Project no.:	**PIRSES-GA-2011-295261**
Project full title:	**Mobility between Europe and Argentina applying Logics to Systems**
Project Acronym:	**MEALS**
Deliverable no.:	**5.4 / 3**
Title of Deliverable:	**Communicating machines as a dynamic binding mechanism of services**

Contractual Date of Delivery to the CEC:	**31-Mar-2015**
Actual Date of Delivery to the CEC:	**31-Mar-2015**
Organisation name of lead contractor for this deliverable:	**UBA**
Author(s):	**Ignacio Vissani, Carlos Gustavo Lopez Pombo, Emilio Tuosto**
Participants(s):	**UBA, ULEIC**
Work package contributing to the deliverable:	**WPN**
Nature:	**R**
Dissemination Level:	**Public**
Total number of pages:	**15**
Start date of project:	1 Oct. 2011 **Duration:** 48 month

Abstract:

Distributed software is becoming more and more dynamic to support applications able to respond and adapt to the changes of their execution environment. For instance, *service-oriented computing* (SOC) envisages applications as services running over globally available computational resources where discovery and binding between them is transparently performed by a middleware. *Asynchronous Relational Networks* (ARNs) is a well-known formal orchestration model, based on hypergraphs, for the description of service-oriented software artefacts. Choreography and orchestration are the two main design principles for the development of distributed software. In this work, we propose *Communicating Relational Networks* (CRNs), which is a variant of ARNs, but relies on choreographies for the characterisation of the communicational aspects of a software artefact, and for making their automated analysis more efficient.

Note:

This deliverable is based on material that has been published as "Ignacio Vissani, Carlos G. Lopez Pombo and Emilio Tuosto. Communicating machines as a dynamic binding mechanism of services. In Alastair Beresford, Simon Gay, Alan Mycroft, Vasco T. Vasconcelos and Nobuko Yoshida (Eds.): Proceedings of the 8th Programming Language Approaches to Concurrency- and Communication-cEntric Software - PLACES 2015, London, United Kingdom, April 18, 2015."

This project has received funding from the European Union Seventh Framework Programme (FP7 2007-2013) under Grant Agreement Nr. 295261.

Contents

1 Introduction and motivation

Distributed software is becoming more and more dynamic to support applications able to respond and adapt to the changes of their execution environment. For instance, *service-oriented computing* (SOC) envisages applications as services running over globally available computational resources; at run-time, services search for other services to bind to and use. Software architects and programmers have no control as to the nature of the components that an application can bind to due to the fact that the discovery and binding are transparently performed by a middleware.

Choreography and orchestration are the two main design principles for the development of distributed software (see e.g., [6]). Coordination is attained in the latter case by an *orchestrator*, specifying (and possibly executing) the distributed work-flow. Choreography features the notion of *global view*, that is a holistic specification describing distributed interactions amenable of being "projected" onto the constituent pieces of software. In an orchestrated model, the distributed computational components coordinate with each other by interacting with a special component, *the orchestrator*, which at run time decides how the work-flow has to evolve. For example the orchestrator of a service offering the booking of a flight and a hotel may trigger a service for hotel and one for flight booking in parallel, wait for the answers of both sites, and then continue the execution. In a choreographed model, the distributed components autonomously execute and interact with each other on the basis of a local control flow expected to comply with their role as specified in the "global viewpoint". For example, the choreography of hotel-flight booking example above could specify that the flight service interacts with the hotel service which in turns communicates the results to the buyer.

We use Asynchronous relational networks (ARNs) [8] as the basis of our approach. In ARNs, systems are formally modelled as hypergraphs obtained by connecting hyperarcs which represent unit of computations and communication. More precisely, hyperarcs are interpreted as either processes (*services* or unit of computation) or as communication channels (unit of communication). The nodes can only be adjacent to: 1. 1. one process hyperarc and one communication hyperarc, meaning that the computation formalised by the process hyperarc is bound through the communication channel formalised by the communication hyperarc, 2. one process hyperarc, meaning that it is a *provides-points* through which the computation formalised by the process hyperarc can be bound to and activity that requires that particular service, or 3. one communication hyperarc, meaning that it is a *requires point* to which a given service can be bound using one of its provides-points. The rationale behind this separation is that a provides-point yields the interface through which a service exports its functionality while a requires-point is the interface through which an activity expects certain service to provide a functionality. Composition of services can then be understood as fusing a provides-point with a requires-point in a way that the service exported by the former satisfy the expectations of the latter, usually formalised as contracts in some formal language.

Hyperarcs are labelled with (Müller) automata; in the case of process hyperarcs, automata formalise the interactions carried out by that particular service while, in the case of communication hyperarcs, they represent the orchestrator coordinating the behaviour of the participants of the communication. In fact, the automaton Λ associated to a communication hyperarc co-

ordinates the processes bound to its ports by, at each time, interacting with one of the processes and deciding, depending on the state Λ is in, what is the next interaction (if any) to execute. The global behaviour of the system is then obtained by composing the automata associated to process and communication hyperarcs. In the forthcoming sections we will introduce a running example to show how definitions work and concretely discuss the contributions of the present work.

