Behavioural, Functional, and Non-functional Contracts for Dynamic Selection of Services

Carlos G. Lopez Pombo[1,2], Hernán Melgratti[3,4], Agustín E. Martinez-Suñé[5], Diego Senarruzza Anabia[4], and Emilio Tuosto[6(✉)]

[1] Centro Interdisciplinario de Telecomunicaciones, Electrónica, Computación y Ciencia Aplicada, Universidad Nacional de Río Negro - Sede Andina, San Carlos de Bariloche, Argentina
cglopezpombo@unrn.edu.ar
[2] Consejo Nacional de Investigaciones Científicas y Técnicas - CONICET, Buenos Aires, Argentina
[3] Instituto de Ciencias de la Computación CONICET—UBA, Buenos Aires, Argentina
hmelgra@dc.uba.ar
[4] Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina
[5] Department of Computer Science, University of Oxford, Oxford, UK
agustin.martinez.sune@cs.ox.ac.uk
[6] Gran Sasso Science Institute, L'Aquila, Italy
emilio.tuosto@gssi.it

Abstract. We propose a mechanism for selecting distributed services which encompasses three orthogonal, yet related type of contracts' compliance. Indeed, we envisage *contract compliance* as the intersection of *behavioural contract* compliance with the compliance of *functional and non-functional* contracts. We model services as *communicating-finite state machines* (CFSMs) suitably extended to capture data-awareness and application-level quality-of-service (QoS). This extension is instrumental to define our notion of contract compliance in terms of a bisimulation relation for this new class of CFSMs. More precisely, we introduce CFSMs where transitions are decorated with constraints on the payloads while states of CFSMs have decorations that carry QoS contracts. This allows us to capture behavioural contracts (considering the communication pattern as usual in systems of CFSMs) as well as functional and non-functional contracts. We use a case study to assess our approach and we discuss tool support for our framework.

Research partly supported by the PRIN PNRR project DeLICE (F53D23009130001), by the MUR dipartimento di eccellenza 2023–2027.
The authors thank the anonymous reviewers for their comments.
C. G. Lopez Pombo—On leave from Instituto de Ciencias de la Computación CONICET–UBA, Buenos Aires, Argentina, and Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Buenos Aires, Argentina.

Published by springer Nature Switzerland AG 2025
C. Di Giusto and A. Ravara (Eds.): COORDINATION 2025, LNCS 15731, pp. 153–174, 2025.
https://doi.org/10.1007/978-3-031-95589-1_8

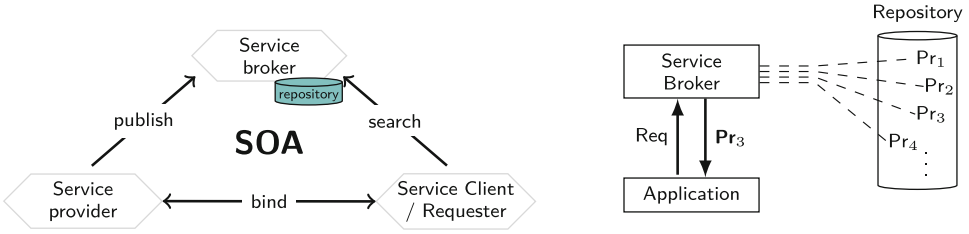


Fig. 1. SOA (Left) and the service selection problem (Right)

Keywords: Service compliance · Service discovery · Bisimulation · Communicating systems · Communicating Finite-State Machines · Behavioural contracts · Quality of Service (QoS) · Service Level Agreement (SLA) · POP protocol

1 Introduction

A paramount requirement of distributed applications is the adaptation to variations in the environment in which they operate. In the past two decades *service-oriented computing* (SOC) has become a predominant paradigm which has generated several variants such as —among others— cloud computing, fog and edge computing, and many forms of distributed computing associated with what is known as the *Internet of Things*.[1] A common characteristic of these *service-oriented architectures* (SOAs) is a globally available computational infrastructure for service composition. In fact, SOC advocates mechanisms for dynamically and automatically search and combine distributed computational resources exposed as services interacting over an existing communication infrastructure. This vision of software systems is partially present in applied technologies, such as RESTful APIs (that foster the API Economy by supporting dynamic reconfiguration); however, to the best of our knowledge, full automation of service composition is not supported.

Figure 1 (left) depicts a general view of the main elements present in SOA. Service Providers make available the interface of their services by publishing them to a Service broker which maintain a repository of services. At run-time, a Service client/Requester may issue a request to the broker for searching a given service to accomplish its goals. Then, the Service broker is in charge of choosing an appropriate service from its repository. This mechanism relies on the ability of the Service broker to solve a *service selection problem* schematically shown in Fig. 1 (Right): the Service broker has to choose a service from a Service repository matching the Service client/Requester needs. Ideally, this selection process should be automatic and take into account constraints on the expected behaviour and on the parameters involved in communications, as well as on *service level agreements*, or SLAs for short [17].

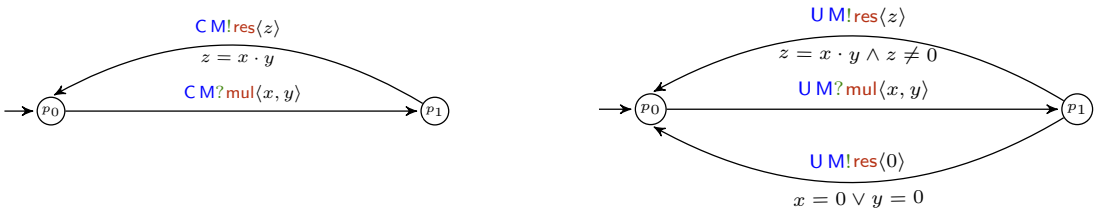
In [27] we proposed SEArch, after *s*ervice *e*xecution *arch*itecture, an implementation of a SOA for the language-independent execution of service-oriented software applications, capable of performing transparent dynamic reconfiguration. Key to this, lays the ability of SEArch of performing automatic service

[1] Although the terminology has evolved, SOC is still widely used.

discovery. To this aim, the infrastructure relies on a choreographic view [11] which proposes an interaction mechanism by conceptually separating the local computations of the components from their communication aspects. More precisely, both provision and requirement contracts are represented as *communicating finite-state machines* [9] (CFSMs) in order to (*i*) foster heterogeneity by abstracting away implementation details and (*ii*) support automatic discovery of services through a mechanism based on bisimilarity introduced in [36] to establish the compliance between provision and requirement contracts.

In this paper we combine the approach in [31] with the service discovery mechanism advocated in [27] and the approach in [24,28]. Noticeably, [31] borrows asserted CFSMs from [18] to formalise a notion of behavioural and functional contracts onto which make service discovery. In fact, the main contribution of [31] is the implementation of the binding mechanism defined in [36] to support service *interoperability* understood as *behavioural* compliance (given by the bisimulation of CFSMs).

Our view of the problem at stake can be understood as the intersection of *behavioural* compliance extended with *functional* compliance (featured by asserted CFSMs), and *non-functional* compliance (attained through labelling states with a formal characterisation of the admissible values of *application-level* quality-of-service (QoS) attributes). As an example, two contracts for a multiplication service can be given by the following decorated CFSMs:



where transitions are labelled with communication actions (e.g., $\mathsf{U\,M!res}\langle z\rangle$ and $\mathsf{U\,M?mul}\langle x, y\rangle$ respectively represents an output of the result from the service to the client and an input of the service from the client). The functional contract is given by the assertions on the payload variables decorating the transitions (e.g., $z = x \cdot y$). Finally, the non-functional contracts (not represented in the machine for readability) are given by associating to each state a QoS specifications, that is a constrain on the QoS attributes in the state; for instance, the state p_1 could be assigned the QoS specification $\mathsf{bytes} \leq 100$ to indicate the total bytes of the payload should be less than 100.

A natural question to ask is whether the services above are equivalent. Intuitively, they are because even if the one on the right responds to clients using different computations for the result (e.g., avoiding to use a multiplication algorithm when one of the inputs is 0). However, this is not easy to attain without accounting for the functional contract in the definition of bisimulation. Moreover, transitions cannot easily be matched in the bisimulation game when, as in our case, they use different names. In fact, we intentionally used C (after client) and U (after user) to refer to clients of the two services, which a plausible choice that a developer can make when registering the contract of their service.

As we will see, our approach tackles this issues by providing a notion of bisimulation to use for service discovery that consider (non-)functional contracts as well as differences on participants, messages, and variables names.

