions by specifying which features will be available and which not. Product families are formally represented as labelled transition systems endowed with *may* and *must* transitions, representing both permitted and necessary features that all derived software share. Additional variability constraints not directly definable are specified through an action-based branching time temporal logic enriched with variability constructs called v-ACTL. The logic is based on a deontic characterisation of feature models [1], specifically characterising those permitted and necessary features of a product family; and techniques for verifying these constraints on product families are introduced. Compared to our approach, *may* and *must* transitions of MSCA model necessary and permitted requirements of service contracts, and are not used for modelling features of SOC software families. Moreover, in [9] the focus is on guaranteeing that variability constraints satisfied in a product family are also satisfied for all derived products, through model checking. We do not focus on checking that the property of agreement is satisfied (that is, the MSCA $\mathcal{A}$ is modal safe), which can be trivially obtained by verifying that the most permissive controller of $\mathcal{A}$ is $\mathcal{A}$ itself [7]. We study the problem of composing several service contracts with variability requirements and synthesising their orchestration, that is the largest sub-portion of the composed automaton that is modal safe.

Supervisory Control Theory has been applied to Software Product Line Engineering in [10]. The `CIF3` toolset is used to synthesise all the valid products of a family composed of behavioural components and behavioural requirements rendered as finite state automata. Additional constraints are generated from an attribute feature model and other behavioural requirements (e.g. guards on events, state invariants). Unlike here, the authors consider all actions to be controllable, and necessary features are behavioural requirements rendered as additional constraints. A further difference is that synthesis of services orchestration is not studied.

Modal I/O Automata (MIOA) are introduced in [12] as an extension of interface automata for modelling product lines in component-based software engineering. An MIOA is a finite state automaton with *may* and *must* transitions labelled by input, output and internal actions. The authors prove that alternating simulation of interface automata coincides with modal refinement of modal transition systems. Compared to MSCA, MIOAs are introduced for modelling a different scenario, therefore the underlying assumptions of the two formalisms are different. For example, in MSCA we do not need to require syntactic consistency (i.e. $T^{\Box} \subseteq T^{\Diamond}$), as it is the case for MIOA, and the same request can be both permitted and necessary in different states (e.g. the request *receipt* of `Client1` in Section 2) or can only be necessary (the request *locker* of `Client3`). The composition operators of MIOA are restricted to composable MIOAs, and do not admit contracts that compete in offering or requiring the same action. We do not have this restriction in MSCAs, and for example in $\mathcal{K}_{(\texttt{Client1} \otimes \texttt{Hotel Service}) \otimes \texttt{Client2}}$ in Figure 1e both `Client1` and `Client2` perform the same offer $\overline{card}$. Moreover, MIOAs are input-enabled (in every state all possible requests must be performed), feature non-linear behaviours (i.e. broadcasting offers to every possible request), and have a notion of compatibility opposite to our agreement (i.e. all offers are matched).

With a coincidentally similar name, contract automata have been introduced in [2] for modelling generic natural language legal contracts between two parties. They are finite state automata in which states are tagged with deontic modalities that are obligations and permissions. A contract is satisfied if all deontic modalities are honoured and it is violated otherwise. Compared to our proposal, MSCAs have been introduced to study a different domain (i.e. service contracts) than the one in [2] (i.e. natural language contracts). We define obligations and permissions on the actions of contracts while in [2] modalities are defined on the states of automata, and only biparty contracts are considered. Our formalism is compositional and an MSCA can represent either a single principal or a multi-party composition of contracts. Finally, we focus on the problem of synthesising an orchestration of services in agreement, while in [2] techniques for solving violation of contracts are mainly studied.

7. CONCLUSION AND FUTURE WORK

In this paper we tackled the problem of specifying variability in service contracts. We have introduced modal service contract automata as a novel compositional formalism for modelling adaptive service contracts. Through MSCA each service provides a behavioural description of both its necessary and permitted requirements. An agreement among the parties is possible if all their necessary requirements are satisfied by corresponding offers.

Our approach is based on an orchestration of services that adapts the contracts to satisfy the maximum number of permitted requirements while guaranteeing agreement. A technique for synthesising an orchestration of service contracts is described, in particular necessary requirements are interpreted as uncontrollable actions in the style of supervisory control for discrete event systems [11], and the orchestrator is defined as the most permissive controller. We have fully mechanised our proposal in a prototypical tool available at https://github.com/davidebasile/CAT.