As anticipated, the composition of ARNs yields a semantic definition of a binding mechanism of services in terms of "fusion" of provides-points and requires-points. Once coalshed, the nodes become "internal", that is they are no longer part of the interface and cannot be used for further bindings. In existing works, like [8], the binding is subject to an entailment relation between *linear temporal logic* [7] theories attached to the provides- and requires-points that can be checked by resorting to any decision procedure for LTL (for example, [4])

Although the orchestration model featured by ARNs is rather expressive and versatile, we envisage two drawbacks:

1. the binding mechanism based on LTL-entailment establishes an asymmetric relation between requires-point and provides-point as it formalises a notion of trace inclusion; also,

2. including explicit orchestrators (the automaton labelling the communication hyperarcs), in the composition, together with the computational units (the automaton labelling the process hyperarcs) increases the size of the resulting automaton making the analysis more expensive.

In the present work we propose *Communicating Relational Networks* (CRNs), a variant of ARNs relying on choreographies to overcome those issues, where provides-points are labelled with *Communicating Finite State Machines* [2] declaring the behaviour (from the communication perspective) exported by the service, and communication hyperarcs are labelled with *Global Graphs* [3] declaring the global behaviour of the communication channel. In this way, our proposal blends the orchestration framework of ARNs with a choreography model based on global graphs and communicating machines. Unlike most of the approaches in the literature (where choreography and orchestration are considered antithetical), we follow a comprehensive approach showing how choreography-based mechanisms could be useful in an orchestration model.

The present work is organised as follows; in Section 2 we provide the formal definitions of most of the concepts used along this paper. Such definitions are illustrated with a running example introduced in Section 3. In Section 4 we introduce the main contribution of this paper, being the definition of CRNs, we show how they are used to rewrite the running example and we discuss several aspects regarding the design-time checking to assert internal coherence of services, the run-time checking ruling the binding mechanism and the cost of software analysis. Finally, in Section 5 we draw some conclusions and discuss some further research directions.

2 Preliminaries

In this section we present the preliminary definitions used throughout the rest of the present work. We first summarise communicating machines and global graphs borrowing definitions

from [5] and from [3]. Finally we introduce some basic definitions in order to present ARNs; the definition here are adapted from [8].

2.1 Communicating machines and global graphs

Communicating machines were introduced in [2] to model and study communication protocols in terms of finite transition systems capable of exchanging messages through some channels. We fix a finite set Msg of *messages*, a finite set P of participants

Definition 1 ([2])**.** A *communicating finite state machine* on Msg (CFSMs, for short) is a finite transition system $(\mathsf{C}, \mathsf{C}, q_0, \mathsf{Msg}, \delta)$ where

- C is a finite set of states;
- $\mathsf{C} = \{\mathsf{pq} \in \mathsf{P}^2 \mid \mathsf{p} \neq \mathsf{q}\}$ is a set of channels;
- $q_0 \in \mathsf{C}$ is an initial state;
- $\delta \subseteq \mathsf{C} \times (\mathsf{C} \times \{!, ?\} \times \mathsf{Msg}) \times \mathsf{C}$ is a finite set of *transitions*.

A *communicating system* is a map S assigning a CFSM $S(\mathsf{p})$ to each $\mathsf{p} \in \mathsf{P}$. We write $q \in S(\mathsf{p})$ when q is a state of the machine $S(\mathsf{p})$ and likewise and $t \in S(\mathsf{p})$ when t is a transition of $S(\mathsf{p})$.

The execution of a system is defined in terms of transitions between configurations as follows:

Definition 2. The *configuration* of communicating system S is a pair $s = (\vec{q}, \vec{w})$ where $\vec{q} = (q_\mathsf{p})_{\mathsf{p}\in\mathsf{P}}$ where $q_\mathsf{p} \in S(\mathsf{p})$ for each $\mathsf{p} \in \mathsf{P}$ and $\vec{w} = (w_\mathsf{pq})_{\mathsf{pq}\in\mathsf{C}}$ with $w_\mathsf{pq} \in \mathsf{Msg}^\star$. A configuration $s' = (\vec{q'}, \vec{w'})$ is *reachable* from another configuration $s = (\vec{q}, \vec{w})$ by the *firing of the transition* t (written $s \xrightarrow{t} s'$) if there exists $\mathsf{m} \in \mathsf{Msg}$ such that either:

1. $t = (q_\mathsf{p}, \mathsf{pq}!\mathsf{m}, q'_\mathsf{p}) \in \delta_\mathsf{p}$ and
   (a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ for all $\mathsf{p}' \neq \mathsf{p}$; and
   (b) $w'_\mathsf{pq} = w_\mathsf{pq} \cdot \mathsf{m}$ and $w'_{\mathsf{p}'\mathsf{q}'} = w_{\mathsf{p}'\mathsf{q}'}$ for all $\mathsf{p}'\mathsf{q}' \neq \mathsf{pq}$; or
2. $t = (q_\mathsf{q}, \mathsf{pq}?\mathsf{m}, q'_\mathsf{q}) \in \delta_\mathsf{q}$ and
   (a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ for all $\mathsf{p}' \neq \mathsf{q}$; and
   (b) $\mathsf{m} \cdot w'_\mathsf{pq} = w_\mathsf{pq}$ and $w'_{\mathsf{p}'\mathsf{q}'} = w_{\mathsf{p}'\mathsf{q}'}$ for all $\mathsf{p}'\mathsf{q}' \neq \mathsf{pq}$

A *global graph* is a finite graph whose nodes are labelled over the set $\mathsf{L} = \{\bigcirc, \circledcirc, \oplus, \boxplus\} \cup \{\mathsf{s} \to \mathsf{r} : \mathsf{m} \mid \mathsf{s}, \mathsf{r} \in \mathsf{P} \wedge \mathsf{m} \in \mathsf{Msg}\}$ according to the following definition.

Definition 3. A *global graph* (over P and Msg) is a labelled graph $\langle V, A, \Lambda\rangle$ with a set of *vertexes* V, a set of *edges* $A \subseteq V \times V$, and *labelling function* $\Lambda : V \to \mathsf{L}$ such that $\Lambda^{-1}(\bigcirc)$ is a singleton and, for each $v \in V$

1. if $\Lambda(v)$ is of the form $\mathsf{s} \to \mathsf{r} : \mathsf{m}$ then v has a unique incoming and unique outgoing edges,
2. if $\Lambda(v) \in \{\oplus, \boxplus\}$ then v has at least one incoming edge and one outgoing edge and,
3. $\Lambda(v) = \circledcirc$ then v has zero outgoing edges.

Label $\mathsf{s} \to \mathsf{r} : \mathsf{m}$ represents an interaction where machine s sends a message m to machine r. A vertex with label $\bigcirc$ reperesents the source of the global graph, $\circledcirc$ represents the termination of a branch or of a thread, $\boxplus$ indicates forking or joining threads, and $\oplus$ marks vertexes corresponding to branch or merge points, or to entry points of loops.

In the following we use a projections algorithms that given a global graph retrieves communicating machines for each of its participants. Undestranding such algorithm is not necessary for the sake of this paper and the interested reader is referred to [5] for its definition.

2.2 Asynchronous relational networks

A Müller automaton is a finite state automaton where final states are replaced by a family of states to define an acceptance condition on infinite words.

Definition 4 (Müller automaton)**.** A *Müller automaton* over a finite set A of *actions* is a structure of the form $\langle Q, A, \Delta, I, \mathcal{F} \rangle$, where

1. Q is a finite set (of *states*)
2. $\Delta \subseteq Q \times A \times Q$ is a *transition relation* (we write $p \xrightarrow{\iota} q$ when $(p, \iota, q) \in \Delta$),
3. $I \subseteq Q$ is the set of *initial states*, and
4. $\mathcal{F} \subseteq 2^Q$ is the set of *final-state sets*.

We say that an automaton *accepts* an inifinite trace $\omega = q_0 \xrightarrow{\iota_0} q_1 \xrightarrow{\iota_1} \ldots$ if and only if $q_0 \in I$ and there exists $i \geq 0$ and $S \in \mathcal{F}$ such that for all $s \in S$, the set $\bigcup_{i \leq j \wedge q_j = s} \{j\}$ is infinite.

Asynchronous relational networks are hypergraphs connecting *ports* that can be thought of as communication end-points through which messages can be sent to or received from other ports.

Definition 5 (Port)**.** A *port* is a structure $\pi = \langle \pi^+, \pi^- \rangle$ where π^+, π^- are disjoint finite sets of messages. We say that two ports are disjoint when they are formed by componentwise disjoint sets of messages. The *actions over* π are $A_\pi = \{m! \mid m \in \pi^-\} \cup \{m¿ \mid m \in \pi^+\}$.

The computational agents of ARNs are *processes* formalised as a set of ports togetherr with a Müller automaton describing the communication pattern of the agents.

Definition 6 (Process)**.** A *process* $\langle \gamma, \Lambda \rangle$ consists of a set γ of pairwise disjoint ports and a Müller automaton Λ over the set of actions $A_\gamma = \bigcup_{\pi \in \gamma} A_\pi$.

Processes are connected through *connections* whose basic role is to establish relations among the messages that processes exchange on the ports of processes and communication hyperedges. Intuitively, one can thing of the messages used by processes and communication hyperedges as 'local' messages whose 'global' meaning is established by connections.