Structure and Contributions. After surveying background material in Sect. 2, we introduce in Sect. 3 a new class of CFSMs, dubbed *extended* CFSM (cf. Definition 6) obtained by blending the variants of CFSMs introduced in [18] with those in [24,28]. More precisely, we borrow *asserted CFSMs* from [18] to specify functional contracts and *QoS-extended* CFSMs from [24,28] for non-functional contracts. We provide a notion of bisimulation (cf. Definition 10) on extended CFSMs together with an algorithm for computing such bisimulation Sect. 3.3. A feature of the algorithm is that it tries to build bisimulation relations *modulo name matching*, namely without assuming that machines necessarily share a naming convention. This feature is inspired by the approach used in the toolkit SEArch [27] which we plan to extend in order to support e-CFSMs. In Sect. 4 we elaborate on the model of the POP protocol used in [24] to highlight the main feature of our approach. We discuss related work and draw conclusions in Sect. 5 which also sketch future research directions.

2 Background

We briefly survey communicating systems [9], first-order logic with equality, and *real-closed field* [33].

2.1 Communicating Systems

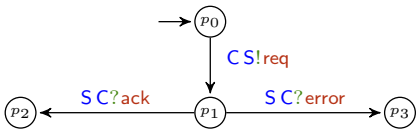
Hereafter, we fix a set $\mathcal{P}$ of *participants* and let $\mathsf{A}, \mathsf{B}, \ldots$ range over $\mathcal{P}$. Participants interact through *channels*; more precisely, a channel $\mathsf{A}\,\mathsf{B}$ allows A to asynchronously send messages to B through an unbounded FIFO buffer associated to the channel. Let $\mathcal{C} = \{\mathsf{A}\,\mathsf{B} \mid \mathsf{A} \neq \mathsf{B} \text{ are participants }\}$ be the set of channels. Fixed a set $\mathcal{M}$ of *messages* such that $\mathcal{P} \cap \mathcal{M} = \emptyset$, we define the set $\mathcal{L}$ of *communication actions* as $\mathcal{L} = \mathcal{L}^! \cup \mathcal{L}^?$ where $\mathcal{L}^! = \{\mathsf{A}\,\mathsf{B}!\mathsf{m} \mid \mathsf{A}\,\mathsf{B} \in \mathcal{C} \text{ and } \mathsf{m} \in \mathcal{M}\}$ and $\mathcal{L}^? = \{\mathsf{A}\,\mathsf{B}?\mathsf{m} \mid \mathsf{A}\,\mathsf{B} \in \mathcal{C} \text{ and } \mathsf{m} \in \mathcal{M}\}$ are respectively the set of *output* and *input* actions. The *subject* of $\mathsf{A}\,\mathsf{B}!\mathsf{m} \in \mathcal{L}$ (resp. $\mathsf{A}\,\mathsf{B}?\mathsf{m} \in \mathcal{L}$), is $\mathsf{sbj}(\mathsf{A}\,\mathsf{B}!\mathsf{m}) = \mathsf{A}$ (resp. $\mathsf{sbj}(\mathsf{A}\,\mathsf{B}?\mathsf{m}) = \mathsf{B}$).

Definition 1 (Communicating systems [9]). *A* communicating finite-state machine *(*CFSM*) is a finite transition system $M = (Q, q_0, \rightarrow)$ where:*

- *Q is a finite set of* states *with $q_0 \in Q$ the* initial *state, and*
- *$\rightarrow \subseteq Q \times \mathcal{L} \times Q$; we write $q \xrightarrow{l} q'$ for $(q, l, q') \in \rightarrow$.*

Given $\mathsf{A} \in \mathcal{P}$, M is A -local *if $\mathsf{sbj}(l) = \mathsf{A}$ for each $q \xrightarrow{l} q'$. A* (communicating) system *is a map $S = (M_\mathsf{A})_{\mathsf{A} \in \mathcal{P}}$ assigning a A-local CFSM M_A to each $\mathsf{A} \in \mathcal{P}$.*

Example 1 (Communicating Finite State Machine). The CFSM below yields the behaviour of a client C interacting with a server S



There, C sends a request to S (message req) and waits either for and acknowledgement (message ack) or for the notification of an error (message error). ◇

2.2 First-Order Logic with Equality

We use first-order logic with equality ($FOL_=$ for short) to formally express functional and non-functional contracts (see [15,32] for an extensive presentation of $FOL_=$).

We fix a set $\mathtt{S}$ of *sorts* (whose elements represent types such as int, string, etc.) and let $\mathtt{S}^\star$ be the set of finite sequences on $\mathtt{S}$ (as usual, we write $s_1\ s_2\ \ldots s_n$ for the elements in $\mathtt{S}^\star$ and ϵ for the empty sequence).

Definition 2 (Signature). *A tuple* $\langle \mathtt{X}, \mathtt{F}, \mathtt{P}, \sigma\rangle$ *is an* $\mathtt{S}$-signature (on $\mathtt{X}$, $\mathtt{F}$, and $\mathtt{P}$) *if* $\mathtt{X}$, $\mathtt{F}$, $\mathtt{P}$ *are three denumerable sets of* variable, function, *and* predicate *symbols respectively, while* σ *maps elements in* $\mathtt{X}$, $\mathtt{F}$, *and* $\mathtt{P}$ *respectively to* $\mathtt{S}$, $\mathtt{S}^\star \times \mathtt{S}$, *and* $\mathtt{S}^\star$.

Hereafter, $\Sigma = \langle \mathtt{X}, \mathtt{F}, \mathtt{P}, \sigma\rangle$ is a fixed yet arbitrary $\mathtt{S}$-signature that determines the constructors for our terms and the predicates of our logic as formalised by Definitions 3 and 4 below. We extend σ on $\mathcal{T}(\Sigma)$ by defining $\sigma(c(t_1, \ldots, t_n)) = s$ if $\sigma(c) = (s_1, \ldots s_n, s)$ and $\sigma(t_i) = s_i$ for all $1 \leq i \leq n$.

Definition 3 (Terms). *The set* $\mathcal{T}(\Sigma)$ *of* terms over Σ *is the smallest set including* $\mathtt{X}$ *and such that* $c(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma)$ *for all* $c \in \mathtt{F}$ *with* $\sigma(c) = (s_1 \ldots s_n, s)$ *and* $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$ *with* $\sigma(t_i) = s_i$ *for all* $1 \leq i \leq n$.

A constant symbol *is a symbol* $c \in \mathtt{F}$ *such that* $\sigma(c) = (\epsilon, S)$.

Definition 4 ($FOL_=$ and theory presentation). *The set of* $FOL_=$ formulae *is the smallest set* $\mathcal{F}(\Sigma)$ *such that:*

- $t = t' \in \mathcal{F}(\Sigma)$ *for all* $t, t' \in \mathcal{T}(\Sigma)$ *with* $\sigma(t) = \sigma(t')$
- $p(t_1, \ldots, t_n) \in \mathcal{F}(\Sigma)$ *for all* $p \in \mathtt{P}$ *and* $t_1, \ldots, t_n \in \mathcal{T}(\Sigma)$ *with* $\sigma(p) = s_1 \ldots s_n$ *and* $\sigma(t_i) = s_i$ *for all* $1 \leq i \leq n$
- $\neg\phi$, $\phi \vee \phi'$, $(\exists x)\phi \in \mathcal{F}(\Sigma)$ *for all* $\phi, \phi' \in \mathcal{F}(\Sigma)$ *and* $x \in \mathtt{X}$.

A Σ-theory presentation *is a pair* $\langle \Sigma, \Gamma\rangle$ *where* $\Gamma \subseteq \mathcal{F}(\Sigma)$.

The semantics of $FOL_=$ is defined as usual in terms of Σ-*models* that is pairs $\langle \iota, \chi\rangle$ where ι interprets symbols as follows:

- $\iota(s)$ it a set for each sort symbol $s \in \mathtt{S}$
- for each function symbol f with $\sigma(f) = (s_1\ \ldots s_n, s)$, $\iota(f)$ is a function from $\iota(s_1) \times \cdots \times \iota(s_n)$ to $\iota(s)$, and
- if p is predicate symbols with $\sigma(p) = s_1\ \ldots s_n$ then $\iota(p) \subseteq \iota(s_1) \times \cdots \times \iota(s_n)$ is a relation over interpretation of $\sigma(p)$;

and χ is a *valuation* of the variable symbols given as a function $\chi : \mathtt{X} \to \bigcup_{s \in \mathtt{S}} \iota(s)$ such that for all $x \in \mathtt{X}$, $\chi(x)$ belongs to the Σ-interpretation of $\sigma(x)$. Terms are then interpreted as: $\langle \iota, \chi \rangle(x) = \chi(x)$ for all $x \in \mathtt{X}$ and $\langle \iota, \chi \rangle(f(t_1, \ldots, t_n)) = \iota(f)(\langle \iota, \chi \rangle(t_1), \ldots, \langle \iota, \chi \rangle(t_n))$. The satisfaction relation (denoted as $\models$) is then defined between Σ-models and formulae in $\mathcal{F}(\Sigma)$ as usual by inductively extending valuations of variables and interpretations of function symbols to terms, and interpretations of predicate symbols to complex formulae.

