Tesis Doctoral

Aspectos formales de un modelo de ejecución orientada a servicios

Vissani, Ignacio

2018

UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Aspectos formales de un modelo de ejecución orienteada a servicios

Tesis presentada para optar al título de Doctor de la Universidad de Buenos Aires en el área Ciencias de la Computación

Lic. Ignacio Vissani

Director de tesis: Dr. Carlos Gustavo Lopez Pombo
Consejero de estudios: Dr. Hernán Melgratti

Lugar de Trabajo: Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires

Fecha de defensa: 30 de Julio de 2018

Formal aspects of a service oriented execution model

Abstract

Distributed software resulting from emerging paradigms such as service-oriented computing (SOC), Cloud/Fog computing and the Internet of Things are transforming the world of software systems in order to support applications able to respond and adapt to the changes of their execution environment, giving impulse to what is called the API's economy. The underlying idea of the API's economy is that it is possible to construct software artifacts, usually by composing services previously registered in repositories and provided by third parties. This envisages a generation of applications running over globally available computational resources and communication infrastructure, which, at run-time, are dynamically and transparently reconfigured by the intervention of a dedicated middleware, subject to the negotiation of a Service Level Agreement – SLA [25]. Under this paradigm software services are accessed by their API.

Many of the aspects related to providing formal foundations and tool support for these new paradigms have been tackled in the last years [66], yet some remain open. In particular the ability to provide a working infrastructure capable of realizing full automatic service discovery and binding is still an open challenge [33, 59].

In this thesis we contribute to this goal by focusing on two facets of the problem: (1) the provision of a formal setting capable of capturing the particularities of these paradigms, being the most relevant the fact that one cannot know at design time which service (if some) will satisfy a requirement and (2) the necessity of being capable of determining at runtime whether there exists a service, in a given repository, capable of satisfying a given requirement.

As contribution to aspect (1) we provide an operational semantics for Asynchronous Relational Networks [24] that takes into account both internal and reconfiguration actions. We also extend this semantics with the ability to capture reconfigurations that are not incremental on the structure in order to support unreliability derived from the execution infrastructure.

As contribution to aspect (2) we explore the usage of CFSMs [9] in our model to express both requirements and provision contracts. In this way we resort to the mechanism given in [38] to provide an automatic interoperability check for services. We also extended CFSMs in order to equip them with data and assume/guarantee conditions in the form of first order formulae

over this data. In this way we transform CFSMs in a suitable mechanism for expressing and checking (restricted) functional contracts.

Keywords: SOC, SOA, formal semantics, choreographies, orchestration, formal methods.

Aspectos formales de un modelo de ejecución orienteada a servicios

Resumen

El software distribuido que resulta de los nuevos paradigmas que están emergiendo, tales como el de computación orientada a servicios (SOC), computación en la nube e internet de las cosas, está transformando el mundo de los sistemas de software de modo de dar soporte a aplicaciones capaces de responder y adaptarse a los cambios en su entorno de ejecución, dando impulso a lo que se conoce como la economía de las APIs. La idea que subyace a la economía de las APIs es que es posible construir piezas de software a partir de componer servicios previamente registrados en repositorios y provistos por terceros. Esto promete una generación de aplicaciones ejecutando sobre recursos computacionales y una infraestructura de comunicación globalmente distribuidos que, en tiempo de ejecución son reconfiguradas dinámica y transparentemente mediante la intervención de un middleware dedicado. Esta reconfiguración está sujeta a la negociación de un acuerdo de nivel de servicio - SLA [25]. En este paradigma los servicios de software son accedidos a través de sus APIs.

Muchos de los aspectos relacionados con la provisión de fundamentos formales y herramientas para dar soporte a estos nuevos paradigmos han sido resueltos en los últimos años [66], sin embargo algunos permanecen abiertos. En particular la habilidad para proveer una infraestructura capaz de llevar a cabo los procesos de *discovery* y *bindig* de manera completamente automática es aún un desafío abierto [33, 59].

En esta tesis contribuimos a este objetivo haciendo foco en dos aspectos del problema: (1) la provisión de elementos formales capaces de capturar las particularidades de estos paradigmas siendo la más relevante el hecho de que no es posible saber en tiempo de diseño qué servicio, si es que alguno, podrá satisfacer un determinado requerimiento y (2) la necesidad de ser capaces de determinar en tiempo de ejecución si existe un servicio particular, en un repositorio dado, capaz de satisfacer un determinado requerimiento.

Como contribución al punto (1) proporcionamos una semántica operacional para las Asynchronous Relational Networks [24] que captura tanto las transiciones internas como las acciones de reconfiguración que ocurren durante la ejecución de un servicio. También extendimos esta semántica con la habilidad para capturar reconfiguraciones no incrementales con respecto

a la estructura, de modo de dar soporte a la falta de confiabilidad derivada de la infraestructura de ejecución.

Como contribución al punto (2) exploramos el uso de CFSMs [9] en nuestro modelo para expresar tanto requerimientos como contratos de provisión de servicio. De esta manera recurrimos al mecanismo dado en [38] para proveer un chequedo de interoperabilidad automático para servicios. También extendimos las CFSMs y las equipamos con datos y condiciones de tipo asunción/garantía en la forma de fórmulas de primer orden sobre esos datos. De este modo transformamos a las CFSMs en un mecanismo apropiado para expresar y chequear contratos funcionales restringidos.

Palabras clave: SOC, SOA, semántica formal, coreografías, orquestación, métodos formales.

Agradecimientos

Escribir los agradecimientos me obliga a repensar el proceso más allá del resultado. Quienquiera que haya escrito alguna vez una tesis sabe que es un trabajo difícil, largo, arduo, lleno de sinsabores, dudas, miedos. O al menos así lo fue para mí. Me es imposible imaginarme sentado en este lugar, en este momento, escribiendo estos agradecimientos, sin recordar cada uno de los empujoncitos hacia adelante, que en algunos casos tuvieron bastante de empujón y bastante poco de *citos*, que me dieron múltiples personas durante todo este tiempo. Valgan entonces estas palabras a modo de agradecimiento para todas las personas que de una u otra manera contribuyeron a que esta tesis sea una realidad.

A Charlie, por haber sido el director más generoso que alguien pudiera pedir. Generoso con su tiempo, con sus recursos. Por haberme dado durante todo este tiempo todas las posibilidades que estuvieron a su alcance. Por nunca haber dejado de empujar hacia adelante y por haber resistido a mi eterno pesimismo. Pero también gracias por haber sido un amigo todo este tiempo. Por la infinidad de anécdotas que quedarán para el buen recuerdo. Por haber estado siempre al pie del cañón. Por seguir siendo un amigo.

A Emilio Tuosto por sus aportes y su tiempo. Por su dedicación desinteresada. Por haberme abierto las puertas de su casa. Por haber aportado su visión crítica y optimista en momentos de desesperación. A Hernán Melgratti por ponderar siempre lo humano por encima de todo y por su tiempo.

A mis amigos Damián y Gutes, por seguir siéndolo a pesar de las distancias, temporales y geográficas. A Nico y Mati por al apoyo incondicional, por la amistad, por los mates y las birras. Por haber hecho de la facultad un lugar al que quería ir (cada tanto).

A mi familia tanto de sangre como política. Por haber soportado a este ente. En especial a mi papá y a mi mamá por haberme dejado ser.

A Ali por ser el pilar fundamental de mi vida. Por haberme sacado del pozo y bajado de las nubes (sí, ambas). Por tener siempre el consejo adecuado. Por bancarse el cansancio un día más y otro día más, y otro más. Por hacer todo. Por despertarse a dormir a Maite, todos los días, durante tanto tiempo. Por volver corriendo para que yo no pierda tiempo. Por quedarse en casa para que yo no pierda tiempo. Por cocinar, lavar, ordernar, para que yo no pierda tiempo. Por cuidar a Maite. Y sobre todo, por quererme igual.

A Maite, por ser mi refugio. Mi lugar feliz. Por tener tu sonrisa y tu ternura siempre dispuesta a alegrar a tu papá loco.

Contents

Chapter 1

Introduction

In the context of global ubiquitous computing, the structure of software systems is becoming more and more dynamic as applications need to be able to respond and adapt to changes in the environment in which they operate. For instance, the new paradigm of *Service-Oriented Computing* (SOC) supports a new generation of software applications that run over globally available computational and network infrastructures where they can procure services on the fly (subject to a negotiation of *Service Level Agreements*, or SLAs for short) and bind to them so that, collectively, they can fulfil given business goals [25]. There is no control as to the nature of the components that an application can bind to. In particular, development no longer takes place in a top-down process in which subsystems are developed and integrated by skilled engineers: in SOC, discovery and binding are performed by middleware.

We may say that the paradigm has been in use for more than a decade now. Yet we should point out that the term SOC is still subject to interpretation and discussion with regards to what exactly their goals are. As an example, the industry has been concerned to address SOC promises by providing tools that cope with interoperability in the form of abstracting services' descriptions from implementation languages. To this end languages and tools like WSDL [64] and SOAP [63] have been developed. Also middlewares capable of locating and delivering messages back and forth from a given service inside a network without the need for hardcoding the location of such services have also been provided in the form of service buses [31, 54]. Besides, languages to describe composition of services in terms of orchestration, such as WS-BPEL [52], or choreographies such as WS-CDL [65] have also been developed, although WS-CDL development has been abandoned

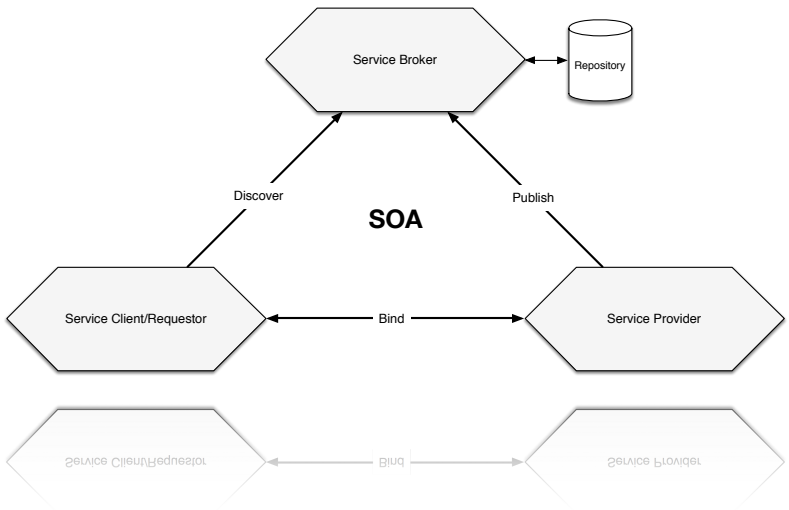


Figure 1.1: Elements of a SOA

in 2009.

In this view, SOC is regarded mainly as a **design or engineering principle** that can contribute to reduce complexity of software development in big organizations by breaking complex monolithic systems into a composition of several simpler and self-contained, globally distributed services. One could say that SOC gathers ideas from component-based design and from programming-in-the-large [20]. This view gave birth to Service Oriented Architecture (SOA for short).

We can also observe that industry contributed with languages for the development of global applications but, in general, they are based into purely informal specifications and the semantics of many of their constructions is still ambiguous [53]. This results in the same program being executed diversely depending on the particular implementation of the language (as an example see [39]).