Definition 7 (Connection)**.** Given a set of pairwise disjoint ports γ, an *attachment injection* on γ is a pair $\langle M, \mu \rangle$ where and a finite set M of messages and $\mu = \{\mu_\pi\}_{\pi \in \gamma}$ is a family of finite partial injections $\mu_\pi \colon M \rightharpoonup \pi^- \cup \pi^+$. We say that $\langle M, \mu, \Lambda \rangle$ is a *connection* on γ iff $\langle M, \mu \rangle$ is an attachment injection on γ and a Müller automaton Λ over $\{m! \mid m \in M\} \cup \{m¡ \mid m \in M\}$ such that:

$$\mu_\pi^{-1}(\pi^-) \subseteq \bigcup_{\hat{\pi} \in \gamma \setminus \{\pi\}} \mu_{\hat{\pi}}^{-1}(\hat{\pi}^+) \qquad \text{and} \qquad \mu_\pi^{-1}(\pi^+) \subseteq \bigcup_{\hat{\pi} \in \gamma \setminus \{\pi\}} \mu_{\hat{\pi}}^{-1}(\hat{\pi}^-).$$

for each $\pi \in \gamma$.

Definition 8 (Asynchronous Relational Network [8])**.** Let M be a finite set of messages. An *asynchronous relational net* α on M is a structure $\langle X, P, C, \{\pi_x\}_{x \in X}, \{\mu_c\}_{c \in C}, \{\gamma\}_{x \in X}, \{\Lambda_e\}_{e \in P \cup C} \rangle$ where

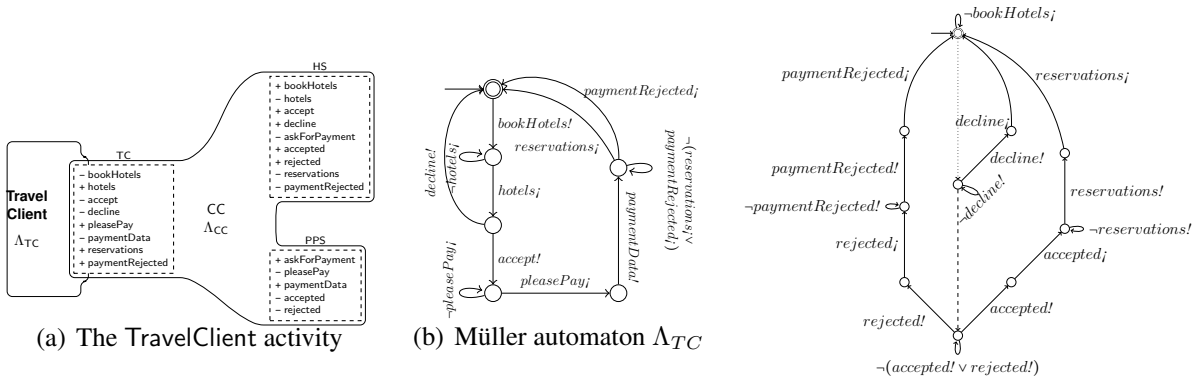
- $\langle X, P \cup C \rangle$ is a hypergraph, with X is a (finite) set of vertexes, P is a set of *hyperedges* (non-empty subsets of X) *computation hyperedges*, and C is a set of *communication hyperedges* such that X, P, and C are pairwise disjoint, no adjacent hyperedges belong to the same partition,
- three labelling functions that assign (*a*) a port π_x with messages in M to each point $x \in X$, (*b*) a process $\langle \gamma_p, \Lambda_p \rangle$ to each hyperedge $p \in P$ such that $\gamma_p \subseteq \{\pi_x\}_{x \in X}$, and (*c*) a connection $\langle M_c, \mu_c, \Lambda_c \rangle$ to each hyperedge $c \in C$.

An ARN with no provides-point is called *activity* and formalises the notion of a software artefact that can execute, while an ARN that has at least one provides-point is called a *service* and can only execute provided it is bound through one of them to a requires-point of an *activity*.

3 The running example

The following running example will help us to present intuitions behind the definitions, and later, to introduce and motivate our contributions. Consider an application providing the service of hotel reservation and payment processing. A client activity TravelClient asks for hotel options made available by a provider HotelsService returning a list of offers. If the client accepts any of the offers, then HotelsService calls for a payment processing service PaymentProcessService which will ask the client for payment details, and notify HotelsService whether the payment was accepted or rejected. Finally, HotelsService notifies the outcome of the payment process to the client.

Figures 1, 2, and 3 show the ARNs (including the automata), for the TravelClient, HotelsService, and PaymentProcessService respectively. The ARN in Fig. 1(a) represents an activity composed with a communication channel. More precisely, TravelClient (in the solid box on the left) represents a process hyperedge whose Müller automaton is Λ_{TC} (depicted in Fig. 1(b)). The solid



(a) The TravelClient activity (b) Müller automaton Λ_{TC} (c) Müller automaton Λ_{CC}.

Figure 1: The TravelClient activity together with the Müller automata.

"y-shaped" contour embracing the three dashed boxes represents a communication hyperedge used to specify the two requires-points (i.e., HS and PPS) of the component necessary to fulfill its goals. Note that such ARN does not provide itself any service to other components and that the dashed box lists the outgoing and incoming messages expected (respectively denoted by names prefixed by '+' and '-' signs).