Standard logical connective and the universal quantifier are defined as usual in terms of $\neg$, $\wedge$, and $\exists$; in particular, we will shorten $\neg(\neg\phi \vee \neg\phi')$ with $\phi \wedge \phi'$, $(\neg\phi) \vee \phi'$ with $\phi \implies \phi'$, and $\neg(\exists x)(\neg\phi)$ with $(\forall x)\phi$. Also, we assume that $\neg$ takes precedence over $\wedge$.

2.3 Real-Close Fields

We will use $FOL_=$ to formalise the algebraic structures needed for the development of the contributions of this work. In particular, QoS constraints will be expressed as a $FOL_=$ language over *real-closed fields* (RCFs), a formalisation of real numbers with equality based on (*ordered*) *field* studied by Tarski [33, Section 5, Note 9].

We start by defining *fields* [6, Definition 3.3.5] as a *theory presentation*, that is a $FOL_=$ language over a signature whose functional symbols include $+$, $\cdot$, 0, and 1 to represent a sum, a product, together with the finite axioms

$$(\forall x)(\forall y)(x + y = y + x) \quad \text{and} \quad (\forall x)(\forall y)(x \cdot y = y \cdot x) \tag{1}$$

$$(\forall x)(\forall y)(\forall z)((x + y) + z = x + (y + z)) \tag{2}$$

$$\text{and} \quad (\forall x)(\forall y)(\forall z)((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

$$(\forall x)(\forall y)(\forall z)((x + y) \cdot z = x \cdot z + y \cdot z) \tag{3}$$

$$(\forall x)(\exists y)(x + y = 0) \quad \text{and} \quad (\forall x)(x = 0) \vee (\exists y)(x \cdot y = 1) \tag{4}$$

to state the commutativity and associativity and of the $+$ and $\cdot$, the distributivity of the product over the sum, and the existence of inverses.

Let Σ_{RCF} be a signature for a field including a predicate symbol $<$ to represent an order relation. Next we show the axiomatic systems of the *real-closed fields* [33, Section 5, Note 9] as a first-order logic theory presentation, denoted by $\langle \Sigma_{\mathrm{RCF}}, \Gamma_{\mathrm{RCF}} \rangle$, as *ordered* fields where Γ_{RCF} contains the axioms (1–4) above together with

- the axioms to make $<$ a strict total order,
- the axiom establishing the existence of square roots of positive elements, and
- the axioms stating that every polynomial of odd degree has at least a zero.

Let us present these axioms formally as $FOL_=$ formulae. Recall that a binary relation $\leq$ is a *total order* if it is reflexive, transitive, antisymmetric, and total:

- $(\forall x)(x \leq x)$ reflexivity
- $(\forall x)(\forall y)(\forall z)(x \leq y \wedge y \leq z \implies x \leq z)$ transitivity
- $(\forall x)(\forall y)(x \leq y \wedge y \leq x \implies x = y)$ antisymmetry
- $(\forall x)(\forall y)(x \leq y \vee y \leq x)$ totality

The strict total order $<$ is then defined in terms of $\leq$ as:

- $(\forall x)(\forall y)(x < y \iff x \leq y \wedge \neg(x = y))$ strictness

while the remaining axioms for real-closed fields are

- $(\forall x)(x > 0 \implies (\exists y)(y \cdot y = x))$ square root
- $(\forall a_0)(\forall a_1)\ldots(\forall a_{2n+1})\left(a_{2n+1} \neq 0 \implies (\exists x)\left(\sum_{i=0}^{2n+1} a_i x^i = 0\right)\right)$ polynomials

respectively stating that every positive element has a square root and that every polynomial of odd degree has a root (note that the latter is an axioms' schema). The acclimatisation shown above was proved to be consistent and complete [34, Section 5, Note 15], its soundness follows from observing that all the axioms are true under the usual interpretation of first-order languages. Finally, [34, Theorem 37] proves that the axiomatic formalisation of the RCFs is decidable. Later, it was shown that all real-closed fields are isomorphic [16, Theorem 2.1] to the real numbers $\mathbb{R}$ which indeed form a real-closed field if equipped with the usual addition and multiplication operations, and the usual order relation.

3 Service Compliance

We aim to develop mechanisms to dynamically select services based on: (i) behavioural compliance (defined in terms of communication patterns of services), (ii) functional compliance (expressed as symbolic constraints on payloads), and (iii) non-functional compliance (understood as application-level QoS).

3.1 Extended CFSMs

We now introduce a new class of CFSMs that blends together the data-awareness proposed in [18] and QoS-extended CFSMs proposed in [24,28]. The former enriches the labels of transitions by incorporating symbolic constraints on the payloads of the messages, while the latter assigns extensions of the theory presentations of RCF (cf. Sect. 2.2) to states (see Definition 6).

Definition 5 (QoS specification). *A* QoS specification $\langle \Sigma_{\text{QoS}}, \Gamma \rangle$ *is a theory presentation extending* $\langle \Sigma_{\text{RCF}}, \Gamma_{\text{RCF}} \rangle$ *as follows:*

1. Σ_{QoS} *extends the set of symbols for function of* Σ_{RCF} *with a distinguished finite set* Q *of constant symbols (other that* 0 *and* 1*) representing the* quantitative attributes *(from now we refer to the elements of* Q *as* QoS attributes*),*
2. *and* $\Gamma = \Gamma_{\text{RCF}} \cup \Gamma'$*, where* Γ' *is a finite set of closed* $FOL_{=}$ *formulae formalising specific constraints over the QoS attributes in* Q*.*

The class of QoS specifications will be denoted as $\mathcal{C}(\mathsf{Q})$.

Notice that, to preserve decidability of QoS properties, QoS specifications can only admit extensions made by adding constant symbols. These symbols are instrumental to the representation of the QoS attributes of components.

Example 2 (QoS specification). With reference to Example 1, possible quantitative attributes of interest in an implementation might be $\mathsf{Q} = \{\mathsf{t}, \mathsf{m}\}$ representing CPU time and memory usage, respectively. Let $\Gamma' = \{1 \leq \mathsf{t} \wedge \mathsf{t} \leq 5, \mathsf{m} \leq 10\}$ then, according to Definition 5, $\langle \Sigma_{\text{QoS}}, \Gamma_{\text{RCF}} \cup \Gamma' \rangle$ is a QoS specification that characterises the computational cost in terms of time and memory consumption of a process that might take place in state p_0. ◇

Functional contracts can be formalised in terms of $FOL_{=}$ formulae predicating on the payloads of messages as shown in the next example.

Example 3 (Functional contracts). Assume that transition $p_0 \xrightarrow{\mathsf{C\,S!\,req}} p_1$ in Example 1 has a string r and an integer b as payloads. The formula ϕ defined as $0 < len(r) < b$ then requires C to provide a non-empty string whose length is strictly smaller than b.[2] ◇

In [18], *asserted CFSMs* (*a-CFSMs*) are introduced to specify contracts, such as the one in Example 3. Fix an arbitrary $FOL_{=}\mathtt{S}$-signature Σ for functional contracts; we assume that the typing function σ of Σ maps messages in $\mathcal{M}$ to $\mathtt{S}^\star$; intuitively, for $\mathsf{m} \in \mathcal{M}$, $\sigma(\mathsf{m})$ yields the sorts of the payload of m (for instance, $\sigma(\mathsf{req}) = \mathsf{string\ int}$ in Example 3). For non-functional contracts, Definition 6 below extends a-CFSMs with QoS specifications on theory presentations $\langle \Sigma_{\text{QoS}}, \Gamma_{\text{RCF}} \cup \Gamma' \rangle$, where Σ_{QoS} is defined as in Definition 5 with the set Q of QoS attributes axiomatised by a finite set of $FOL_{=}$ formulae Γ'. We assume that the set of symbols for variables in Σ and those in Σ_{QoS} are disjoint and fix the set $\mathcal{C}$ of QoS specifications on Σ_{QoS}.