The subject of SOC has provoked an important amount of work from academia also. In particular the necessity to provide formal foundations for the paradigm that enable a systematic approach to service design and development and formalisms and tools to support description and verification of services and their composition had been the focus of the community for a long time. Of these efforts we may distinguish the SENSORIA [66] project, a joint European project that gave birth to tens of the most prominent pub-

lications in the field. Another topic that was given attention is the subject of providing *semantics* to service descriptions by means of arranging the concepts in services' interfaces into ontologies.

> In particular, the innovation is to make explicit the business or user-domain semantics of services, so far implied by the syntax of their descriptions. Employing syntactic descriptions either makes service semantics ambiguous or calls for a syntax-level agreement between service developers/providers and service users, which is too restrictive for the inherent loosely coupled character of SOC.[...]Ontology languages support formal description and machine reasoning upon ontologies; the Web Ontology Language (OWL) is the standard established by W3C. [33]

While in this thesis we do not address the topic of ontologies, this quote that we share suggests that **there is more to SOC** than some architectural and design principles on steroids. It is our opinion that one of the key aspects of the paradigm resides in that the discovery and binding of services should be performed at runtime in a non programmed (i.e. non hardcoded) way. We argue that in this sense the paradigm has not been realized to its full extent yet. To us it still remains as an open problem the ability to implement a service broker capable of fully automatically resolve discovery and binding of services [59] subject to the verification of 1. Semantic compatibility (i.e. ontologies), 2. communicational interoperability (i.e. protocol implementation) 3. functional specifications and, 4. service level agreement. In our view the paramount realization of this paradigm would permit one to write some sort of contract specifying requirements in terms of items 1–3. This requirements would then be provided to the service broker. The service broker would then query the repository or repositories to select viable candidates (i.e. services capable of fullfiling the requirements) that would in turn be subject to a negotiation of service level.

To achieve that objective one needs to have contracts expressive enough to represent the exposed behaviour of a service as well as a computationally inexpensive procedure (or a way of amortizing it) to determine contracts satisfaction. This two goals are, in general, contradictory. One can rapidly fall into undecideable theories, or even when restraining to decideable theories, most of the procedures to stablish satisfaction of functional contracts are computationally hard. Several efforts have been made in this topic and much of the progress has been achieved by the use of derivated forms of the π-calculus [47, 48]. Examples of these efforts can be found in [7, 13].

Reasonably, most of the work in this area concentrates on communicational aspects of services, on the one hand because this is the aim and goal of π-calculus, on the other hand because it is argued that the level of abstraction of communication protocols is close to what is expected as a service interface. Communicating Finite State Machines [9] and global graphs [19] provide graphical automata-based descriptions of a communication protocol. Graphical languages, from our perspective, are interesting in their own right as they have a better chance to be adopted by industry.

A recent result [38] make Communicating Machines and global graphs more interesting as the authors provide a sufficient condition, and a decideable procedure to stablish it, that characterizes when a set of communicating processes can yield a choreography (i.e. a global graph) which means they can interact free of communication errors. Even though this procedure is computationally expensive in the general case, it has proven to be very efficient in a number of cases. In this thesis we try to enrich communicating machines while keeping the results in [38] in order to use them to express richer contracts for services.

As we stated before, SOC brings to the table new and profound challenges at the level of systems' structure. This is a result of their ability to discover and bind dynamically to other systems at runtime. Therefore the structure of these systems is intrinsically dynamic. Giving formal semantics to these kind of systems is, in our opinion, a fundamental issue that needs to be addressed in order to let us understand their behaviour and to highlight the fundamental differences between this paradigm and others, specially because there is still some disagreement on what is really new about this type of software. On top of that, formal semantics are, in general, a prerequisite *sine qua non* in order to develop tools and techniques that allow developers and designers to build dependable systems. In the particular scenario of services the dynamic changes suffered at the level of the structure of these systems needs to be a central part of this semantics as the properties and guarantess that we need to establish depend greatly on this subject. In the end, an execution model with formal semantics that integrates dynamic reconfigurations subject to discovery and binding is required. Towards this end we extend the work done in [23, 24, 62] by providing an operational semantics that integrates reconfiguration actions into the framework of Asynchronous Relational Networks.

1.1 Structure of the thesis

The structure of this thesis is as follows:

- In Chapter 2 we provide the preliminary definitions related to ARNs. In Section 2.1 we give a brief introduction of catefory theory definitions and results that are used in the definition of ARNs in which we base our work. In Section 2.2 we give the definitions of Asynchronous Relational Networks based on the presentation given in [62].

- In Chapter 3 we extend the work in [62] to provide an operational semantics for ARNs. This semantics integrates in a coherent yet abstract way reconfigurations into a trace semantics. In Section 3.1 we provide semantics considering only monotonic (i.e. incremental) reconfigurations. To demonstrate how this semantics can be used to reason about these kind of systems in Section 3.1.5 we show how to express LTL Formulae and check for their satisfaction. Afterwards in Section 3.2 we modify ARNs in order to also consider non monotonic reconfigurations (i.e. remotions). Again in Section 3.2.5 we depict how LTL formulae reasoning can be adapted to the latter scenario.

- In Chapter 4 we present our approach to service binding based on combining communicating finite state machines together with ARNs. In Section 4.2 we give the preliminary definitions on communicating finite state machines and global graphs. In Section 4.4 we present a prooposal to integrate the framework of ARNs with the choreography model provided by CFSMs and discuss the points that need to be addressed. In Section 4.6 we present a variant of communicating machines capable of handling values associated to the messages and we discuss their limitation. In Section 4.7 and Section 4.8 we discuss two other ways of considering data associated to messages. Finally in Section 4.9 we introduce a new kind of automata to reason about compositions of communicating processes.

- In Chapter 5 we point out general conclusions and further lines of research.

1.2 Resumen

En el contexto de la computación global uibicua la estructura de los sistemas de software se está volviendo cada vez más dinámica dado que las

aplicaciones deben responder y adaptarse a los cambios que se producen en el entorno en el que operan. Por ejemplo, el paradigma de *software orientado a servicios* (SOC) da soporte a una nueva generación de aplicaciones que ejecutan sobre recursos computacionales y de red globalmente distribuidos a través de los cuales pueden obtener servicios sobre la marcha (sujeto a la negociación de *acuerdos de nivel de servicio*) de manera que, colectivamente, pueden cumplir un determinado objetivo de negocios [25]. En este paradigma no hay control sobre la naturaleza de los componentes a los que una aplicación puede vincularse. En particular, el desarrollo ya no ocurre en un proceso de arriba hacia abajo en el que los subsistemas son desarrollados e integrados por ingenieros altamente capacitados: en SOC el descubrimiento y la vinculación son llevadas a cabo por una capa intermedia.

Las *Communicating Finite State Machines* [9] y los *global graphs* [19] proveen una descripción gráfica de protocolos de comunicación basada en autómatas. Desde nuestra perspectiva los lenguajes gráficos son interesantes por derecho propio dado que tienen una probabilidad más alta de ser adoptados por la industria. Un resultado reciente [38] hace a las *Communicating Machines* y a los *global graphs* más interesantes ya que los autores proporcionan una condición suficiente, junto con un procedimiento decidible para determinarla, que permite caracterizar cuándo un conjunto de procesos dan lugar a una coreografía, lo que significa que pueden interactuar sin errores de comunicación. Si bien este procedimiento es computacionalmente caro en el caso general, el mismo ha mostrado ser muy eficiente en una cantidad importante de casos. En esta tesis intentmos enriquecer las *communicating machines* manteniendo los resultados obtenidos en [38] de modo de poder utilizarlas para expresar contratos de servicio más ricos.

SOC pone sobre la mesa nuevos y profundos desafíos en el plano de la estructura de los sistemas como consecuencia de la capacidad de descrubir y vincularse dinámicamente a otros sistemas en tiempo de ejecución. Por lo tanto la estructura de estos sistemas es intrínsecamente dinámica. Proporcionar una semántica formal para este tipo de sistemas es, en nuetras opinión, una cuestión fundamental que debe ser resuelta para permitirnos entender el comportamiento de este tipo de software y para poder remarcar las diferencias fundamentales entre este paradigma y otros, sobre todo porque existe aún cierta discusión al respecto de cuáles son los aspectos verdaderamente innovadores de SOC. Por sobre eso, tener una semántica formal es, en general, un prerequisito *sine qua non* para poder desarrollar herramientas y técnicas que permitar a los desarrolladores y diseñadores

construir sistemas confiables. En el escenario particular de SOC los cambios dinámicos que ocurren en el plano de la estructura de estos sistemas debe formar parte central de la semántica ya que las propiedades y garantías que necesitamos establecer dependen enormemente de ésto. Al fin de cuentas se necesita un modelo de ejecución con semántica formal que incorpore la reconfiguración dinámica. Con este objetivo extenedmos el trabajo realizado en [23, 24, 62] y proporcionamos una semántica operacional que internaliza las acciones de reconfiguración en el *framework* de las *Asynchronous Relational Networks*.

Chapter 2

Formal Introduction

In the present chapter we introduce the definitions and results that we use throughout the rest of this thesis. In the first place we introduce general notions on category theory that we use as formal background. After that Asynchronous Relational Networks, a computational model for services, are presented.

2.1 Category theory

In this section we introduce those concepts coming from category theory we will use throughout the rest of this work. This chapter does not pretend to be an exhaustive presentation of category theory so we point the interested reader to [22] for a gentle introduction to category theory for computer scientists and to [44] for an introduction to category theory for mathematicians.

From here on, we assume the reader has a nodding acquaintance with basic concepts from category theory such as the notions of category, functor, natural transformation and colimit. This section only summarizes these definitions in order to fix the notation.

Definition 1 (Graph)**.** *A* graph *is a structure* $\langle G_0, G_1 \rangle$ *where* G_0 *and* G_1 *are collections of nodes and arrows respectively.* $G_1 \subseteq G_0 \times G_0$ *and if* $f \in G_1$ *is an arrow from* X *to* Y*, it will be denoted as* $f : X \to Y$ *or* $X \xrightarrow{f} Y$*.*

Definition 2 (Category)**.** *A* category *is a structure* $\langle G, \circ, id \rangle$ *where* $G = \langle G_0, G_1 \rangle$ *is a graph,* $\circ : G_1 \times G_1 \to G_1$ *is the composition of elements in* G_1 *(if* $f : X \to Y, g : Y \to Z \in G_1$*, the composition of* f *and* g *is denoted as*

$f \circ g : X \to Z$) and $id : G_0 \to G_1$ the identity map (if $X \in G_0$, $id_X : X \to X$ is the identity arrow for X).

If C is a category, by graph(C) we denote the graph of C.

Elements in a category are called objects *and arrows are referred to as* morphisms.

Definition 3 (The category $\mathbb{SET}$)**.** $\mathbb{SET} = \langle G, \circ, id \rangle$ *such that:*

- $G = \langle G_0, G_1 \rangle$*, where G_0 is the collection of all sets and G_1 is the collection of all total functions between sets,*
- *if $S, S', S'' \in G_0$ and $f : S \to S', f' : S' \to S'' \in G_1$, then $f \circ f' : S \to S''$ is defined as $f \circ f'(s) = f'(f(s))$ for all $s \in S$,*
- *if $S \in G_0$, then $id_S : S \to S$ is the identity on S.*

Instead of following the formal definition of a category, we will usually define them just by declaring what are their objects and morphisms, and by defining their composition and identities, omiting any mention to its graph. Thus, we will present a category as a structure $\langle \mathcal{O}, \mathcal{A} \rangle$ where $\mathcal{O}$ is the collection of objects and $\mathcal{A}$ is the collection of morphisms together with an appropriate definition of $id_X \in \mathcal{A}$ for each object $X \in \mathcal{O}$ and $\circ : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$. Given a category C, the collection of objects of C, will be denoted $|\mathsf{C}|$.