It is worth remarking that communication hyperarcs in ARNs yield the coordination mechanism among a number of services. In fact, a communication hyperarc enables the interaction among the services that bind to its requires-points such as TravelClient, HotelsService, and PaymentProcessService in our example. The coordination is specified through a Müller automaton associated with the communication hyperarc that acts as the orchestrator of the servises. In our running example, the communication hyperarc of Fig. 1 is labeled with the automaton Λ_{CC} of Fig. 1(c) where, for readability and conciseness, the dotted and dashed edges stand for the paths

$$\xrightarrow{bookHotels!} \cdot \xrightarrow{bookHotels_i} \cdot \xrightarrow{hotels!} \cdot \xrightarrow{hotels_i}$$

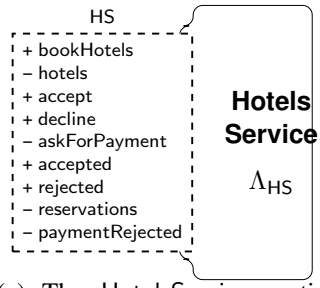
and

$$\xrightarrow{accept!} \cdot \xrightarrow{accept_i} \cdot \xrightarrow{askForPayment!} \cdot \xrightarrow{askForPayment_i} \cdot \xrightarrow{paymentData!} \cdot \xrightarrow{paymentData_i}$$

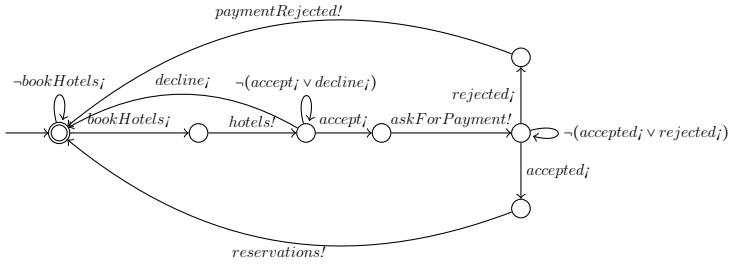
respectively. As we will see, such automaton corresponds to a global choreography when replacing the binding mechanism of ARNs with choreography-based mechanisms. The transitions of the automata are labelled with input/output actions; according to the usual ARNs notation, a label $m!$ stands for the ouput of message m while label m_i stands for the input of message m.

Figures 2 and 3 represent two services with their automata (resp. Λ_{HS} and Λ_{PPS}) and their provides-point (resp. HS and PPS) not bound to any communication channel yet.

The composition of ARNs yields a semantic definition of a binding mechanism of services in terms of "fusion" of provides-points and requires-points. More precisely, the binding is subject

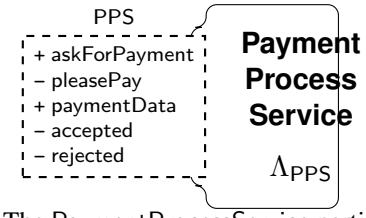


(a) The HotelsService participant

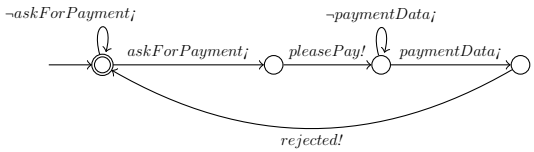


(b) Müller automaton Λ_{HS}

Figure 2: The HotelsService participant together with the machine Hs



(a) The PaymentProcessService participant



(b) Müller automaton Λ_{PPS}, that only reject paymens

Figure 3: The PaymentProcessService participant.

to an entailment relation between *linear temporal logic* [7] theories attached to the provides- and requires-points as illustrated in the following.

4 Communicating Relational Networks

As we mentioned before, even when the orchestration model featured by ARNs is rather expressive and versatile, we envisage two drawbacks which now can be presented in more detail.

4.1 On the binding mechanism

If we consider the binding mechanism based on LTL entailment presented in previous works, the relation between requires-point and provides-point is established in an asymmetric way whose semantics is read as trace inclusion. This asymmetry leads to undesired situations. For instance, if we return to our running example, a contract stating that the outcome of an execution is either $accept$ or $reject$ of a payment could be specified by assigning the LTL formula

$$\Diamond((-accept \vee -reject) \wedge \neg(-accept \wedge -reject))$$

to the requires-point PPS of Fig. 1(a). Likewise, one could specify a contract for the provides-point PPS of the ARN in Fig. 3(b) stating that payments are always rejected by including the

formula[1]

$$\Diamond(-reject \wedge \neg - accept)$$

It is easy to show that

$$\Diamond(-reject \wedge \neg - accept) \quad \vdash^{LTL} \quad \Diamond((-accept \vee -reject) \wedge \neg(-accept \wedge -reject))$$

by resorting to any decision procedure for LTL (see for instance, [4]). The intuition is that every state satisfying $-reject \wedge \neg - accept$ also satisfies $(-accept \vee -reject) \wedge \neg(-accept \wedge -reject)$ so if the former eventually happens, then also the latter.