Definition 6 (Extended CFSMs). *An* extended CFSM *(e-CFSM for short) is a tuple* $\langle M, F, \mathit{qos}, \mathit{asrt} \rangle$ *where:*

- $M = \langle Q, q_0, \rightarrow \rangle$ *is a CFSM with* $F \subseteq Q$ *the set of* final states,
- $\mathit{qos} : Q \to \mathcal{C}$ *maps states of* M *to QoS specifications, and*
- asrt *maps transitions of* M *to first-order formulae in* $\mathcal{F}(\Sigma)$.

An extended communicating system *is a map* $(M_{\mathsf{A}})_{\mathsf{A} \in \mathcal{P}}$ *assigning an* A*-local e-CFSM* M_{A} *to each* $\mathsf{A} \in \mathcal{P}$.

In order to lighten the notation, we omit representing assertions that are *True*. For a state q of an e-CFSM M, we let $\mathit{paths}(q)$ be the set of simple paths[3] that start at the initial state of M and end at q. The set of variables

[2] Formally, ϕ should be defined as $0 < len(r) \wedge len(r) < b$. For the sake of clarity, we adopt conventional abbreviations in the arithmetic and logical expressions used in the examples.

[3] A path is simple when it has no cycle.

of a transition t, denoted as $var(t)$, is the set of variables $var(l)$ occurring in the label l of t; given a path π, $vars(\pi)$ is the union of the variables from each transition on π. We say that a transition t *assigns* variable v if $v \in var(t)$.

Functional contracts introduce an undesired form of non-determinism on e-CFSMs as illustrated by the following machines:[4]

The only non-deterministic behaviour in the machine on the left is the internal choice made by the sender C when assigning variable n. The machine on the right, in addition the internal choice of C, exhibits further non-deterministic behaviour when C decides to assign 0 to n, as the computation can continue from either q_2 or q_3. This form of uncontrolled non-determinism is undesired because it is not controlled by any of the participants in the system. Hereafter, we restrict to *assertion-deterministic* machines according to the next definition.

Definition 7 (Assertion determinism). *An e-CFSM M is* assertion-deterministic *if for every pair of transitions $q \xrightarrow[\phi_1]{l} q'$ and $q \xrightarrow[\phi_2]{l} q''$ in M there is no Σ-model $\mathcal{M}$ such that $\mathcal{M} \models \phi_1$ and $\mathcal{M} \models \phi_2$.*

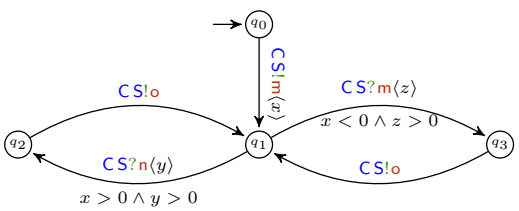
Note that the definition above does not rule out non-determinism due to internal choice of senders discussed above.

The next concept is of utmost importance, as it determines whether the assertion labelling a transition can be evaluated, depending on whether the variables involved in the assertion have been assigned a value or not.

Definition 8 (History sensitivity). *Let M be an e-CFSM. A transition $t = q \xrightarrow[\phi]{l} q'$ of M* knows *a variable v if: t assigns v, or $v \in vars(\pi)$ for all $\pi \in paths(q)$. Machine M is* history sensitive *if each transition t of M knows all the free variables in the assertion of t.*

3.2 Bisimulation Relations for Extended-CFSMs

In this section we give a definition of bisimulation inspired by, but also extending, the one given in [31, Definition 19] for a-CFSMs. We now address some nuances involved in developing a bisimulation notion for e-CFSM. Consider the e-CFSM and the paths below:



$\pi_0 = q_0 \xrightarrow{C\,S!m\langle x\rangle} q_1$

$\pi_1 = \pi_0 \xrightarrow[x>0\wedge y>0]{C\,S?n\langle y\rangle} q_2$

$\pi_2 = \pi_1 \xrightarrow{C\,S!o} q_1$

$\pi_3 = \pi_2 \xrightarrow[x>0\wedge y>0]{C\,S?n\langle y\rangle} q_2$

[4] In the following, we will omit assertions when they are *True* for readability.

We have that the assertion $x > 0 \wedge y > 0$ holds along $\pi_2 \in \mathit{paths}(q_1)$ while the assertion $x < 0 \wedge z > 0$ holds on the path $\pi_0 \xrightarrow[x<0\wedge z>0]{\mathsf{C\,S?m}\langle z\rangle} q_3 \xrightarrow{\mathsf{C\,S!o}} q_1 \in \mathit{paths}(q_1)$. Therefore, we should consider $(q_1, x > 0 \wedge y > 0)$ and $(q_1, x < 0 \wedge z > 0)$ as different states, since the *knowledge* of C differs depending on the path followed to reach q_1. This shows that the bisimulation relation must be formalised, not only in terms of the states of the machines, but also considering the logical information collected along paths. This definition of knowledge, and its association to discrete states, resembles the use of assignment's information in order to characterise states in data-flow analysis [1, Chapter 9]. Below, we formally define the notion of *knowledge*, which intuitively corresponds to the conjunction of the assertions ϕ on a path π, where the variables are not re-assigned after the occurrence of ϕ on π.

Definition 9 (Knowledge). *Given a communication action $l \in \mathcal{L}$, recall that $\mathit{var}(l)$ is the set of variables in the payload of l. The* residual *of an assertion ϕ after l, written $\phi \mathbin{\overline{\wedge}} l$, is defined as*

$$p(x_1,\ldots,x_n) \mathbin{\overline{\wedge}} l = \begin{cases} p(x_1,\ldots,x_n) & \mathit{var}(l) \cap \{x_1,\ldots,x_n\} = \emptyset \\ \bot & \text{otherwise} \end{cases}$$

$$(\neg\phi) \mathbin{\overline{\wedge}} l = \begin{cases} \bot & \phi \mathbin{\overline{\wedge}} l = \bot \\ \neg(\phi \mathbin{\overline{\wedge}} l) & \text{otherwise} \end{cases}$$

$$(\phi_1 \vee \phi_2) \mathbin{\overline{\wedge}} l = \begin{cases} (\phi_1 \mathbin{\overline{\wedge}} l) \vee (\phi_2 \mathbin{\overline{\wedge}} l) & \phi_1 \mathbin{\overline{\wedge}} l \neq \bot \text{ and } \phi_2 \mathbin{\overline{\wedge}} l \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

$$((\exists x)\phi) \mathbin{\overline{\wedge}} l = \begin{cases} (\exists x)(\phi \mathbin{\overline{\wedge}} l) & x \notin \mathit{var}(l) \text{ and } \phi \mathbin{\overline{\wedge}} l \neq \bot \\ ((\exists y)(\phi[y/x])) \mathbin{\overline{\wedge}} l & x \in \mathit{var}(l),\ y \text{ fresh, and } \phi \mathbin{\overline{\wedge}} l \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

The *knowledge* $\mathcal{K}(\pi)$ *on* π is the inductively defined on π as $K(\pi, \{\mathit{True}\})$ where

$$K(\pi, X) = \begin{cases} \bigwedge_{\psi \in X} \psi, & \pi \text{ is the empty path} \\ K(\pi', \{\psi \mid \psi \in X \text{ and } \psi \mathbin{\overline{\wedge}} l \neq \bot\} \cup \{\phi\}), & \pi = q \xrightarrow[\phi]{l} \pi' \end{cases}$$

Intuitively, the knowledge on a path is the conjunction of all the assertions ϕ that are not annihilated by actions that assign variables over which ϕ predicates. Notice that the residual of a disjunction is $\bot$ when the residual of either of the disjuncts is annihilated because if the knowledge of one sub-formula becomes uncertain then nothing can be said about the disjunction.