Sometimes categories have distinguished objects with particular behaviors which can be identified by the way they are related to other objects in the category. Among these we single out the *initial* and *terminal* objects.

Definition 4 (Initial object)**.** *Let C be a category and $x \in |\mathsf{C}|$, then x is* initial *if and only if for all $y \in |\mathsf{C}|$, there exists a unique morphism $f : x \to y$.*

Analogously, x is terminal *if and only if for all $y \in |\mathsf{C}|$, there exists a unique morphism $f : y \to x$.*

These distinguished elements have some interesting properties. The next proposition shows a very usefull property of initial objects.

Proposition 1. *([22, §4.1.2]) Any two initial objetcs are isomorphic.*

Proof. Any objects isomorphic to an initial object are also initial. □

Most of the reasoning in category theory can be carried out by observing properties in *diagrams*.

Definition 5 (Diagram)**.** *Let C be a category and I a graph. A* diagram *with shape I in C is a graph homomorphism $\delta = \langle \delta_0, \delta_1 \rangle : I \to graph(\mathsf{C})$.*

Definition 6 (Commutative diagrams)**.** *Let* C *be a category and* $I = \langle G_0, G_1 \rangle$ *a graph. A diagram* $\delta = \langle \delta_0, \delta_1 \rangle : I \to graph(\mathsf{C})$ *commutes if and only if for every* $X, Y \in G_0$ *and* $f_0 \circ \cdots \circ f_i : X \to Y, g_0 \circ \cdots \circ g_j : X \to Y \in G_1$*,*

$$\delta_1(f_0) \circ \cdots \circ \delta_1(f_i) = \delta_1(g_0) \circ \cdots \circ \delta_1(g_j) \ .$$

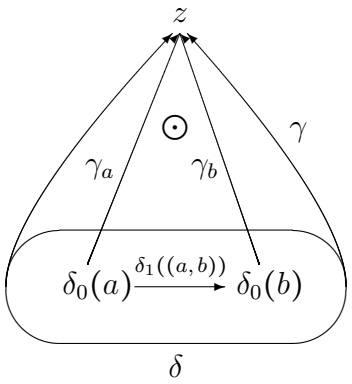
Diagrams will be usually presented by their graphical representation resorting to their image on the category instead of using the previous definitions. Commutative diagrams will be distinguished by decorating them with the symbol $\odot$ inside.

Given a diagram, it is possible to identify the collective behavior of the objects it involves. To capture this collective behavior we introduce the notion of *(co)cones* and *(co)limits*.

Definition 7 (Cocone)**.** *Let* C *be a category and* $I = \langle G_0, G_1 \rangle$ *be a graph. Let* $\delta = \langle \delta_0, \delta_1 \rangle : I \to graph(\mathsf{C})$*. A* cocone *with base* δ *is an object* $z \in \mathsf{C}$ *(the vertex of the cocone), together with a family of morphisms* $\{\gamma_a : \delta_0(a) \to z\}_{a \in \delta_0(G_0)}$ *(the edges of the cocone), usually denoted* $\gamma : \delta \to z$*.*

A cocone is commutative *if and only if for any pair of vertexes* $a, b \in G_0$ *such that* $\langle a, b \rangle \in G_1$*,* $\delta_1(\langle a, b \rangle) \circ \gamma_b = \gamma_a$*.*

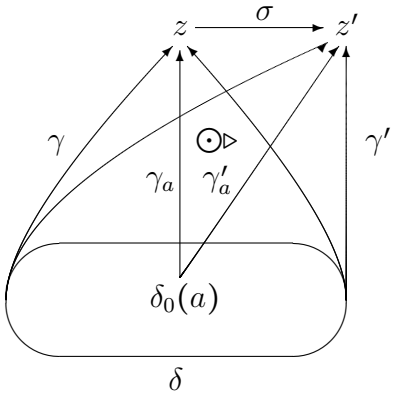
The concept of cone *is the dual notion of cocone (i.e. the equivalent definition but reversing the arrows).*



Definition 8 (Colimit)**.** *Let* C *be a category and* $I = \langle G_0, G_1 \rangle$ *be a graph. Let* $\delta = \langle \delta_0, \delta_1 \rangle : I \to graph(\mathsf{C})$*. A* colimit *is a commutative cocone* $\gamma : \delta \to z$ *such that for every commutative cocone* $\gamma' : \delta \to z'$*, there is a unique morphism* $\sigma : z \to z'$ *satisfying* $\gamma \circ \sigma = \gamma'$ *(i.e. for all* $a \in G_0$*,* $\gamma_a \circ \sigma = \gamma'_a$*).*

The concept of limit *is the dual notion of colimit (i.e. the equivalent definition but reversing the arrows).*

As in the case of diagrams, (co)cones and (co)limits can be presented by its graphical representation by resorting to its image on the category and decorating colimits and limits with $\odot\triangleright$ and $\odot\triangleleft$ respectively.



Definition 9 (Cocompleteness)**.** *A category is (finitely) cocomplete if and only if all (finite) diagrams have colimits.*

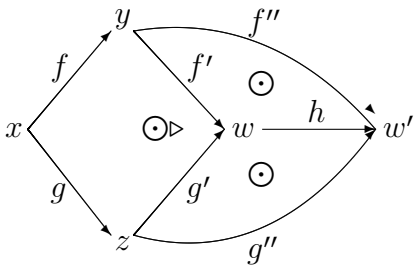
A category is (finitely) complete if and only if all (finite) diagrams have limits.

Definition 10 (Pushouts)**.** *Let* C *be a category and* $f : x \to y$*,* $g : x \to z$ *morphisms in* C*, then a* pushout *of* f *and* g *consists of* $f' : y \to w$*,* $g' : z \to w$ *morphisms in* C *such that:*

- $f \circ f' = g \circ g'$*, and*
- *for any other* $f'' : y \to w'$*,* $g'' : z \to w'$ *morphisms in* C *such that* $f \circ f'' = g \circ g''$*, there exists a unique* $h : w \to w'$ *morphism in* C *such that* $f' \circ h = f''$ *and* $g' \circ h = g''$*.*

Pullback *are defined analogously to pushouts but considering the arrows in the opposite direction.*

When pushouts and pullbacks are presented in a diagramatic way they are decorated with $\odot\triangleright$ and $\odot\triangleleft$ respectively.



There are several results on (co)completeness of categories, one of them is the following proposition.

Proposition 2. *([22, §4.4.7])*
A category is cocomplete if and only if it has initial objects and pushouts of all pairs of morphisms with common source.

A category is complete if and only if it has final objects and pullbacks of all pairs of morphisms with common source. □

Categories can not only be constructed by giving its graph, composition operation and identity map, new categories can be built from existing ones by using some elementary operations. The advantage of constructing new categories from existing ones is that the former "inherit" properties from the latter.

We will now present some (those we will use throwghout the rest of the work) of these elementary constructions.

Definition 11 (Opposite, or dual, category)**.** *Let* $\mathsf{C} = \langle \mathcal{O}, \mathcal{A} \rangle$ *be a category. Then* C^{op} *(the* opposite category *of* C*) is defined as* $\langle \mathcal{O}, \mathcal{A}^{\mathsf{op}} \rangle$ *where the morphisms in* $\mathcal{A}^{\mathsf{op}}$ *are the morphisms of* $\mathcal{A}$ *reverted (i.e.* $f^{\mathsf{op}} : Y \to X \in \mathcal{A}^{\mathsf{op}}$ *if and only if* $f : X \to Y \in \mathcal{A}$*).*

The previous definitions reflects that the direction of morphisms in a category has no essential significance because structural properties of the category are, in some sense, preserved in its opposite category; but the interpretation of the morphisms have a "natural" direction which helps the understanding of the category. Of course the interpretation of morphisms differ from one category to its opposite. If we recall on the definition of $\mathbb{SET}$, morphisms in $\mathbb{SET}^{\mathsf{op}}$, as they go in the opposite direction, can be regarded as the corresponding partial surjective relations.

An example of a property which can be proved by using this universal construction is the following proposition.

Proposition 3. *Let* C *be a category and* $x \in |\mathsf{C}|$*, then* x *is terminal if and only if* x *is initial in* C^{op}*.*

Proof. The proof follows trivially by Defs. 2, 11 and 4. □

Definition 12 (Product category)**.** *Let* $\mathsf{C} = \langle \mathcal{O}_{\mathsf{C}}, \mathcal{A}_{\mathsf{C}} \rangle$ *and* $\mathsf{D} = \langle \mathcal{O}_{\mathsf{D}}, \mathcal{A}_{\mathsf{D}} \rangle$ *be categories. Then* $\mathsf{C} \times \mathsf{D}$ *(the* product category *of* C *and* D*) is defined as* $\langle \mathcal{O}_{\mathsf{C}} \times \mathcal{O}_{\mathsf{D}}, \mathcal{A} \rangle$ *where the morphisms in* $\mathcal{A} \subseteq \mathcal{A}_{\mathsf{C}} \times \mathcal{A}_{\mathsf{D}}$ *are such that* $f_{\mathsf{C}} : X_{\mathsf{C}} \to Y_{\mathsf{C}} \in \mathcal{A}_{\mathsf{C}}$ *and* $f_{\mathsf{D}} : X_{\mathsf{D}} \to Y_{\mathsf{D}} \in \mathcal{A}_{\mathsf{D}}$ *if and only if* $\langle f_{\mathsf{C}}, f_{\mathsf{D}} \rangle : \langle X_{\mathsf{C}}, X_{\mathsf{D}} \rangle \to \langle Y_{\mathsf{C}}, Y_{\mathsf{D}} \rangle \in \mathcal{A}$*.*

There are many of these elementary operations which permit the construction of new categories from existing ones but will not be used in this work. The interested reader is pointed to [44] for a complete presentation of these operations. Those that are most commonly used in computer science can be found in [22].

Another way to create a new category is to consider it as a *subcategory* of another one. This means that, given a category, it is possible to build a new one by removing some of its objects and morphisms.

Definition 13 (Subcategory)**.** *Let* $\mathsf{C} = \langle \mathcal{O}_{\mathsf{C}}, \mathcal{A}_{\mathsf{C}} \rangle$ *and* $\mathsf{D} = \langle \mathcal{O}_{\mathsf{D}}, \mathcal{A}_{\mathsf{D}} \rangle$ *be categories. Then* D *is a* subcategory *of* C *if and only if:*

- $\mathcal{O}_{\mathsf{D}} \subseteq \mathcal{O}_{\mathsf{C}}$*,*
- $\mathcal{A}_{\mathsf{D}} \subseteq \mathcal{A}_{\mathsf{C}}$ *such that:*
  - *if* $X \in \mathcal{O}_{\mathsf{D}}$*, then* $id_{\mathsf{D}X} = id_{\mathsf{C}X}$*,*
  - *if* $f : X \to Y, g : Y \to Z \in \mathcal{A}_{\mathsf{D}}$*, then* $f \circ_{\mathsf{D}} g = f \circ_{\mathsf{C}} g$*.*

Whenever D *is a subcategory of* C*, it is denoted by* $\mathsf{D} \hookrightarrow \mathsf{C}$*.*

There are many ways in which categories relate to each other. The most common way categories relate is through *functors*.