We describe some possible future developments of the proposed approach. Concerning the synthesis of the orchestrator, note that our orchestration generates a “greedy” coordination policy: the mpc can prevent a principal from firing a (matched) necessary request as long as the first match is available. Indeed, in our formalism necessary match actions are uncontrollable. It would be interesting to study two different policies of orchestration.

In the first one we could allow necessary match transitions to be controllable, to avoid the aforementioned case. We conjecture that this form of orchestration guarantees larger behaviours in agreement than the one presented in this paper, because intuitively we are increasing the controllable actions.

On the converse, a second stricter orchestration policy could not prevent a service from firing one of its requests even if the corresponding offer is not yet available (i.e. necessary matched requests are uncontrollable). This could be used to model critical service requests to be satisfied in each possible state. This scenario is reasonable in case of critical applications, where an example of a critical request could be to seal the doors of a chemical laboratory where infective pathogens have been accidentally released. We believe that the two discussed policies of orchestration can be synthesised with minor changes to Theorem 1.

Other more general future improvements follow. An interesting line of research concerns the identification of the conditions for removing the orchestrator to obtain a fully distributed choreography of services in agreement. These conditions have been successfuly identified in contract automata [4], but it is not clear whether they hold for the MSCA formalism. It is also worthwhile to analyse circular issues in modal contracts, similarly to [7] (see Section 6). Studying security-related variability in a SOC framework [6], where sessions among services are triggered by MSCAs in agreement, is an interesting future implementation of our proposal.

Along the line of [9], we are planning to enrich MSCAs with logic formulae for defining feature diagrams-related variability constraints that cannot be expressed only through MSCA, as for example exclusive or (at most one among a set of permitted requirements must be satisfied).

Another important future improvement is the enhancement of request and offer actions with quantities. For example, `Client1` in Section 2 could express the actual amount of money that it is willing to pay. Reaching an agreement would amount to find the optimal trade-off among principals, such that each one has a positive pay-off function. More generally, we are planning to formalise Quality of Service parameters in Service Level Agreement through our formal model, to assess non-functional parameters as reliability or energy consumption in a composition of service contracts. We would like to apply our formalism to model and analyse real world service-based applications, as it has been done in [5].

Finally we are planning to improve the scalability of our prototypical tool and to add a user-friendly GUI.

8. REFERENCES

[1] Asirelli, P., ter Beek, M.H., Fantechi, A., Gnesi, S.: A Logical Framework to Deal with Variability, pp. 43–58. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)

[2] Azzopardi, S., Pace, G.J., Schapachnik, F., Schneider, G.: Contract automata. Artificial Intelligence and Law 24(3), 203–243 (2016)

[3] Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in behavioural contracts: A brief survey. In: Essays Dedicated to Pierpaolo Degano on Programming Languages with Applications to Biology and Security - Volume 9465. pp. 103–121. Springer-Verlag (2015)

[4] Basile, D., Degano, P., Ferrari, G., Tuosto, E.: Relating two automata-based models of orchestration and choreography. JLAMP 85(3), 425–446 (2016), http://www.sciencedirect.com/science/article/pii/S2352220815000930

[5] Basile, D., Chiaradonna, S., Giandomenico, F.D., Gnesi, S.: A stochastic model-based approach to analyse reliable energy-saving rail road switch heating systems. Journal of Rail Transport Planning & Management 6(2), 163 – 181 (2016), http://www.sciencedirect.com/science/article/pii/S2210970616300051

[6] Basile, D., Degano, P., Ferrari, G.L.: A formal framework for secure and complying services. The Journal of Supercomputing 69(1), 43–52 (2014)

[7] Basile, D., Degano, P., Ferrari, G.L.: Automata for Specifying and Orchestrating Service Contracts. Logical Methods in Computer Science Volume 12, Issue 4 (Dec 2016), http://lmcs.episciences.org/2618

[8] Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Playing with our cat and communication-centric applications. In: Albert, E., Lanese, I. (eds.) FORTE 2016. pp. 62–73. Springer International Publishing

[9] 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. JLAMP 85(2), 287 – 315 (2016), http://www.sciencedirect.com/science/article/pii/S2352220815001431

[10] ter Beek, M.H., Reniers, M.A., de Vink, E.P.: Supervisory controller synthesis for product lines using CIF 3. In: Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Proceedings, Part I. pp. 856–873 (2016)

[11] Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

[12] Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories, pp. 64–79. Springer Berlin Heidelberg (2007)

[13] Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer 40(11), 38–45 (Nov 2007)