Example 4 (Knowledge). If we consider the paths π_0, π_1, π_2 and π_3 detailed above we have:

$$\mathcal{K}(\pi_0) = (\mathcal{K}(\epsilon) \mathbin{\overline{\wedge}} \mathsf{C\,S!m}\langle x\rangle) = \mathit{True}$$
$$\mathcal{K}(\pi_1) = \mathit{True} \wedge x > 0 \wedge y > 0$$
$$\mathcal{K}(\pi_2) = \mathcal{K}(\pi_3) = \mathit{True} \wedge x > 0 \wedge y > 0 \wedge \mathit{True}$$

Notice that in the case of path π_3, the variable y is reassigned by the last transition of π_3; as a consequence, the previous assertion mentioning y is removed from the knowledge. Also, notice that on $\pi_2 \xrightarrow[x<0 \wedge z>0]{\mathsf{CS?m}\langle z\rangle} q_3$, q_3 is not accessible because the assertion associated with the last transition can only be taken when $x < 0$ holds; however $\mathcal{K}(\pi_1)$ implies that $x > 0$ holds. ◇

A consideration that must be taken into account is that the usual simulation condition cannot be naïvely applied since the knowledge at a given state q depends on the path used to reach q. We therefore give a definition of bisimulation which accounts for the assertion labelling transitions and the knowledge available at each state. More precisely, we consider a labelled transition system where the states are all the pairs (p, K) with p a state of the e-CFSM and K the knowledge of a path in $paths(p)$.

Next definition uses the auxiliary operation $\bar{\cdot}$ that given a conjunction ϕ of formulae in $\mathcal{F}(\Sigma)$ yields the formula $\overline{\phi}$ obtained by removing redundant (i.e., syntactically equal) conjuncts from ϕ. The reason to remove syntactically equal formulae is that cycles would induce infinitely many states that differ only because their associated knowledge repeats a same conjunct a different number of times; to avoid this it is enough to keep syntactically equal formulae just once.

Definition 10 (Bisimulating e-CFSMs). *Let M_1 and M_2 be two assertion-deterministic and history sensitive e-CFSMs on states Q_1 and Q_2 respectively with $q_{0,1} \in Q_1$ and $q_{0,2} \in Q_2$ be their initial states. A relation $\mathcal{R} \subseteq (Q_1 \times \mathcal{F}(\Sigma)) \times (Q_2 \times \mathcal{F}(\Sigma))$ is a* simulation *if, for all $l \in \mathcal{L}$, whenever $((p, K), (q, K')) \in \mathcal{R}$ and $p \xrightarrow[\phi]{l} p'$ in M_1, there is a non-empty set of transitions $T = \{q \xrightarrow[\psi_1]{l} q_1, \ldots, q \xrightarrow[\psi_k]{l} q_k\}$ in M_2 such that:*

1. *the formula $\neg\big(((K \,\overline{\wedge}\, l) \wedge \phi) \implies \bigvee_{q \xrightarrow[\psi]{l} q' \in T} ((K' \,\overline{\wedge}\, l) \wedge \psi)\big)$ is not satisfiable*
2. *$\big\{\big((p', \overline{(K \,\overline{\wedge}\, l) \wedge \phi \wedge \psi}), (q', \overline{(K' \,\overline{\wedge}\, l) \wedge \psi})\big) \mid q \xrightarrow[\psi]{l} q' \in T\big\} \subseteq \mathcal{R}$*
3. *if $p \in F_1$, then $q \in F_2$*
4. *if $qos(p) = \langle \Sigma, \Gamma_1 \rangle$ and $qos(q) = \langle \Sigma, \Gamma_2 \rangle$ then $\neg\big(\bigwedge_{\phi \in \Gamma_1} \phi \implies \bigwedge_{\phi \in \Gamma_2} \phi\big)$ is not satisfiable.*

Machines M_1 and M_2 are bisimilar *if there is a simulation $\mathcal{R}$ such that $\mathcal{R}^{-1}$ is a simulation and $((q_{0,1}, True), (q_{0,2}, True)) \in \mathcal{R}$.*

As hinted by the example in Sect. 1, a transition with label l and assertion ϕ can be simulated by several transitions of the other machine. This is captured in Definition 10 by the existence of a set of transitions T in the simulating machine that satisfy condition 1 and condition 2. Intuitively, the former checks that the transition $p \xrightarrow[\phi]{l} p'$ can be matched by a set of transitions from q such that the transitions have the same communication action l and whose assertions

"cover" the assertion ϕ. This boils down to say that the residual of K' encompasses the (residual) assertion at p.[5] Condition 2 requires that the target states of the transitions keep coinductively simulating each other. Crucially, for the target state p' to be simulated by q' along transition $q \xrightarrow[\psi]{l} q' \in T$, we add to the knowledge at p' the assertion ψ so that the simulation game continues recording the assumptions on the variables enforced by the transition $q \xrightarrow[\psi]{l} q'$. Condition 3 requires, as usual, that final states in one machine can be simulated only by final states in the other. Finally, Condition 4 requires that the QoS specification associated to q encompasses the one associated with p. Note that assertions of payloads do not affect non-functional contracts due to the fact that the variables of functional and non-functional contracts are disjoint.

3.3 Computing Bisimulation Relations

The implementation aspects and the extension of SEArch with e-CFSM are not in the scope of this work; nevertheless, we discuss the effectiveness of the computation of the bisimulation relation in Definition 10 also in relation to some practical aspects considered in the design of SEArch. A practical assumption done in SEArch is that there is a mismatch on the names used in the contracts on the provision's side and those on the requester's side. In fact, often these sides are developed by different parties without a common vocabulary.

Thus, we have to establish bisimilarity *modulo name matching*, namely we allow bisimilar e-CFSMs to have different signatures, set of participants, and set of messages. More precisely, the process of bisimulation checking between two e-CFSMs involves the search for a suitable bijection on the names used in their transitions. This can be done by adapting standard iterative refinement; in particular, we generalise the algorithm in [20].

For simplicity, we assume that the two signatures differ only on their set of variable symbols, while function symbols and predicate symbols coincide; in this way, we have to construct a name match only for the variable symbols of the signatures, the sets of participant names and messages used in communications (extending the algorithm to cover the construction of name matchings for function symbols and predicate symbols is straightforward).

For $i \in \{1, 2\}$, let $\Sigma_i = \langle \mathtt{X}_i, \mathtt{F}, \mathtt{P}, \sigma_i \rangle$ be two $\mathtt{S}$-signatures and

$$M_i = \langle (Q_i, q_{0_i}, \rightarrow_i), F_i, \mathsf{qos}_i, \mathsf{asrt}_i \rangle$$

be two e-CFSMs defined over a set of participants $\mathcal{P}_i$ and a set of messages $\mathcal{M}_i$ on Σ_i.

The computation of a bisimulation relation between two e-CFSMs (if any) is described by Algorithms 1 to 3 which search the space of bijections on participants, on messages, and between sets of variable symbols. For this task we

[5] Notice that, following the way SMT solvers check for validity, condition 1 requires the negated implication not to be satisfiable, which is equivalent to impose the validity of the implication.

Algorithm 1: Enumeration of matchers for computing a bisimulation

Function `e-CFSMBisimulation`:
Input : $M_1 = \langle (Q_1, q_{0_1}, \rightarrow_1), F_1, \mathsf{qos}_1, \mathsf{asrt}_1 \rangle$
Input : $M_2 = \langle (Q_2, q_{0_2}, \rightarrow_2), F_2, \mathsf{qos}_2, \mathsf{asrt}_2 \rangle$
Output : $(R, \mathsf{matcher}) \in \wp(\mathcal{Q}_1 \times \mathcal{Q}_2) \times \mathsf{Matcher}$
1 matcher $\leftarrow$ `newMatcher`(M_1, M_2)
2 **while** `moreMatchers`(matcher) **do**
3 $\sim \leftarrow$ `computeBisimulation`$(M_1, M_2, \mathsf{matcher})$
4 **if** `isValid`$(\sim)$ **then**
5 **return** $\langle \sim, \mathsf{matcher} \rangle$
6 **else**
7 matcher $\leftarrow$ `nextMatcher`(matcher)
8 **return** $\langle \emptyset, \emptyset \rangle$

relay on the datatype Matcher (not explicitly declared in our pseudo-code below) which provides, not only the implementation of the bijections acting as name matchings, but also data structures used to support the iteration of the plausible name matchings.

Given two e-CFSMs, the function e-CFSMBisimulation in Algorithm 1 returns a bisimulation with a corresponding name bijection (if any). The **while**-loop in Algorithm 1 iterates (via the auxiliary function `moreMatchers` that returns the next available name matching) on the possible name matchings (initialised on line 1 by the auxiliary function `newMatcher`). More precisely, for each such matcher function computeBisimulation defined in Algorithm 2 (cf. page 14) is invoked to check that a bisimulation under the provided matcher exists. The loop terminates either finding a valid bisimulation under a suitable name matching (returned in the structure matcher) or reporting that there is no more name matchings to iterate (when the auxiliary function `moreMatchers` returns *False* on matcher).