Definition 14 (Functor)**.** *Let* $\mathsf{C} = \langle \mathcal{O}_{\mathsf{C}}, \mathcal{A}_{\mathsf{C}} \rangle$ *and* $\mathsf{D} = \langle \mathcal{O}_{\mathsf{D}}, \mathcal{A}_{\mathsf{D}} \rangle$ *be categories. Then* $\delta : \mathsf{C} \to \mathsf{D}$ *is a* functor *if and only if:*

- *if* $X \in \mathcal{O}_{\mathsf{C}}$*, then* $\delta(X) \in \mathcal{O}_{\mathsf{D}}$*,*
- *if* $f : X \to Y \in \mathcal{A}_{\mathsf{C}}$*, then* $\delta(f) : \delta(X) \to \delta(Y) \in \mathcal{A}_{\mathsf{D}}$*, such that:*
  - *if* $X \in \mathcal{O}_{\mathsf{C}}$*,* $\delta(id^{\mathsf{C}}{}_X) = id^{\mathsf{D}}{}_{\delta(X)}$*,*
  - *if* $f : X \to Y, g : Y \to Z \in \mathcal{A}_{\mathsf{C}}$*, then* $\delta(f \circ_{\mathsf{C}} g) = \delta(f) \circ_{\mathsf{D}} \delta(g)$*.*

It is now possible to define the category Cat, whose objects and morphisms are categories and functors respectively. This requires to prove that the identity law and composition law are preserved by functors which trivially follows from Def. 14.

2.2 Asynchronous Relational Networks

Asynchronous Relational Networks (ARNs) were presented by Fiadeiro and Lopes in [24] with the aim of formalising the elements of an interface theory for service-oriented software designs. ARNs are a formal orchestration model based on hypergraphs whose hyperedges[1] are interpreted either as processes or as communication channels. The nodes (or points) that are only adjacent to process hyperedges are called *provides-points*, while those adjacent only to communication hyperedges are called *requires-points*. The rationale behind this separation is that a provides-point is 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. The composition of ARNs (i.e., the binding mechanism of services) is obtained by "fusing" provides-points and requires-points, subject to a certain compliance check between the contract associated to them. For example, in [23] the binding is subject to a (semantic) entailment relation between theories over *linear temporal logic* [57], which are attached to the provides- and the requires-points of the considered networks.

Providing semantics to ARNs requires to carefully combine different elements intervening in the rationale behind the formalism and its intended behaviour. In their first definition, ARNs were given semantics in terms of infinite sequences of sets of actions, which capture the behaviour of the service. In this presentation, the behavioural description was given in terms of linear temporal logic theory presentations [24]. A more modern (and also more operational) presentation of the semantics of ARNs, the one on which we rely in this work, resorts to automata on infinite objects whose inputs consist of sequences of sets of actions (see [62]), as defined in the original semantics of ARNs. Under this formalism, both types of hyperedges are labelled with Muller automata; in the case of process hyperedges, the automata formalise the computation carried out by that particular service, while in the case of communication hyperedges, the automata represent the orchestrator that syncs the behaviour of the participants in the com-

[1] In their original presentation ARNs are graph based.

munication process. The behaviour of the system is then obtained as the composition of the Muller automata associated to both computation and communication hyperedges. Finally, the reconfiguration of networks (realized through the discovery and binding of services) is defined by considering an institutional framework in which signatures are ARNs and models are morphisms into ground ARNs, which have no dependencies on external services (see, e.g., [62] for a more in depth presentation of this semantics).

Under the above-mentioned consideration, the operational semantics of ARNs (as a set of execution traces) is based on the fact that a network can be reconfigured until all its external dependencies (captured by requires-points) are fulfilled, i.e., the original network admits a morphism to a ground ARN.

In this section we present the preliminary definitions we use throughout this work. For hypergraph terminology, notation and definitions, the reader is pointed to [1, 6, 15], while for automata on infinite sequences we suggest [56, 60].

Definition 15 (Hypergraph [6])**.** *Let* $X = \{x_1, x_2, \ldots, x_n\}$ *be a finite set. A* hypergraph *on* X *is a family* $H = (E_1, E_2, \ldots, E_m)$ *of subsets of* X *such that*

1. $E_i \neq \varnothing$ *for all* $1 \leq i \leq m$
2. $\bigcup_{i \in \{1,2,\ldots,m\}} E_i = X$

A simple hypergraph *is a hypergraph* $H = (E_1, E_2, \ldots, E_m)$ *such that*

3. $E_i \subset E_j \implies i = j$

The elements $x_1, x_2, \ldots, x_n$ *of* X *are called* vertices*,* points *or* nodes*, and the sets* $E_1, E_2, \ldots, E_m$ *are the* edges*,* hyperedges *or* hyperarchs *of the hypergraph.*

A bipartite hypergraph is a hypergraph whose hyperedges can be partitioned in two disjoint classes such that no two hyperedges of the same class are adjacent.

Definition 16 (Bipartite hypergraph)**.** *Let* $H = (E_1, \ldots, E_m)$ *be a hypergraph then* H *is a* bipartite hypergraph *if and only if if there exists* $H_0, H_1 \subseteq H$ *such that* $H_0 \cap H_1 = \varnothing$ *and* $H_0 \cup H_1 = H$ *and for all* $e_j, e_k \in H_i$*,* $j \neq k \implies e_j \cap e_k = \varnothing$ *with* $(i \in \{0, 1\})$*.*

To formalize behaviour, ARNs rely on Muller automata. Muller automata are a class of automata on infinite words.

Definition 17 (Muller automaton)**.** *The category* $\mathbb{MA}$ *of (action-based) Muller automata (see, e.g. [62]) is defined as follows:*

The objects *of* $\mathbb{MA}$ *are pairs* $\langle A, \Lambda \rangle$ *consisting of a set* A *(of* actions*) and a* Muller automaton $\Lambda = \langle Q, 2^A, \Delta, I, \mathcal{F} \rangle$ *over the alphabet* 2^A*, where*

- Q *is the set of* states *of* Λ*,*
- $\Delta \subseteq Q \times 2^A \times Q$ *is the* transition relation *of* Λ*, with transitions* $(p, \iota, q) \in \Delta$ *usually denoted by* $p \xrightarrow{\iota} q$*,*
- $I \subseteq Q$ *is the set of* initial states *of* Λ*, and*
- $\mathcal{F} \subseteq 2^Q$ *is the set of* final-state sets *of* Λ*.*

For every pair of Muller automata $\langle A, \Lambda \rangle$ *and* $\langle A', \Lambda' \rangle$*, with* $\Lambda = \langle Q, 2^A, \Delta, I, \mathcal{F} \rangle$ *and* $\Lambda' = \langle Q', 2^{A'}, \Delta', I', \mathcal{F}' \rangle$*, an* $\mathbb{MA}$-morphism $\langle \sigma, h \rangle \colon \langle A, \Lambda \rangle \to \langle A', \Lambda' \rangle$ *consists of functions* $\sigma \colon A \to A'$ *and* $h \colon Q' \to Q$ *such that* $(h(p'), \sigma^{-1}(\iota'), h(q')) \in \Delta$ *whenever* $(p', \iota', q') \in \Delta'$*,* $h(I') \subseteq I$*, and* $h(\mathcal{F}') \subseteq \mathcal{F}$*.*

The composition *of* $\mathbb{MA}$*-morphisms is defined componentwise.*

The intuition behind the definition is that ARNs are hypergraphs where the hyperedges are divided in two classes: computation hyperedges and communication hyperedges. Computation hyperedges represent processes, while communication hyperedges represent communication channels. Hypergraph nodes (also called points) are labelled with *ports*, i.e., with structured sets $M = M^- \cup M^+$ of *publication* (M^-) and *delivery messages* (M^+),[2] along the lines of [5, 9]. At the same time, hyperedges are labelled with Muller automata; thus, both processes and communication channels have an associated behaviour given by their corresponding automata, which interact through (messages defined by) the ports labelling their connected points.

The following definitions formalise the manner in which the computation and communication units are structured to interact with each other.

Definition 18 (Process)**.** *A* process $\langle \gamma, \Lambda \rangle$ *consists of a set* γ *of pairwise disjoint ports and a Muller automaton* Λ *over the set of actions* $A_\gamma = \bigcup_{M \in \gamma} A_M$*, where* $A_M = \{m! \mid m \in M^-\} \cup \{m¡ \mid m \in M^+\}$*.*

As an example, Figure 2.1 (a) depicts a process $\langle \gamma_{\mathsf{TA}}, \Lambda_{\mathsf{TA}} \rangle$ where $\gamma_{\mathsf{TA}} = \{\mathsf{TA}_0, \mathsf{TA}_1, \mathsf{TA}_2\}$ and Λ_{TA} is the automaton presented in Figure 2.1 (b).

[2] Formally, we can define ports as sets M of messages together with a function $M \to \{-, +\}$ that assigns a polarity to every message.

The travel agent is meant to provide hotel and/or flight bookings in the local currency of the customers. Accomplishing this task requires two different interactions to take place: on one hand, the communication with hotel-accommodation providers and with flight-tickets providers, and on the other hand, the communication with a currency-converter provider. In order for the composition of the automata developed along our example to behave well, we need that every automaton is able to stay in any state indefinitely. This behaviour is achieved by forcing every state to have a self-loop labelled with the emptyset. With the purpose of easing the figures we avoid drawing these self-loops. The reader should still understand the descriptions of the automata as if there was a self-loop transition, labelled with the empty set, for every state.

Definition 19 (Connection)**.** *Let γ be a set of pairwise disjoint ports. A* connection $\langle M, \mu, \Lambda\rangle$ *between the ports of γ consists of a set M of messages, a partial attachment injection $\mu_i \colon M \rightharpoonup M_i$ for each port $M_i \in \gamma$, and a Muller automaton Λ over $A_M = \{m! \mid m \in M\} \cup \{m¡ \mid m \in M\}$ such that*

$$(a)\ \ M = \bigcup_{M_i \in \gamma} \mathrm{dom}(\mu_i) \quad \textit{and} \quad (b)\ \ \mu_i^{-1}(M_i^{\mp}) \subseteq \bigcup_{M_j \in \gamma \smallsetminus \{M_i\}} \mu_j^{-1}(M_j^{\pm}).$$

In Figure 2.2 (a) a connection C_0 is shown whose set of messages is the union of the messages of the ports $\mathsf{TA}_1, \mathsf{H}_0, \mathsf{F}_0$ and the family of mappings μ is formed by the trivial identity mapping. In Figure 2.2 (b) the automaton Λ_{C_0} that describes the behaviour of the communication channel is shown. This connection just delivers every published message. Nevertheless it imposes some restrictions to the sequences of messages that can be delivered. For example, notice that after the message $\mathsf{getHotels}$ of TA_1 is received (and delivered), only the message hotels of H_0 is accepted for delivery. In a way, the automaton that labels a connection hyperarch acts as an orchestrator for the communication that takes place through that particular channel.