The reader should note that this scenario leads us to accept a service provider that, even when it can appropriately ensure a subset of the expected outcomes, cannot guaranty that all possible outcomes will eventually be produced.

Communicating Relational Networks are defined exactly as ARNs but with definition of *Connection* based global graphs where, given a set of ports, the messages are related to the messages in the ports, and the participants are identified by the ports themselves.

Definition 9 (Connection)**.** We say that $\langle M, \mu, \Gamma\rangle$ is a *connection* on γ iff $\langle M, \mu\rangle$ is an attachment injection on γ and Γ is a global graph where the set of participants is $\{\mathsf{p}_\pi\}_{\pi\in\gamma}$ exchanging messages in M such that:

$$\mu_\pi^{-1}(\pi^-) \subseteq \bigcup_{\hat{\pi}\in\gamma\setminus\{\pi\}} \mu_{\hat{\pi}}^{-1}(\hat{\pi}^+) \qquad \text{and} \qquad \mu_\pi^{-1}(\pi^+) \subseteq \bigcup_{\hat{\pi}\in\gamma\setminus\{\pi\}} \mu_{\hat{\pi}}^{-1}(\hat{\pi}^-).$$

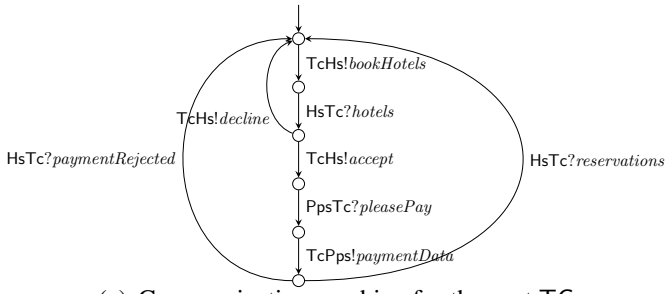
for each $\pi \in \gamma$.

Definition 10 (Communicating relational network)**.** A *communicating relational net* α is a structure $\langle X, P, C, \gamma, M, \mu, \Lambda\rangle$ consisting of:

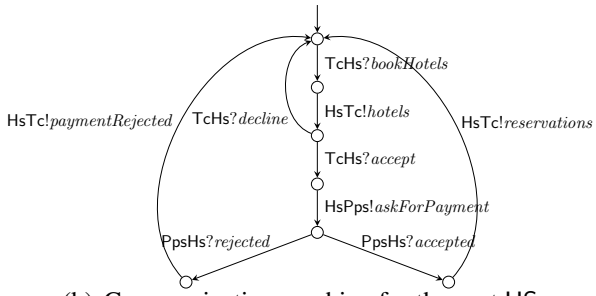
- a hypergraph $\langle X, E\rangle$, where X is a (finite) set of *points* and $E = P \cup C$ is a set of *hyperedges* (non-empty subsets of X) partitioned into *computation hyperedges* $p \in P$ and *communication hyperedges* $c \in C$ such that no adjacent hyperedges belong to the same partition, and
- three labelling functions that assign (*a*) a port M_x to each point $x \in X$, (*b*) a process $\langle \gamma_p, \Lambda_p\rangle$ to each hyperedge $p \in P$, and (*c*) a connection $\langle M_c, \mu_c, \Lambda_c\rangle$ to each hyperedge $c \in C$.

Figures 4 and 5 show the communicating machines and global graphs that can be used to redefine of the same services of the running example presented in Sec. 2, but as CRNs.

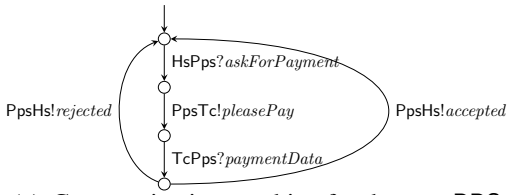
The machine in Fig. 4(a) specifies that upon reception of a *bookHotel* message from the client, HotelsService sends back a list of *hotels*; if the client accepts then computation continues, otherwise the HotelsService returns to its initial state, etc.. Also, Figs. 4(b) and (c) depict the communicating machines associated to the provides-points of services HotelsService and



(a) Communicating machine for the port TC



(b) Communicating machine for the port HS



(c) Communicating machine for the port PPS

Figure 4: Communicating machines labelling the ports TC, HS and PPS.

PaymentProcessService, respectively. From the point of view of the requires-points, the expected behaviour of the participants of a communication is declared by means of a choreography associated to communication hyperarcs. We illustrate such graphs by discussing the choreography in Fig. 5 (corresponding to the automaton in Fig. 1(c)). The graph dictates that first client and HotelsService interact to make the request and receive a list of available hotels, then the client decides whether to accept or decline the offer, etc. Global graphs are a rather convenient formalism to express distributed choices (as well as parallel computations) of work-flows. As we mentioned before, an interesting feature of global graph is that they can easily show branch/merge points of distributed choices; for instance, in the global graph of Fig. 5 branching points merge in the loop-back node underneath the initial node.