The function computeBisimulation in Algorithm 2 computes a bisimulation (if any) between two e-CFSMs given name matching. For readability, we fix

$$\widehat{\mathcal{Q}_1} = Q_1 \times \wp\Big(\bigcup_{t \in \rightarrow_1} \mathsf{asrt}_1(t) \cup \bigcup_{t \in \rightarrow_2} \mathsf{asrt}_2(\mathtt{match}(\mathbf{matcher}, t)) \Big)$$

$$\widehat{\mathcal{Q}_2} = Q_2 \times \wp\Big(\bigcup_{t \in \rightarrow_2} \mathsf{asrt}_2(t) \cup \bigcup_{t \in \rightarrow_1} \mathsf{asrt}_1(\mathtt{match}(\mathbf{matcher}^{-1}, t)) \Big)$$

where the `match`(matcher, t) is the transition obtained by renaming the variables occurring in the communication action and assertion of transition t according to matcher. The process starts from the total relation (see Line 1) and iteratively removes pairs of states when they are shown to be not bisimilar.[6]

For each pair in the current relation the bisimulation conditions are checked by invoking isSimulation (Algorithm 3, page 15) and checking the conditions

[6] Optimisations are possible by starting from smaller relations than the total one. For this reason, implementations actually start from a subset of the total relation by

Algorithm 2: Computation of a bisimulation modulo name matching

```
   Function computeBisimulation:
   |  Input  : M1 = ⟨(Q1, q01, →1), F1, qos1, asrt1⟩
   |  Input  : M2 = ⟨(Q2, q02, →2), F2, qos2, asrt2⟩
   |  Input  : matcher ∈ Matcher
   |  Output : newRel ⊆ Q̂1 × Q̂2
1  |  newRel ← Q̂1 × Q̂2
2  |  repeat
3  |  |  currentRel ← newRel
4  |  |  foreach ((p, K), (q, K')) ∈ currentRel do
5  |  |  |  if
   |  |  |  |  isSimulation(M1, M2, (p, K), (q, K'), currentRel, matcher) and
   |  |  |  |  not Sat(¬(qos1(p) ⟹ qos2(q))) and
   |  |  |  |  isSimulation(M2, M1, (q, K'), (p, K), currentRel, matcher^-1) and
   |  |  |  |  not Sat(¬(qos2(p) ⟹ qos1(q))) and
   |  |  |  |  (p ∈ F1 ⟺ q ∈ F2)
6  |  |  |  then
7  |  |  |  |  newRel ← newRel ∪ {((p, K), (q, K'))}
8  |  until newRel = currentRel
9  |  return newRel
```

on QoS specifications and state acceptance. More precisely, the algorithm searches for a simulation of M_1 with M_2 and a simulation of M_2 with M_1. It is worth observing that the result is a bisimulation relation where the variables in the assertions on the transitions of second machine M_2 are renamed by the name matching given in input. In fact, besides the renaming done in $\mathcal{Q}$, the simulation of M_2 with M_1 is computed using the inverse name matching as per the second conjunct of guard of the **if** statement on Line 5. When such conditions are violated, the next iteration continues without the pair of states under scrutiny; otherwise the pair is kept in the next current relation. In other words the states in the pair will be considered bisimilar in the next iteration. The function stops when a fix-point is reached.

The call to the function `Sat` in the guard of the **if** statement on Line 5 represents the invocation to an SMT-solver [5,12,14] for determining the satisfiability of a first-order formula expressing that the QoS specification of the state in the first machine does not imply the QoS specification of the state in the second machine.[7] Notice that a positive answer from the SMT-solver results in the conclusion that the corresponding states cannot are not bisimilar. Likewise, the states are not bisimilar if one of them is accepting while the other is not.

pruning as many pairs of "trivially" non-bisimilar states as possible, while keeping the computational complexity of the optimisation process low.

[7] In theory invocation to SAT solvers could be computationally expensive. However, as observed in [26], typically QoS constraints are trivial or not complex formulae that SAT solvers can decide efficiently.

Algorithm 3: Checking simulation conditions under a name match

Function `isSimulation`:
- **Input** : $M_1 = \langle (Q_1, q_{0_1}, \rightarrow_1), F_1, \mathsf{qos}_1, \mathsf{asrt}_1 \rangle$
- **Input** : $M_2 = \langle (Q_2, q_{0_2}, \rightarrow_2), F_2, \mathsf{qos}_2, \mathsf{asrt}_2 \rangle$
- **Input** : $(p, K) \in \widehat{Q_1}$
- **Input** : $(q, K') \in \widehat{Q_2}$
- **Input** : $\mathsf{matcher} \in \mathrm{Matcher}$
- **Input** : $R \subseteq \widehat{Q_1} \times \widehat{Q_2}$
- **Output** : bool

1 **foreach** $p \xrightarrow[\phi]{l} p' \in \rightarrow_1$ **do**
2 $\quad l' \leftarrow \mathtt{match}(\mathsf{matcher}, l)$
3 $\quad \mathsf{inCurrentRelation}, \mathsf{simKnw} \leftarrow \mathrm{true}, \mathrm{true}$
4 $\quad$ **foreach** $T \in \wp(\{t \in \rightarrow_2 \mid \textit{the source of } t \textit{ is } q \textit{ and the label of } t \textit{ is } l'\})$ **and** inCurrentRelation **and** simKnw **do**
5 $\quad\quad K_1 \leftarrow \mathtt{match}(\mathsf{matcher}, K \barwedge l \wedge \phi)$
6 $\quad\quad K_2 \leftarrow K' \barwedge l'$
7 $\quad\quad \mathsf{simKnw} \leftarrow \bigwedge_{q \xrightarrow[\psi]{l} q' \in T} \textbf{not}\left(\mathtt{Sat}(\neg((K_1 \wedge \psi) \implies (K_2 \wedge \psi)))\right)$
8 $\quad\quad \mathsf{candidates} \leftarrow \{((p', K \barwedge l \wedge \phi \wedge \mathtt{match}(\mathsf{matcher}^{-1}, \psi)), (q', K_2 \wedge \psi))\}_{q \xrightarrow[\psi]{l} q' \in T}$
9 $\quad\quad \mathsf{inCurrentRelation} \leftarrow \mathsf{candidates} \subseteq R$
10 $\quad$ **if not** simKnw **or not** inCurrentRelation **then**
11 $\quad\quad$ **return** *false*
12 **return** *true*

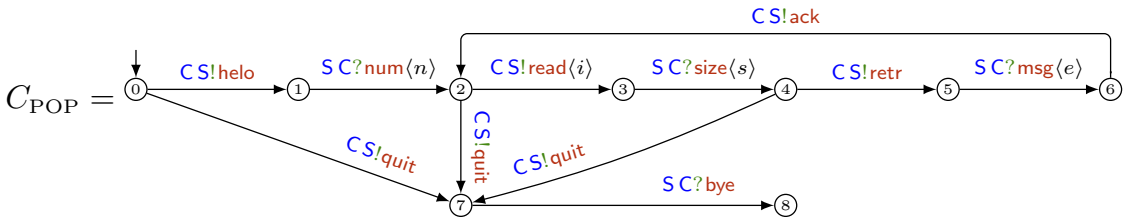
Algorithm 3 checks the simulation condition specified in Definition 10 on the states and relation passed as input to the function computeBisimulation. Function `match` (Line 2) performs the translation of a label according to a matcher; this function is polymorphically applied on the $FOL_=$ formula yielding the knowledge at state p (cf. Line 5). This renaming is necessary in order to check that the knowledge of the source state of a transition in the first machine, is simulated by the knowledge of the source state of a transition in the second machine. For this we appeal to an SMT-solver (cf. the invocation to `Sat` on Line 7).

4 Case-Study: A Variant of the POP Protocol

We now apply both our framework to the POP protocol [3] that allows mail clients to access a remote mailbox and retrieve e-mails. More precisely, a client C starts the communication by sending an helo message[8] to a POP server S. The

server replies with the number of unread messages in the mailbox using a num message. At this point C can either halt the protocol by sending a quit message to S or read one of the e-mails by sending a read message to S. In the former case, the server acknowledges with a bye message and the protocol ends. In the latter case, S sends C the number of bytes of the current unread message in a size message letting the client choose between quitting the protocol (as before) or actually requesting an e-mail by sending S a retr message. In the latter case the server sends a msg message (whose payload is meant to be the selected e-mail); the reception is acknowledged by the client with an ack message which re-starts the reading process.