Given a connection $\langle M, \{\mu_x \colon M \rightharpoonup M_x | x \in X\}, \Lambda\rangle$ we define partial translations $\{A_{\mu_x} \colon A_M \rightharpoonup A_{M_x} \mid x \in X\}$ given by:

- $dom(A_{\mu_x}) = \{m! \mid m \in \mu_x^{-1}(M_x^-)\} \cup \{m \mid m \in \mu_x^{-1}(M_x^+)\}$,
- $A_{\mu_x}(m!) = \mu_x(m)!$ for all $m \in \mu_x^{-1}(M_x^-)$,
- $A_{\mu_x}(m¡) = \mu_x(m)¡$ for all $m \in \mu_x^{-1}(M_x^+)$.

We often designate the partial maps A_{μ_x} simply by μ_x if there is no risk of confusion. Every connection $\langle M, \{\mu_x \colon M \rightharpoonup M_x | x \in X\}, \Lambda\rangle$ defines a family of spans $\{A_M \xleftarrow{\supseteq} dom(\mu_x) \xrightarrow{\mu_x} A_{M_x}\}_{x \in X}$ in $\mathbb{SET}$.

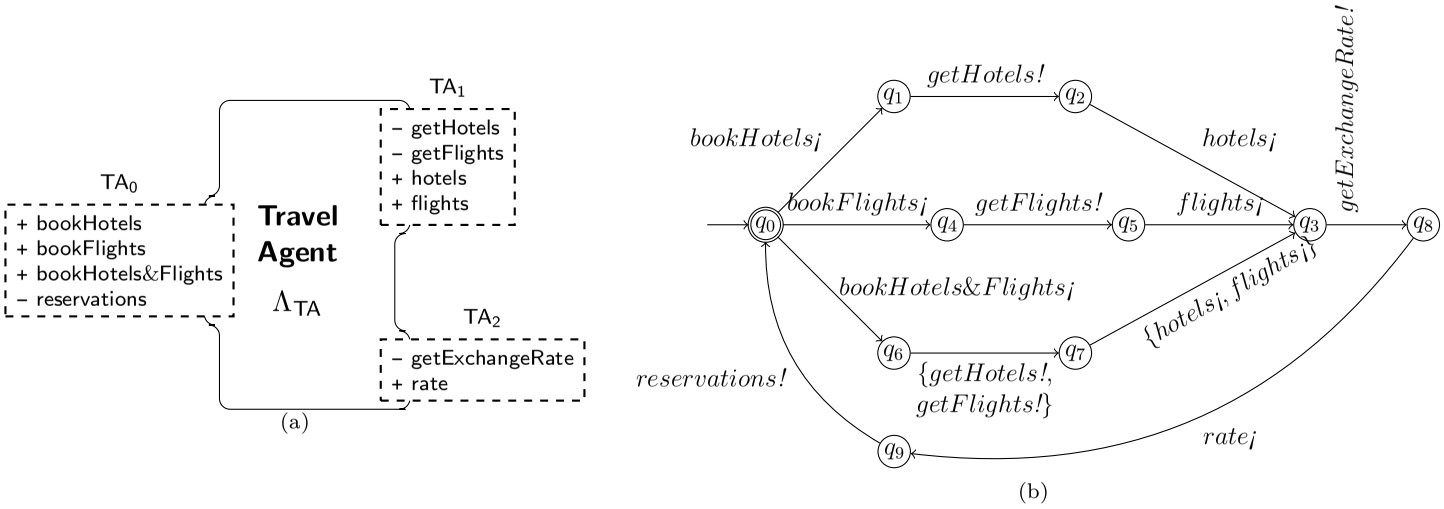


Figure 2.1: The TravelAgent process together with its automaton Λ_{TA}

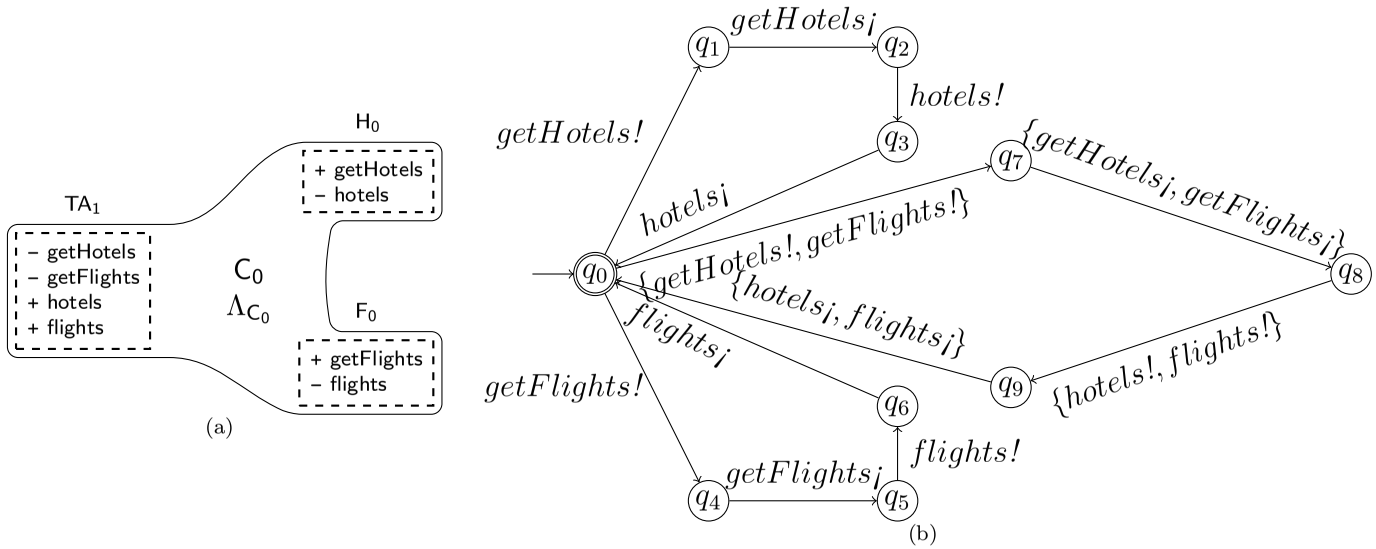


Figure 2.2: The C_0 connection

With these elements we can now define asynchronous relational networks.

Definition 20 (Asynchronous Relational Net [62])**.** *An* asynchronous relational net $\alpha = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ *consists 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,*

- γ_e *is the set of vertices to which the hyperedge* e *is adjacent to, i.e.* $\gamma_e = e$ *for all* $e \in E$*, 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* $\langle M_c, \mu_c, \Lambda_c \rangle$ *over* γ_c *to each hyperedge* $c \in C$*.*

Definition 21 (Morphism of ARNs)**.** *A* morphism $\delta : \alpha \to \alpha'$ *between two ARNs* $\alpha = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$ *and* $\alpha' = \langle X', P', C', \gamma', M', \mu', \Lambda' \rangle$ *consists of*

- *an injective map* $\delta : X \to X'$ *such that* $\delta(P) \subseteq P'$ *and* $\delta(C) \subseteq C'$*, that is an injective homomorphism between the underlying hypergraphs of* α *and* α' *that preserves the computation and communication hyperedges, and*

- *a family of polarity-preserving injections* $\delta^{pt}_x : M_x \to M'_{\delta(x)}$*, for* $x \in X$*,*

such that

- *for every point* $x \in \bigcup P$*,* $\delta^{pt}_x = 1_{M_x}$*,*

- *for every computation hyperedge* $p \in P$*,* $\Lambda_p = \Lambda'_{\delta(p)}$*, and*

- *for every communication hyperedge* $c \in C$*,* $M_c = M'_{\delta(c)}$*,* $\Lambda_c = \Lambda'_{\delta(c)}$ *and, for every point* $x \in \gamma_c$*,* $\mu_{c,x}; \delta^{pt}_x = \mu'_{\delta(c),\delta(x)}$*.*

ARNs together with morphisms of ARNs form a category, denoted $\mathbb{ARN}$, in which the composition is defined component-wise, and left and right identities are given by morphisms whose components are identity functions.

Intuitively, an ARN is a hypergraph for which some of the hyperedges (process hyperedges) formalise computations as Muller automata communicating through ports (identified with nodes of the hypergraph) over a fixed language of actions. Note that the communication between computational

units is not established directly but mediated by a communication hyperedge; the other kind of hyperedge which use Muller automata to formalise communication channels.

In order to define service modules, repositories, and activities, we need to distinguish between two types of interaction-points, i.e. of points that are not incident with both computation and communication hyperedges.

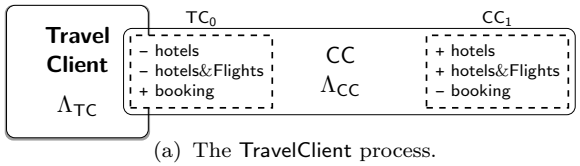
Definition 22 (Requires- and provides-point)**.** *A* requires-point *of an ARN is a point that is incident only with a communication hyperedge. Similarly, a* provides-point *is a point incident only with a computation hyperedge.*

Definition 23 (Service repository)**.** *A* service repository *is just a set $\mathcal{R}$ of* service modules*, that is of triples $\langle P, \alpha, R\rangle$, also written $P \underset{\alpha}{\leftarrow} R$, where α is an ARN, P is a provides-point of α, and R is a finite set of requires-points of α.*

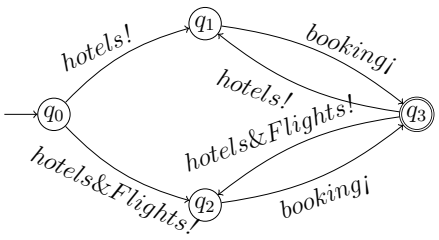
Definition 24 (Activity)**.** *An* activity *is a pair $\langle \alpha, R\rangle$, also denoted $\underset{\alpha}{\vdash} R$, such that α is an ARN and R is a finite set of requires-points of α.*

The previous definitions formalise the idea of a service-oriented software artefact as an activity whose computational requirements are modelled by "dangling" connections, and that do not pursue the provision of any service to other computational unit, modelled as the absence of provides points. Figure 2.3 depicts a TravelClient activity with a single requires-point through which this activity can ask either for hotels or hotels and flights reservations. As we will show in the forthcoming sections, requires-points act as the ports to which the provides-points of services are bound in order to fulfil these requirements.

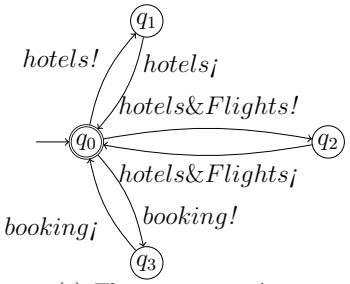
Turning a process into a service available for discovery and binding requires, as we mentioned in the previous definitions, the declaration of the communication channels that will be used to connect to other services. In the case of TravelAgent, three services are required to execute, communicating over two different communication channels. On one of them the process interacts with accommodation providers and flight tickets providers, while through the other the process will obtain exchange rates to be able to show the options to the customer in its local currency. In some sense, TravelAgent provides the ability to coherently combine these three services in order to offer a richer experience. It should then be clear that whenever the TravelAgent is asked for hotels and flights reservations, it will require both services in order to fulfil its task, plus the service for currency exchange conversion. Figure 2.4 (a) shows the TravelAgent service obtained



(a) The TravelClient process.



(b) The automaton Λ_{TC}



(c) The automaton Λ_{CC}

Figure 2.3: The TravelClient activity.