Based on Definition 10, we can define two new binding mechanisms by exploiting the "top-down" (projection) and "bottom-up" (synthesis) nature offered by choreographies.

Top-Down According to the first mechanism, provides-points are bound to require points when the projections of the global graph attached to the communication hyperarc are bisimilar to the corresponding communicating machine (exposed on the provides-points of services being evaluated for binding).

[1]In this examples we use two propositions, $accept$ and $reject$, forcing us to include in the specification their complementary behaviour, but making the formulae easier to read.

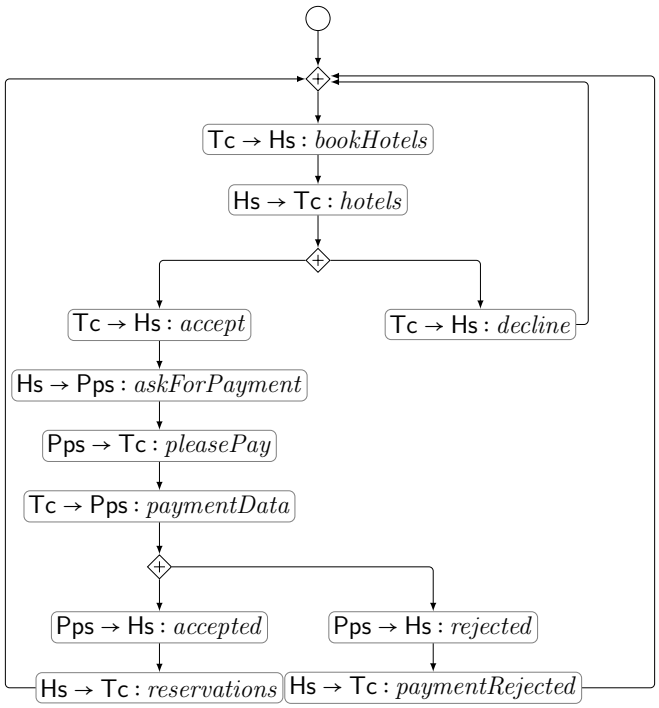


Figure 5: Global graph of the running example

Bottom-Up The second mechanism is more flexible and it is based on a recent algorithm to synthesise choreographies out of communicating machines [5]. More precisely, one checks that the choreographies synthesised from the communicating machines, associated to the provides-points of services being evaluated for binding are isomorphic to the one labelling the communication hyperarc.

For example, the projections of the global graph of Fig. 5 with respect to the components HotelsService and PaymentProcessService yields the communicating machines in Figures 4(b) and 4(c) respectively; so, when adopting the first criterion, the binding is possible and it is guaranteed to be well-behaved (e.g., there will be no deadlocks or unspecified receptions [2]). Likewise, when adopting the second criterion, the binding is possible because the synthesis of the machines in Fig. 4 yields the global graph of Fig. 5.

In this way, our approach combines choreography and orchestration by exploiting their complementary characteristics at two different levels. On the one hand, services use global graphs to declare the behaviour expected from the composition of all the parties and use communicating machines to declare their exported behaviour. On the other hand, the algorithms available on choreographies are used for checking the run-time conditions on the dynamic binding.

The resulting choreography-based semantics of binding guarantees properties of the composition of services that are stronger than those provided by the traditional binding mechanism of ARNs, and yielding a more symmetric notion of interoperability between activities and services.

4.2 Comparison of the analysis and the binding mechanism

Among the many advantages of developing software using formal tools, is the possibility of providing analysis as a means to cope with (critical) requirements. This approach generally involves the formal description of the software artefact through some kind of contract describing its behaviour. As we mentioned before, in SOC, services are described by means of their contracts associated to their provide- and require-points, playing the role that in structured programming play post- and pre-conditions of functions, respectively. From this point of view, analysing a software artefact requires:

- the verification of the computational aspects of a service with respect to its contracts, yielding a *coherence condition*, whose checking takes place at design-time, and
- the verification of the satisfaction of a property by an activity with respect to a given service repository, yielding a *quality assessment* of the software artefact, whose checking takes place also at design-time.

On the other hand, service-oriented software artefacts require the run-time checking associated to the *binding mechanism*, in order to decide whether a given service taken from the repository provides the service required by an executing activity.

Table 1 shows a comparison of the procedures that have to be implemented for checking the coherence condition of a service, the quality assessment of a service-oriented software artefact with respect to a particular repository, and for obtaining a binding mechanism for both of the approaches, the one based on ARNs, and the one based on CRNs.

5 Concluding Remarks

We propose the use of communicating relational networks as a formal model for service-oriented software design. CRNs are a variant of ARNs that harnesses the orchestration perspective underlying ARNs with a choreography viewpoint for characterising the behaviour of participants (services) over a communication channel. The condition for binding a provides-points of services to the require points of a communication channel of an activity relies on checking the compliance of the local perspective of the process, declared as communicating machines, with the global view implicit in the choreography associated to the communication channel. The binding mechanisms of ARNs (i.e., the inclusion of the set of traces of the provides-point of the service bound in the set of traces allowed by the requires-point of the activity) yields an asymmetric acceptation condition. Our approach provides a more symmetric mechanism based on rely-guarantee types of contracts.