We elaborate on the POP client's used in [24] by adding payloads on some as the following CFSM:



Being POP a two-party protocol, a CFSM for the server is obtained by *dualising* the CFSM above, that is by replacing each send action with the corresponding receive action and vice versa.

To make C_{POP} above an e-CFSM we simply assign QoS contracts to states and assertions to transitions. The former rely on the QoS specification in [24] which considers the set of QoS attributes $\mathsf{Q} = \{\mathsf{t}, \mathsf{c}, \mathsf{m}\}$ representing CPU time, monetary cost, and memory usage, respectively and based on the following QoS:[9]

$$
\begin{aligned}
\Gamma_{\text{Low}} &= \{\mathsf{t} \leq 0.01,\ \mathsf{c} \leq 0.01,\ \mathsf{m} \leq 0.01\} \\
\Gamma_{\text{Chk}} &= \{\mathsf{t} \leq 5,\ \mathsf{c} = 0.5,\ \mathsf{m} = 0\} \\
\Gamma_{\text{Mem}} &= \{1 \leq \mathsf{t} \leq 6,\ \mathsf{c} = 0,\ \mathsf{m} \leq 64\} \\
\Gamma_{\text{DB}} &= \{\mathsf{t} \leq 3 \implies (\exists x)(0.5 \leq x \leq 1 \wedge \mathsf{c} = \mathsf{t} \cdot x),\ \mathsf{t} > 3 \implies \mathsf{c} = 10,\ \mathsf{m} \leq 5\}
\end{aligned}
$$

to respectively characterise the phases of the protocol with low computational cost, the cost for checking the integrity of emails, the cost related to reception of messages from the server, and the monetary cost according to the execution time required by an operation.

[8] We stick to the naming convention for messages in [3].
[9] These QoS specifications are representative of typical constraints defined in SLAs.

Following Definition 6 and letting $t_{h,k}$ be the transition in C_{POP} from state h to state k we can define:

$$\mathsf{qos}_{\text{POP}} : j \mapsto \begin{cases} \Gamma_{\text{Low}}, & j \in \{0,1,3,7,8\} \\ \Gamma_{\text{DB}}, & j \in \{2,4\} \\ \Gamma_{\text{Mem}}, & j = 5 \\ \Gamma_{\text{Chk}}, & j = 6 \end{cases}$$

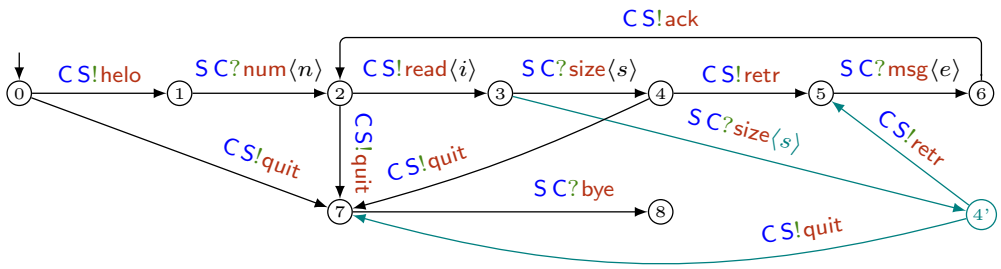
and

$$\mathsf{asrt}_{\text{POP}} : t_{h,k} \mapsto \begin{cases} n \geq 0, & h = 1, k = 2 \\ 0 < i \leq n, & h = 2, k = 3 \\ s \geq 0, & h = 3, k = 4 \\ \texttt{len}(e) = s, & h = 5, k = 6 \\ \mathit{True}, & \mathit{otw} \end{cases}$$

Note that Γ_{Low} is assigned to most of the states. As observed in [24], this is typical since most of the state in an e-CFSM represent phases of computations that do not have a significant impact on the non-functional properties of applications.

Let us now consider another POP client that treats differently emails with size below a given bound, say $\hat{s}$, than those with size bigger than $\hat{s}$. We can specify such client with the e-CFSM C'_{POP} below:

$C'_{\text{POP}} =$



obtained by adding the coloured part to C_{POP} and defining QoS specification $\mathsf{qos}'_{\text{POP}}$ and assertions $\mathsf{asrt}'_{\text{POP}}$ as:

$$\mathsf{qos}'_{\text{POP}} = \mathsf{qos}_{\text{POP}}[4' \mapsto \mathsf{qos}_{\text{POP}}(4)] = \Gamma_{\text{DB}} \quad \text{and}$$
$$\mathsf{asrt}'_{\text{POP}} = \mathsf{asrt}_{\text{POP}}[t_{3,4} \mapsto \hat{s} > 0 \wedge 0 \leq s \leq \hat{s}, t_{3,4'} \mapsto s > \hat{s} \wedge \hat{s} > 0]$$

For instance, 4' is a state where an internal computation checks whether the necessary disk space is available or not. It is easy to see that C_{POP} and C'_{POP} are bisimilar since

- the QoS specification in states 4 of C_{POP} and 4' in C'_{POP} are the same,
- the transition from state 3 to state 4 C_{POP} is simulated by the two transitions from state 3 in C'_{POP} and

– each transition from state 3 in C'_{POP} is simulated by the transition from state 3 to state 4 C_{POP} since the assertion on the former implies the assertion on the latter.

Note that the bisimulation relation would not hold if we had assigned a QoS specification of 4' different than the QoS specification of 4 in C_{POP}, namely if we had $\mathsf{qos}'_{\text{POP}}(4') \neq \mathsf{qos}_{\text{POP}}(4)$. Likewise, changing the assertion of $t_{3,4}$ in C'_{POP} to $\hat{s} > 0 \wedge 0 \leq s < \hat{s}$ would spoil the bisimilarity relation with C_{POP} the implication $s \geq 0 \implies (\hat{s} > 0 \wedge 0 \leq s < \hat{s}) \vee (s > \hat{s} \wedge \hat{s} > 0)$ (since for $s = \hat{s}$ the antecedent holds while the consequence does not).

5 Conclusions, Related and Future Work

We define *extended-CFSM* (e-CFSMs for short), an automata model for expressing service contracts. Our model blends the variants of CFSMs [9] recently introduced in [18] and in [24,28]. Like those variants, behavioural contracts are formalised by the communication pattern described by the underlying CFSM; functional contracts are essentially borrowed from *asserted* CFSMs [18], while non-functional contracts are formalised in terms of the QoS specifications introduced in [24,28].

We elaborate the framework studied in [31] to introduce a bisimulation *up-to name matching* (cf. Definition 10), designed to tackle the practical problem of name mismatching in contracts specified by independent parties that very likely do not share the same naming convention. This problem is commonplace in the reference application driving our contributions: transparent dynamic discovery and binding of service-based software artefacts. In this context, our framework contributes to make the run-time selection of services more precise –since our e-CFSMs comprehensively account for behavioural, functional, and non-functional constraints– and more flexible –since interoperability can be guaranteed also when provision and required contracts use mismatching names. Finally, we showed the use of the language through enriching a case-study taken from the literature [24].

Besides a slightly different view[10] we use the asserted CFSMs in [18] for capturing functional contracts. Asserted CFSMs build on choreography automata [4] to generalise the notion of choice usually adopted in behavioural types and neither consider interoperability nor QoS contracts. We remark that the notion of history sensitivity was introduced in [8] and inspired both the notion of history sensitivity in [18] and ours (cf. Definition 8). On the contrary, our notion of knowledge differs from those in [8] and in [18] since our notion not only consider which variables are available to a participant at a given point of the computation, but also which assertions on such variables the participant can safely assume. The first model for functional contracts for multiparty session types is

[10] For technical reasons we use typed messages, namely we assume a function $\sigma : \mathcal{M} \to \mathtt{S}^*$ mapping every $\mathsf{m} \in \mathcal{M}$ to its corresponding type.

presented in [8]. More recently, ideas similar to the one in [8] have been developed in [37] to propose refined multiparty session types. As asserted CFSMs, these models are designed to guarantee communication correctness in presence of data dependencies. In fact, we implicitly rely on the results in [18] to distil contracts as e-CFSMs obtained by decorating with QoS specifications asserted CFSMs resulting by projection of well-formed asserted choreography so to ensure that our behavioural contracts are communication-sound.

The extension of CFSMs to QoS specifications introduced in [24,28] aimed to statically analyse SLAs in the composition of services. The analysis in [24,28] hinges on *aggregation operators* that are not used in the present work since here we only employ QoS specifications in order to guarantee SLAs in the run-time discovery of services. Functional contracts are not considered in [24,28].