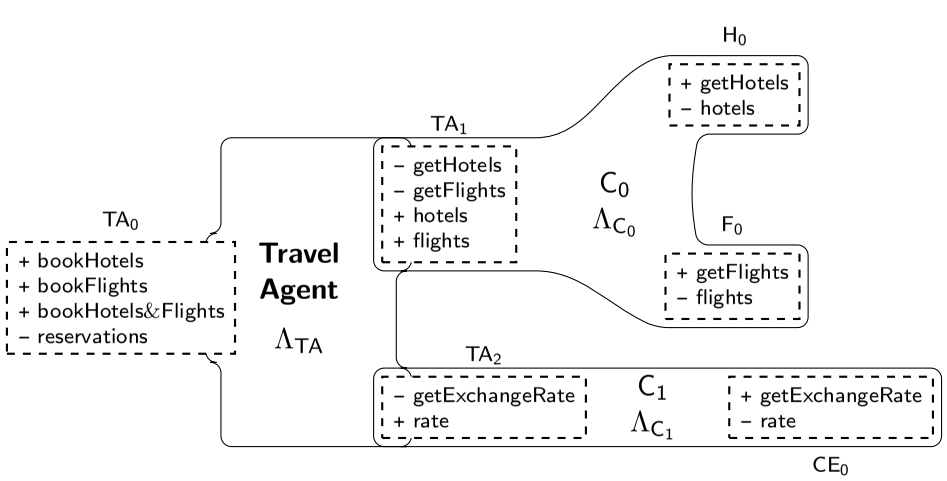
by attaching the communication channels to two of the ports defined by the TravelAgent process, resulting in a network with three requires-points. The Figure 2.4 (b) shows the automaton for the connection C_1.

The alphabet of an ARN is intuitively obtained by considering the alphabet of each process and connection mapped apropriately according to connections.

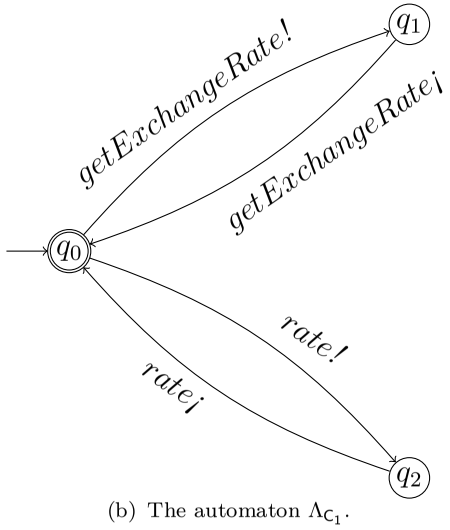
Definition 25 (Alphabet of an ARN)**.** *The* alphabet associated with an ARN α *is the vertex* A_α *of the colimit* $\xi: D_\alpha \Rightarrow A_\alpha$ *of the functor* $D_\alpha: \mathbb{J}_\alpha \to$ Set*, where*

- $\mathbb{J}_\alpha$ *is the preordered category whose objects are points* $x \in X$*, hyperedges* $e \in P \cup C$*, or "attachments"* $\langle c, x\rangle$ *of* α*, with* $c \in C$ *and* $x \in \gamma_c$*, and whose arrows are given by* $\{x \to p \mid p \in P, x \in \gamma_p\}$*, for computation hyperedges, and* $\{c \leftarrow \langle c, x\rangle \to x \mid c \in C, x \in \gamma_c\}$*, for communication hyperedges;*

- D_α *defines the sets of actions associated with the ports, processes and channels, together with the appropriate mappings between them. For example, given a communication hyperedge* $c \in C$ *and a point* $x \in \gamma_c$*,*
  - $D_\alpha(c) = A_{M_c}$, $D_\alpha(\langle c, x\rangle) = dom(\mu_x^c)$, $D_\alpha(x) = A_{M_x}$
  - $D_\alpha(\langle c, x\rangle \to c) = (dom(\mu_x^c) \subseteq A_{M_c})$
  - $D_\alpha(\langle c, x\rangle \to x) = \mu_x^c$



(a) The TravelAgent service.



(b) The automaton Λ_{C_1}.

Figure 2.4: The TravelAgent service module.

In this way we conclude the formal introduction. In the next chapter we present the first contribution of this thesis: a full operational semantics for ARNs.

2.3 Resumen

En este capítulo se introducen las definicinoes y resultados que utilizamos en el resto de la tesis. En primer lugar se introducen nociones generales de teoría de categorías que utilizamos como *background* formal. Luego presentamos las definiciones de *Asynchronous Relational Networks*, un modelo computacional para servicios.

En la sección sobre teoría de categorías introducimos los conceptos provenientes de esta teoría que son utilizados más tarde en la tesis. No pretendemos dar una presentación exhaustiva de teoría de categorías y refereimos al lector interesado a [22] para una introducción amena a la teoría de categorías para científicos de la computación y a [44] para una introducción dirigida a matemáticos.

Una categoría es, intuitivamente, una colección de objetos y morfismos (flechas) entre objetos tal que la composición de morfismos de una categoría es un morfismo de la categoría y las identidades de cada objeto son morfismos de la categoría.

Gran parte del razonamiento en teoría de categorías puede ser llevado a cabo observando propiedades de diagramas. Un diagrama de una categoría puede pensarse como un homomorphismo de un grafo hacia la categoría de modo que los nodos del grafo son mapeados a objetos de la categoría y los ejes del grafo a morfismos. De especial interés son los conos y los coconos (que son conceptos duales). Un cocono de un diagrama es un objeto (el vértice del cocono) en la categoría junto con una familia de morfismos desde los objetos del diagrama hacia el vértice. Un cocono conmutativo con vértice z es un colímite si para todo otro cocono conmutativo con vértice z' existe un único morfismo δ de z a z' tal que ambos coconos conmutan a través de δ.

Existen muchas formas en que las categorías se relacionan unas con otras. La más común es a través de functores. Un functor es un mapeo total de objetos y morfismos de una categoría en objetos y morfismos de otra categoría tal que preserva las identidades y las composiciones. Es decir que un morfismo identidad de un objeto x es mapeado al morfismo identidad del objeto al que es mapeado x y que la aplicación del functor

a la composición de dos morfismos en la categoría de origen es igual a la composición en la categoría de destino de la aplicación del functor a cada uno de los morfismos. Dicho esto se puede definir la categoría Cat cuyos objetos son las categorías y cuyos morfismos son functores.

Las ARNs fueron presentadas por Fiadeiro y Lopes en [24] con el objetivo de formalizar los elementos de una teoría de interfaces para el diseño de software orientado a servicios. Las ARNs son un modelo formal de orquestación basado en hipergrafos cuyos hiperarcos son interpretados o como procesos o como canales de comunicación. Los nodos adyacentes únicamente a hiperarcos de proceso son llamados *provides-points* mientras que aquelos adyacentes únicamente a hiperarcos de comunicación son llamados *requires-points*. La idea detrás de esta separación es que un *provides-point* es la interfaz a través de la cual un servicio exporta su funcionalidad, mientras que un *requires-point* es la interfaz a través de la cual una actividad espera que algún servicio le provea funcionalidad. La composición de ARNs (es decir, el mecanismo de *binding*) se obtiene al fusionar *provides-points* y *requires-points*, sujeto a algún chequeo de conformidad entre los contratos asociados a ellos. Por ejemplo, en [23] el *binding* está sujeto a una relación de *entailment* (semántico) entre teorías sobre *linear temporal logic* [57] que están ligadas a los *provides-* y *requires-poinst* de las ARNs en consideración.

Proporcionar semántica a las ARNs requiere combinar cuidadosamente diferentes elementos que intervienen en la racionalización que hay detrás del formalismo y su comportamiento esperado. En su primera definicion, se les dio a las ARNs una semántico en términos de secuencias inifinitas de conjuntos de acciones, que capturan el comportamiento de un servicio. En esta presentación, la descripción del comportamiento se da en términos de presentaciones de teorías de lógica temporal lineal [24]. Una versión más moderna y, también, más operacional de semántica para ARNs, aquella sobre la que trabajamos en esta tesis, recurre a autómatas sobre objetos infinitos cuyas entradas son secuencias de conjuntos de acciones (ver [62]). Bajo este formalismo, tanto los hiperarcos de proceso como los de comunicación se etiquetan con autómatas de Muller; en el caso de los hiperarcos de proceso el autómata formaliza el cómputo llevado a cabo por ese servicio particular mientras que en el caso de los hiperarcos de comunicación el autómata representa el orquestador que sincroniza el comportamiento de los participantes de la comunicación. El comportamiento del sistema se obtiene entonces como la composición de los autómatas de Muller asociados a los hiperarcos de proceso y de comunicación. Finalmente la reconfiguración de las ARNs se define considerando un framework basado en instituciones en

el que las signaturas son ARNs y los modelos son morfismos hacia ARNs *ground*, que no tienen dependencias de servicios externos. En esta presentación, la semántica operacional se basa en el hecho de que una ARNs pueda ser reconfigurada hasta que todas sus dependencias externas sean satisfechas.

En las ARNs los nodos del hipergrafo (también llamados puntos) son etiquetados con puertos, es decir con conjuntos estructurados $M = M^- \cup M^+$ de *mensajes de publicación* (M^-) y *mensajes de despacho* (M^+), en el sentido de [5,9]. Las acciones de los autómatas que etiquetan los hiperarcos de procesos son conjuntos de mensajes de aquellos puertos (nodos) a los que el hiperarco es adyacente. Por otro lado las conexiones (hiperarcos de comunicación) tiene un alfabeto propio y un mapeo injectivo del lenguaje de cada uno de los puertos a los que son adyacentes al lenguaje interno de la conexión. De modo que las conexiones pueden potencialmente colapsar mensajes de distintos puertos. Las acciones del autómata que etiqueta un hiperarco de comunicación son conjuntos de mensajes del lenguaje de la conexión.

Asimismo se definen morfismos entre ARNs esencialmente como homomorfismos inyectivos entre los hipergrafos subyacentes que preservan los hiperarcos de proceso y de comunicación. Las ARNs junto con sus morfismos forman una categoría en la que la composición se define componente a componente y las identidades a izquierda y derecha están por morfismos cuyos componentes son fuciones identidad.

El alfabeto de una ARN se obtiene, intuitivamente, considerando el alfabeto de cada proceso y conexión mapeado apropiadamente de acuerdo a las conexiones.

Chapter 3

Semantics of Service Oriented Systems

3.1 Operational semantics of ARNS

3.1.1 Introduction and Motivation

The aim of this work is to provide a trace-based operational semantics for service-oriented software designs reflecting the true dynamic nature of run-time discovery and binding of services. This is done by making the reconfiguration of an activity an observable event of its behaviour. In SOC, the reconfiguration events are triggered by particular actions associated with a requires-point; at that moment, the middleware has to procure a service that meets the requirements of the activity from an already known repository of services. From this perspective our proposal is to define execution traces where actions can be either

- internal actions of the activity: actions that are not associated with requires-points, thus executable without the need for reconfiguring the activity, or
- reconfiguration actions: actions that are associated with a requires-point, thus triggering the reconfiguration of the system by means of the discovery and binding of a service providing that computation.

Summarising, the main contributions of this chapter are:

1. we provide a trace-based operational semantics for ARNs reflecting both internal transitions taking place in any of the services already

intervening in the computation and dynamic reconfiguration actions resulting from the process by binding the provides-point of ARNs taken from the repository to its require-points, and

2. we provide support for defining a model-checking technique that can enable the automatic analysis of linear temporal logic properties of activities.