Our framework requires the definition of a criterion to establish the coherence among the Müller automaton Λ of a process hyperedge and the communicating machines associated to its provides-points. This criterion, checked only at design time, is the bisimilarity of the communicating machine projected from Λ and the ones associated to the provides-points. The reader familiar with Mu¨ller automata should note that defining such projection is not trivial when the

automata are defined over a powerset of actions. The definition of the projection from Müller automata to communicating machine is conceptually straightforward (although technically not trivial) if the automata are defined over sets of actions (instead of powersets of them). Altough this is enough for the purposes of this paper, a better solution would be to extend communicating machines so to preserve the semantics of Müller automata even when they are defined on powersets of actions. This is however more challenging (as the reader familiar with Muller automata would recognise) and it is left as a future line of research.

We strived here for simplicity suggesting trivial acceptance conditions. For instance, in the "bottom-up" binding mechanism we required that the exposed global graph coincides (up to isomorphism) to the synthesised one. In general, one could extend our work with milder conditions using more sophisticated relations between choreographies. For instance, one could require that the interactions of the synthesised graph can be simulated by the ones of the declared global graph.

We also envisage benefits that the orchestration model of ARNs could bring into the choreography model we use (similarly to what suggested in [1]). In particular, we argue that the 'incremental binding' naturally featured in the ARN model could be integrated with the choreography model of global graphs and communicating machines. This would however require the modifications of algorithms based on choreography to allow incremental synthesis of choreographies.

Bibliography

[1] D. Basile, P. Degano, G. L. Ferrari, and E. Tuosto. From orchestration to choreography through contract automata. In *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, pages 67–85, 2014.

[2] D. Brand and P. Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, Apr. 1983.

[3] P. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, pages 194–213, 2012.

[4] Y. Kesten, Z. Manna, and H. M. A. Pnueli. A decision algorithm for full propositional temporal logic. In *CAV*, pages 97–109, 1993.

[5] J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In *Principles of Programming Languages (PoPL)*, 2015. To appear.

[6] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[7] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Comput. Sci.*, 13(1):45–60, 1981.

[8] I. Ţuţu and J. L. Fiadeiro. A logic-programming semantics of services. In *CALCO*, pages 299–313, 2013.

MEALS Partner Abbreviations

SAU: Saarland University, D

RWT: RWTH Aachen University, D

TUD: Technische Universität Dresden, D

INR: Institut National de Recherche en Informatique et en Automatique, FR

IMP: Imperial College of Science, Technology and Medicine, UK

ULEIC: University of Leicester, UK

TUE: Technische Universiteit Eindhoven, NL

UNC: Universidad Nacional de Córdoba, AR

UBA: Universidad de Buenos Aires, AR

UNR: Universidad Nacional de Río Cuarto, AR

ITBA: Instituto Técnológico Buenos Aires, AR

Formalisation	Coherence Condition	Quality assessment	Binding Mechanism
ARNs	$\{\Delta_{\Lambda_p} \vDash^{\mathsf{LTL}} \Gamma_\pi\}_{\pi \in \gamma_p}$ where $p \in P$, $\langle \gamma_p, \Lambda_p \rangle$ is a process, Δ_{Λ_p} the set of traces of the Müller automaton Λ_p and Γ_π is the LTL contract associated to port π.	$\prod_{m \in P \cup C} \Lambda_m$	$\Gamma_\pi \vdash^{\mathsf{LTL}} \Gamma_\rho$ where π is a provides point of a service, ρ is a requires point of an activity, and Γ_π and Γ_ρ their LTL contract respectively.
CRNs	$\{\Lambda\|_{p_\pi} \approx \mathcal{A}_\pi\}_{\pi \in \gamma_p}$ where $\Lambda\|_{p_\pi}$ is the projection of Müller automaton Λ over the alphabet of port π, $\mathcal{A}_\pi$ is the communication machine labelling port π and $\approx$ denotes bisimilarity.	$\prod_{m \in P} \Lambda_m$	**Top-Down**: $G\|_\rho \approx \mathcal{A}_\pi$ where π is a provides point of a service, ρ is a requires point of an activity, $G_c\|_{p_\rho}$ is the projection of the global graph G_c over the language of the port ρ, $\mathcal{A}_\pi$ is the communication machine labelling port π and $\approx$ denotes bisimilarity. **Bottom-Up**: $S(\{\mathcal{A}_\pi\}_{\pi \in \Pi}) \equiv G_c$ where Π is the set of provides-points of the services to be bound, G_c is the global graph associated to $c \in C$, $S(\bullet)$ is the algorithm for synthesising choreographies from communication machines [5] and $\equiv$ denotes isomorphism.

Table 1: Comparison of the procedures for the approaches based in ARNs and CRNs