The literature on QoS spans a wide range of contexts and methods [2,19]. As in [24,28], we abstractly model application-level QoS using RCFs. Other abstract models in the literature such as quantales [30] and c-semirings [10,13,23], are not amenable of being used in modern SMT-solvers [7, Chapter 33] (which we need in the algorithms presented in Sect. 3.3). The language for QoS properties in [29] is restricted on convex polytopes; this restriction does not affect RCFs. In [36] CFSMs are the basis for dynamic binding mechanism of services and can only capture behavioural contracts.

Our approach has two main limitations that we plan to overcome in future work. A first limitation is that bisimulation could be too a rigid mechanism for service interoperability. Besides using a weak notion of bisimulation, we plan to replace bisimulation with simulation relations akin to behavioural subtyping to have a more general mechanism. A second limitation is to consider name matching mechanisms such as name *embeddings* rather that bijections.

We plan to integrate the models and algorithms proposed in this work in a common framework which includes SEArch and MoCheQos [25] to provide more general service brokerage mechanisms. In fact, this will allow to obtain sound e-CFSMs by projecting global choreographic models featuring behavioural, functional, and non-functional contracts in the vein of [21] where QoS contracts of components are derived from global specifications for run-time prediction, adaptive composition, or compliance checking, similarly to what is done in [22]. As said, this top-down approach is indeed featured in [18] for functional contracts.

Once our discovery mechanism is implemented, we plan to assess its effectiveness with an experimental evaluation. We expect that for "small" contracts (such as those commonplace in microservices) the approach can be feasibly used for on-the-fly discovery. In cases where the compliance check is computationally heavy, the approach can be still applied by making use of off-line discovery whereby brokers prepare lists of available services compatible with expected request.

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Pearson Education Inc. (2006)

2. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: a systematic literature review. IEEE Trans. Softw. Eng. **39**, 658–683 (2013)
3. Anonymous: Post office protocol: Version 2. On-line (1985). https://rfc-editor.org/rfc/rfc937.txt
4. Barbanera, F., Lanese, I., Tuosto, E.: Choreography automata. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION 2020. LNCS, vol. 12134, pp. 86–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50029-0_6
5. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) TACAS 2022. LNCS, vol. 13243, pp. 415–442. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99524-9_24
6. Beachy, J.A., Blair, W.D.: Abstract Algebra, 3rd edn. Waveland Press Inc. (2006)
7. Biere, A., Heule, M., Van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability: Second Edition, Frontiers in Artificial Intelligence and Applications, vol. 336. IOS Press (2021). https://doi.org/10.3233/FAIA336
8. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 162–176. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_12
9. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983)
10. Buscemi, M.G., Montanari, U.: CC-Pi: a constraint-based language for specifying service level agreements. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 18–32. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71316-6_3
11. World Wide Web Consortium: Web services description language (WSDL) version 2.0 part 1: Core language. On-line. https://www.w3.org/TR/wsdl20/
12. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
13. De Nicola, R., Ferrari, G., Montanari, U., Pugliese, R., Tuosto, E.: A process calculus for QoS-aware applications. In: Jacquet, J.-M., Picco, G.P. (eds.) COORDINATION 2005. LNCS, vol. 3454, pp. 33–48. Springer, Heidelberg (2005). https://doi.org/10.1007/11417019_3
14. Biere, A., Bloem, R. (eds.): CAV 2014. LNCS, vol. 8559. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9
15. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press (1972)
16. Erdös, P., Gillman, L., Henriksen, M.: An isomorphism theorem for real-closed fields. Ann. Math. **61**(3), 542–554 (1955)
17. Fiadeiro, J.L., Lopes, A., Bocchi, L.: An abstract model of service discovery and binding. Formal Aspects Comput. **23**(4), 433–463 (2011)
18. Gheri, L., Lanese, I., Sayers, N., Tuosto, E., Yoshida, N.: Design-by-contract for flexible multiparty session protocols. In: Ali, K., Vitek, J. (eds.) Proceedings of 36th European Conference on Object-Oriented Programming, ECOOP 2022. LIPIcs, vol. 222, pp. 8:1–8:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
19. Hayyolalam, V., Kazem, A.: A systematic literature review on QoS-aware service composition and selection in cloud environment. J. Netw. Comput. Appl. **110**, 52–74 (2018)
20. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM **32**(1), 137–161 (1985)

21. Ivanović, D., Carro, M., Hermenegildo, M.V.: A constraint-based approach to quality assurance in service choreographies. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service-Oriented Computing, pp. 252–267. Springer, Heidelberg (2012)
22. Kattepur, A., Georgantas, N., Issarny, V.: QoS analysis in heterogeneous choreography interactions. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, pp. 23–38. Springer, Heidelberg (2013)
23. Lluch-Lafuente, A., Montanari, U.: Quantitative μ-calculus and CTL based on constraint semirings. Electron. Notes Theor. Comput. Sci. **112**, 37–59 (2005). https://doi.org/10.1016/j.entcs.2004.02.063. https://www.sciencedirect.com/science/article/pii/S1571066104052417. Proceedings of the Second Workshop on Quantitative Aspects of Programming Languages (QAPL 2004)
24. Lopez Pombo, C.G., Martinez Suñé, A.E., Tuosto, E.: A dynamic temporal logic for quality of service in choreographic models. In: Ábrahám, E., Dubslaff, C., Tarifa, S.L.T. (eds.) Proceedings of 20th International Colloquium on Theoretical Aspects of Computing - ICTAC 2023. LNCS, vol. 14446, pp. 119–138. Springer, Lima (2023)
25. Lopez Pombo, C.G., Martinez Suñé, A.E., Tuosto, E.: MoCheQoS: automated analysis of quality of service properties of communicating systems. On-line (2023). https://arxiv.org/abs/2311.01415
26. Lopez Pombo, C.G., Martinez Suñé, A.E., Tuosto, E.: Automated static analysis of quality of service properties of communicating systems. In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds.) Proceedings of 26th. International Symposium on Formal Methods (FM 2024) - Part II. LNCS, vol. 14934, pp. 84–103. Springer, Cham (2025)
27. Lopez Pombo, C.G., Montepagano, P., Tuosto, E.: SEArch: an execution infrastructure for service-based software systems. In: Castellani, I., Tiezzi, F. (eds.) Proceedings of Coordination Models and Languages - 26nd IFIP WG 6.1 International Conference, COORDINATION 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024. LNCS, vol. 14676, pp. 314–330. Springer, Cham (2024)
28. Martinez Suñé, A.E.: Formalización y análisis de requerimientos no-funcionales cuantificables. Ph.D. thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires (2024). Advisor: Carlos G. Lopez Pombo
29. Martinez Suñé, A.E., Lopez Pombo, C.G.: Automatic quality-of-service evaluation in service-oriented computing. In: Riis Nielson, H., Tuosto, E. (eds.) COORDINATION 2019. LNCS, vol. 11533, pp. 221–236. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22397-7_13
30. Rosenthal, K.I.: Quantales and Their Application. Pitman Research Notes in Mathematics Series, vol. 234. Longman Scientific & Technical (1990)
31. Anabia, D.N.S.: Bisimulación de Data-aware Communicating Finite State Machines con propiedades en las acciones. Master's thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires,: advisors: Carlos G. Lopez Pombo and Hernán C, Melgratti (2023)
32. Smullyan, R.M.: First-Order Logic. Dover Publishing (1968)
33. Tarski, A.: A decision method for elementary algebra and geometry. Memorandum RM-109, RAND Corporation (1951), later published in [34] and reprinted in [35]
34. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press (1951), originally published in [33] and reprinted in [35]
35. Tarski, A.: A Decision Method for Elementary Algebra and Geometry, pp. 24–84. Texts and Monographs in Symbolic Computation, Springer, Cham (1998), originally published in [33] and reprinted from [34]

36. Vissani, I., Lopez Pombo, C.G., Tuosto, E.: Communicating machines as a dynamic binding mechanism of services. In: Gay, D., Alglave, J. (eds.) Proceedings of 8th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES. Electronic Proceedings in Theoretical Computer Science, vol. 203, pp. 85–98 (2016)
37. Zhou, F., Ferreira, F., Hu, R., Neykova, R., Yoshida, N.: Statically verified refinements for multiparty protocols. In: OOPSLA 2020: Conference on Object-Oriented Programming Systems, Languages and Applications. p. 30 pages. No. OOPSLA (Article 148) in PACMPL. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3428216