In our work, we consider that semantics is assigned modulo a given repository of services, forcing us to drop the assumption that given an ARN it is possible to find a ground network to which the former has a morphism. Regarding previous works, we believe that this approach results in a more realistic executing environment where the potential satisfaction of requirements is limited by the services registered in a repository, and not by the entire universe of possible services.

In this way, our work departs from previous approaches to dynamic reconfiguration in the context of service-oriented computing, such as [58], which reasons about functional behaviour and control concerns in a framework based on first-order logic, [10], which relies on typed graph-transformation techniques implemented in Alloy [34] and Maude [17], [11], which makes use of graph grammars as a formal framework for dealing with dynamicity, and [12, 27], which proposes architectural design rewriting as a term-rewriting-based approach to the development and reconfiguration of software architectures. A survey of these general logic-, graph-, or rewriting-based formalisms can be found in [8].

This chapter is organised as follows. Preliminary definitions were given in Chapter 2. In Section 3.1.2 we give appropriate definitions for providing operational semantics for ARNs based on a (quasi) automaton *generated* by a repository and on the traces accepted by it. We also provide a variant of Linear Temporal Logic (in Section 3.1.5) that is suitable for defining and checking properties related to the execution of activities. As a running example, we gradually introduce the details of travel-agent scenario, which we use to illustrate the concepts presented in the Section 3.1.2 and Section 3.1.5. Finally in Section 3.3 we draw some conclusions and discuss further lines of research.

3.1.2 Operational Semantics for ARNs

As we mentioned before, in this chapter we focus on providing semantics to service-oriented software artefacts. To do that, we resort to the formal language of *asynchronous relational nets* (see, e.g., [24]). In this section we present the main contribution of the chapter, being a full operational semantics for activities executing with respect to a given repository. To do this, we introduce two different kinds of transitions for activities:

1. internal transitions, those resulting from the execution of a certain set of actions by the automata that synchronise over them, and

2. reconfiguration transitions, the ones resulting from the need of executing a set of actions on a port of a communication hyperedge.

Then, runs (on given traces) are legal infinite sequences of states related by appropriate transitions.

We recall Definition 25 from Chapter 2:

Definition (Alphabet of an ARN)**.** *The* alphabet associated with an ARN α *is the vertex* A_α *of the colimit* $\xi: D_\alpha \Rightarrow A_\alpha$ *of the functor* $D_\alpha: \mathbb{J}_\alpha \to \mathbb{S}\mathrm{et}$*, where*

- $\mathbb{J}_\alpha$ *is the preordered category whose objects are points* $x \in X$*, hyperedges* $e \in P \cup C$*, or "attachments"* $\langle c, x\rangle$ *of* α*, with* $c \in C$ *and* $x \in \gamma_c$*, and whose arrows are given by* $\{x \to p \mid p \in P, x \in \gamma_p\}$*, for computation hyperedges, and* $\{c \leftarrow \langle c, x\rangle \to x \mid c \in C, x \in \gamma_c\}$*, for communication hyperedges;*

- D_α *defines the sets of actions associated with the ports, processes and channels, together with the appropriate mappings between them.*

Definition 26 (Automaton of an ARN)**.** *Let* $\alpha = \langle X, P, C, \gamma, M, \mu, \Lambda\rangle$ *be an ARN and* $\langle Q_e, 2^{A_{M_e}}, \Delta_e, I_e, \mathcal{F}_e\rangle$ *be the components of* Λ_e*, for each* $e \in P \cup C$*. The automaton* $\Lambda_\alpha = \langle Q_\alpha, 2^{A_\alpha}, \Delta_\alpha, I_\alpha, \mathcal{F}_\alpha\rangle$ associated with α *is defined as follows:*

$$
\begin{aligned}
Q_\alpha &= \textstyle\prod_{e \in P \cup C} Q_e,\\
\Delta_\alpha &= \{(p, \iota, q) \mid (\pi_e(p), \xi_e^{-1}(\iota), \pi_e(q)) \in \Delta_e \text{ for each } e \in P \cup C\},\\
I_\alpha &= \textstyle\prod_{e \in P \cup C} I_e, \text{ and}\\
\mathcal{F}_\alpha &= \{F \subseteq Q_\alpha \mid \pi_e(F) \in \mathcal{F}_e \text{ for all } e \in P \cup C\},
\end{aligned}
$$

where $\pi_e: Q_\alpha \to Q_e$ *are the corresponding projections of the product* $\prod_{e \in P \cup C} Q_e$*.*

Proposition 4. *Under the notations of Definition 26, for every hyperedge e of α, the maps ξ_e and π_e define an $\mathbb{MA}$-morphism $\langle \xi_e, \pi_e \rangle : \langle A_{M_e}, \Lambda_e \rangle \to \langle A_\alpha, \Lambda_\alpha \rangle$.*

Proof. According to Definition 17 an $\mathbb{MA}$-*morphism* $\langle \sigma, h \rangle : \langle A, \Lambda \rangle \to \langle A', \Lambda' \rangle$ consists of functions $\sigma : A \to A'$ and $h : Q' \to Q$ such that:

1. $(h(p'), \sigma^{-1}(\iota'), h(q')) \in \Delta$ whenever $(p', \iota', q') \in \Delta'$,
2. $h(I') \subseteq I$, and
3. $h(\mathcal{F}') \subseteq \mathcal{F}$.

It is immediate to verify that $\langle \xi_e, \pi_e \rangle$ satisfies both 2 and 3 just by observing that $\pi_e(I') = I_e$ and $\pi_e(\mathcal{F}') = \mathcal{F}_e$. To see that it also satisfies 1 it suffices to observe that the definition of Δ_α establishes precisely that.

□

Intuitively, the automaton of an ARN is the automaton resulting from taking the product of the automata of the several components of the ARN. This product is synchronized over the shared alphabet of the components. Notice that the notion of *shared alphabet* is given by the mappings defined in the connections.

Proposition 5. *For every ARN $\alpha = \langle X, P, C, \gamma, M, \mu, \Lambda \rangle$, the $\mathbb{MA}$-morphisms $\langle \xi_e, \pi_e \rangle : \langle A_{M_e}, \Lambda_e \rangle \to \langle A_\alpha, \Lambda_\alpha \rangle$ associated with hyperedges $e \in P \cup C$ form colimit injections for the functor $G_\alpha : \mathbb{J}_\alpha \to \mathbb{MA}$ that maps*

- *every computation or communication hyperedge $e \in P \cup C$ to $\langle A_{M_e}, \Lambda_e \rangle$ and*
- *every point $x \in X$ (or attachment $\langle c, x \rangle$) to $\langle A_{M_x}, \Lambda_x \rangle$, where Λ_x is the Muller automaton $\langle \{q\}, 2^{A_{M_x}}, \{(q, \iota, q) \mid \iota \subseteq A_{M_x}\}, \{q\}, \{\{q\}\} \rangle$ with only one state, which is both initial and final, and with all possible transitions.*

Therefore, both the alphabet and the automaton of an ARN α are given by the vertex $\langle A_\alpha, \Lambda_\alpha \rangle$ of a colimiting cocone of the functor $G_\alpha : \mathbb{J}_\alpha \to \mathbb{MA}$.

Proof. The universal property of A_α is a result of it being the vertex of the co-limiting co-cocone formed by the injections ξ_e (by Definition 25). On the other side Λ_α is the product automaton of all the Λ_e (note that it is actually the product of the co-free expansions of Λ_e along ξ_e) and therefore is the vertex of the limiting cone formed by the π_e projections. Thus the universal property of the pair is proved. □

The universality property of the alphabet and of the automaton of an ARN discussed above allows us to extend Definition 26 to morphisms of networks.

Corollary 1. *For every morphism of ARNs $\delta\colon \alpha \to \alpha'$ there exists a unique $\mathbb{MA}$-morphism $\langle A_\delta, \_{\restriction_\delta}\rangle\colon \langle A_\alpha, \Lambda_\alpha\rangle \to \langle A_{\alpha'}, \Lambda_{\alpha'}\rangle$ such that*

$$(a)\ \ \xi_x ; A_\delta = \xi'_{\delta(x)} \quad \textit{and} \quad (b)\ \ (\_{\restriction_\delta}) ; \pi_x = \pi'_{\delta(x)}$$

for every point or hyperedge x of α, where $\langle \xi_x, \pi_x\rangle$ and $\langle \xi'_{x'}, \pi'_{x'}\rangle$ are components of the colimiting cocones of the functors $G_\alpha\colon \mathbb{J}_\alpha \to \mathbb{MA}$ and $G_{\alpha'}\colon \mathbb{J}_{\alpha'} \to \mathbb{MA}$.[1]

Operational semantics of ARNs. From a categorical perspective, the uniqueness aspect of Corollary 1 is particularly important in capturing the operational semantics of ARNs in a fully abstract manner: it enables us to describe both automata and morphisms of automata associated with ARNs and morphisms of ARNs through a functor $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$ that maps every ARN α to $\langle A_\alpha, \Lambda_\alpha\rangle$ and every morphisms of ARNs $\delta\colon \alpha \to \alpha'$ to $\langle A_\delta, \_{\restriction_\delta}\rangle$.

3.1.3 Open Executions of ARNs

In order to formalise open executions of ARNs, i.e. of executions in which not only the states of the underlying automata of ARNs can change as a result of the publication or the delivery of various messages, but also the ARNs themselves through discovery and binding to other networks, we rely on the usual automata-theoretic notions of execution, trace, and run, which we consider over a particular (super-)automaton of ARNs and local states of their underlying automata.

Definition 27. *The "flattened" automaton $\mathcal{A}^\sharp = \langle Q^\sharp, A^\sharp, \Delta^\sharp, I^\sharp, \mathcal{F}^\sharp\rangle$ induced by the functor $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$*[2] *is defined as:*

$$\begin{aligned}
Q^\sharp &= \{\langle \alpha, q\rangle \mid \alpha \in |\mathbb{ARN}| \textit{ and } q \in Q_\alpha\},\\
A^\sharp &= \{\langle \delta, \iota\rangle \mid \delta\colon \alpha \to \alpha' \textit{ and } \iota \subseteq A_\alpha\},\\
\Delta^\sharp &= \{(\langle \alpha, q\rangle, \langle \delta, \iota\rangle, \langle \alpha', q'\rangle) \mid \delta\colon \alpha \to \alpha' \textit{ and } (q, \iota, q'{\restriction_\delta}) \in \Delta_\alpha\},\\
I^\sharp &= \{\langle \alpha, q\rangle \mid \alpha \in |\mathbb{ARN}| \textit{ and } q \in I_\alpha\}, \textit{ and}\\
\mathcal{F}^\sharp &= \{\{\langle \alpha, q\rangle \mid q \in F\} \mid \alpha \in |\mathbb{ARN}| \textit{ and } F \in \mathcal{F}_\alpha\}.
\end{aligned}$$

[1] The definitions of G_α and $G_{\alpha'}$ follow the presentation given in Proposition 5.

[2] Note that $\Lambda^\sharp$ is in fact a quasi-automaton, because its components are proper classes.

This "flattened" automaton amalgamates in a single structure both the configuration and the state of the system. These two elements are viewed as a pair ⟨ARN, state⟩. Now the transitions in this automaton can represent state changes and structural changes together. In this sense, the "flattened" automaton achieves the goal of giving us a unified view of both aspects of a service oriented system. Inlcuding the ARN in the state of the flattened automaton highlighs the fact that the structure is dynamic. The construction of this automaton can be seen, from a categorical point of view, as the flattening of the indexed category induced by $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$.

We recall that a *trace* over a set A of actions is an infinite sequence $\lambda \in (2^A)^\omega$, and that a *run* of a Muller automaton $\Lambda = \langle Q, 2^A, \Delta, I, \mathcal{F}\rangle$ on a trace λ is a sequence of states $\varrho \in Q^\omega$ such that $\varrho(0) \in I$ and $(\varrho(i), \lambda(i), \varrho(i+1)) \in \Delta$ for every $i \in \omega$; together, λ and ϱ form an *execution* of the automaton Λ. An execution $\langle \lambda, \rho\rangle$, or simply the run ϱ, is *successful* if the set of states that occur infinitely often in ϱ, denoted $\mathrm{Inf}(\varrho)$, is a member of $\mathcal{F}$. Furthermore, a trace λ is *accepted* by Λ if and only if there exists a successful run of Λ on λ.

Definition 28 (Open execution of an ARN)**.** *An* open execution *of an ARN α is an execution of $\mathcal{A}^\sharp$ that starts from an initial state of Λ_α, i.e. a sequence*

$$\langle \alpha_0, q_0\rangle \xrightarrow{\delta_0,\iota_0} \langle \alpha_1, q_1\rangle \xrightarrow{\delta_1,\iota_1} \langle \alpha_2, q_2\rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

such that $\alpha_0 = \alpha$, $q_0 \in I_\alpha$ and, for every $i \in \omega$, $\langle \alpha_i, q_i\rangle \xrightarrow{\langle \delta_i,\iota_i\rangle} \langle \alpha_{i+1}, q_{i+1}\rangle$ is a transition in $\Delta^\sharp$. An open execution as above is successful *if it is successful with respect to the automaton $\mathcal{A}^\sharp$, i.e. if there exists $i \in \omega$ such that*

a) for all $j \geq i$, $\alpha_j = \alpha_i$, $\delta_j = 1_{\alpha_i}$, and

b) $\{q_j \mid j \geq i\} \in \mathcal{F}_{\alpha_i}$.

Based on the definition of the transitions of $\mathcal{A}^\sharp$ and on the functoriality of $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$, it is easy to see that, for every ARN α, every successful open execution of α gives a successful execution of its underlying automaton Λ_α.

Proposition 6. *For every (successful) open execution*

$$\langle \alpha_0, q_0\rangle \xrightarrow{\delta_0,\iota_0} \langle \alpha_1, q_1\rangle \xrightarrow{\delta_1,\iota_1} \langle \alpha_2, q_2\rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

of the quasi-automaton $\mathcal{A}^\sharp$, the infinite sequence

$$q_0 \xrightarrow{\iota_0} q_1{\restriction_{\delta_0}} \xrightarrow{A^{-1}_{\delta_0}(\iota_1)} q_2{\restriction_{\delta_0;\delta_1}} \xrightarrow{A^{-1}_{\delta_0;\delta_1}(\iota_2)} \cdots$$

corresponds to a (successful) execution of the automaton Λ_{α_0}.

Proof. We need to show that the following sequence

$$q_0 \xrightarrow{\iota_0} q_1 \restriction_{\delta_0} \xrightarrow{A^{-1}_{\delta_0}(\iota_1)} q_2 \restriction_{\delta_0;\delta_1} \xrightarrow{A^{-1}_{\delta_0;\delta_1}(\iota_2)} \cdots$$

is an execution of Λ_{α_0}. As a consequence of Corollary 1 and the compositionality of morphisms it is straightforward to show that $q_i \restriction_{\delta_0;\ldots;\delta_{i-1}}$ is indeed a state of Λ_{α_0}. In the same way we can see that $A^{-1}_{\delta_0;\ldots;\delta_{i-1}}(\iota_i)$ is indeed a transition from $q_i \restriction_{\delta_0;\ldots;\delta_{i-1}}$ to $q_{i+1} \restriction_{\delta_0;\ldots;\delta_i}$. Remember that we are assuming that every automaton can stay in a state indefinitely by means of an $\varnothing$ transition.

We can see that the projection by $\restriction_{\delta_0;\delta_1;\ldots;\delta_i}$ of the set of infinitely often visited states is indeed an element of $\mathcal{F}_0$ by resorting to Definition 26 and Definition 27. In particular if the execution is successful, because of Definition 27, the set of infinitely often visited states is a set $\{\langle \alpha_i, q\rangle \mid q \in F\} \mid F \in \mathcal{F}_{\alpha_i}$ for some $i \geq 0$. Therefore, as $\delta_0;\ldots;\delta_i : \alpha_0 \to \alpha_i$ and using Definition 26, the reduct $\delta_0;\ldots;\delta_i$-reduct of that set is indeed an element of $\mathcal{F}_0$. □

Note that, since the restrictions imposed to the transitions of $\mathcal{A}^\sharp$ are very weak – more precisely, because there are no constraints on the morphisms of ARN $\delta{:}\alpha \to \alpha'$ underlying open-transitions $\langle \alpha, q\rangle \xrightarrow{\delta,\iota} \langle \alpha', q'\rangle$ – Proposition 6 cannot be generalised to executions of the automata Λ_{α_i}, for $i > 0$. To address this aspect, we need to take into consideration the fact that, in practice, the reconfigurations of ARNs are actually triggered by certain actions of their alphabet, and that they comply with the general rules of the process of service discovery and binding. Therefore, we need to consider open executions of activities with respect to given service repositories.

3.1.4 Open Executions of Activities

For the rest of this section we assume that $\mathcal{R}$ is an arbitrary but fixed repository of service modules.

Definition 29. *The* activity (quasi-)automaton $\mathcal{R}^\sharp = \langle Q^{\mathcal{R}}, A^{\mathcal{R}}, \Delta^{\mathcal{R}}, I^{\mathcal{R}}, \mathcal{F}^{\mathcal{R}}\rangle$ *generated by the service repository* $\mathcal{R}$ *is defined as follows:*

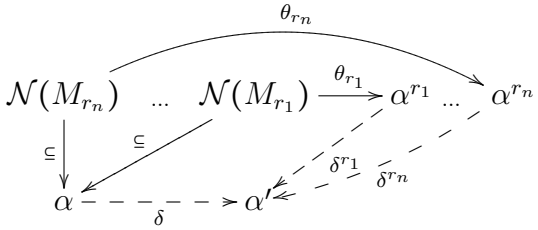
The states *in* $Q^{\mathcal{R}}$ *are pairs* $\langle \underset{\alpha}{\vdash\!\!\!-} R, q\rangle$, *where* $\underset{\alpha}{\vdash\!\!\!-} R$ *is an activity – i.e.* α *is an ARN and* R *is a finite set of requires-points of* α *–* and q *is a state of* Λ_α.

The alphabet $A^{\mathcal{R}}$ *is given by pairs* $\langle \delta, \iota \rangle$*, where* $\delta \colon \alpha \to \alpha'$ *is a morphism of ARNs and* ι *is a set of* α*-actions; thus,* $A^{\mathcal{R}}$ *is just the alphabet of* $\mathcal{A}^\sharp$*.*

There exists a transition $\langle \underset{\alpha}{\vdash\!\!\!-} R, q \rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!\!-} R', q' \rangle$ *whenever:*

1. $\langle \alpha, q \rangle \xrightarrow{\delta,\iota} \langle \alpha', q' \rangle$ *is a transition of* $\mathcal{A}^\sharp$*;*
2. *for each requires-point* $r \in R$ *such that* $\xi_r(A_{M_r^+}) \cap \iota \neq \varnothing$ *there exists*
   - *a service module* $P^r \underset{\alpha^r}{\longleftarrow\!\!\!-} R^r$ *in* $\mathcal{R}$ *and*
   - *a polarity-preserving injection* $\theta_r : M_r \to M_{P^r}$

   such that the following colimit can be defined in the category of ARNs



   where $\{r_1, \ldots, r_n\}$ *is the biggest subset of* R *such that* $\xi_{r_i}(A_{M_{r_i}^+}) \cap \iota \neq \varnothing$ *for all* $1 \leq i \leq n$ *and* $\mathcal{N}(M_{r_i})$ *is the atomic ARN that consists of only one point, labelled with the port* M_{r_i}*, and no hyperedges;*
3. *there exists a transition* $p' \xrightarrow{\iota'} q'$ *of* $\Lambda_{\alpha'}$ *such that* $p'\restriction_\delta = q$*,* $A_\delta^{-1}(\iota') = \iota$ *and, for each requires-point* $r \in R$ *as above,* $p'\restriction_{\delta^r}$ *is an initial state of* Λ_{α^r}*.*

The states *in* $I^{\mathcal{R}}$ *are those pairs* $\langle \underset{\alpha}{\vdash\!\!\!-} R, q \rangle$ *for which* $q \in I_\alpha$*.*

The final-state sets *in* $\mathcal{F}^{\mathcal{R}}$ *are those sets* $\{\langle \underset{\alpha}{\vdash\!\!\!-} R, q \rangle \mid q \in F\}$ *for which* $F \in \mathcal{F}_\alpha$*.*

Note that the definition of the transitions of $\mathcal{R}^\sharp$ integrates both the operational semantics of ARNs given by the functor $\mathcal{A} \colon \mathbb{ARN} \to \mathbb{MA}$ and the logic-programming semantics of service discovery and binding described in [62], albeit in a simplified form, since here we do not take into account the linear temporal sentences that label requires-points. The removal of linear temporal sentences does not limit the applicability of the theory, but rather enables us to give a clearer and more concise presentation of the operational semantics of activities.

Open executions of activities can be defined relative to the automaton $\mathcal{R}^\sharp$ in a similar way to the open executions of ARNs (see Definition 28).

Definition 30 (Open execution of an activity)**.** *An* open execution *of an activity* $\underset{\alpha}{\vdash\!\!-} R$ *with respect to* $\mathcal{R}$ *is an execution of the quasi-automaton* $\mathcal{R}^\sharp$ *that starts from an initial state of* Λ_α*, i.e. a sequence of transitions of* $\mathcal{R}^\sharp$

$$\langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0\rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1\rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2\rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

such that $\alpha_0 = \alpha$*,* $R_0 = R$*, and* $q_0 \in I_\alpha$*. An open execution as above is* successful *if there exists* $i \in \omega$ *such that* (a) *for all* $j \geq i$*,* $\alpha_j = \alpha_i$*,* $\delta_j = 1_{\alpha_i}$*, and* (b) $\{q_j \mid j \geq i\} \in \mathcal{F}_{\alpha_i}$*.*

To illustrate open executions, let's consider a repository $\mathcal{R}$ formed by the service TravelAgent (depicted in Figure 2.4) and the very simple services CurrenciesAgent, AccomodationAgent and FlightsAgent described in Figure 3.1. Let's also consider the TravelClient activity of Figure 2.3. Observing the automata of Figure 2.3 (b) and Figure 2.3 (c), an execution starts with the activity TravelClient performing one of two actions, *hotels*! or *hotels*&*Flights*!. Let us assume it is *hotels*! without loss of generality. The prefix of the execution after the transition has the following shape:

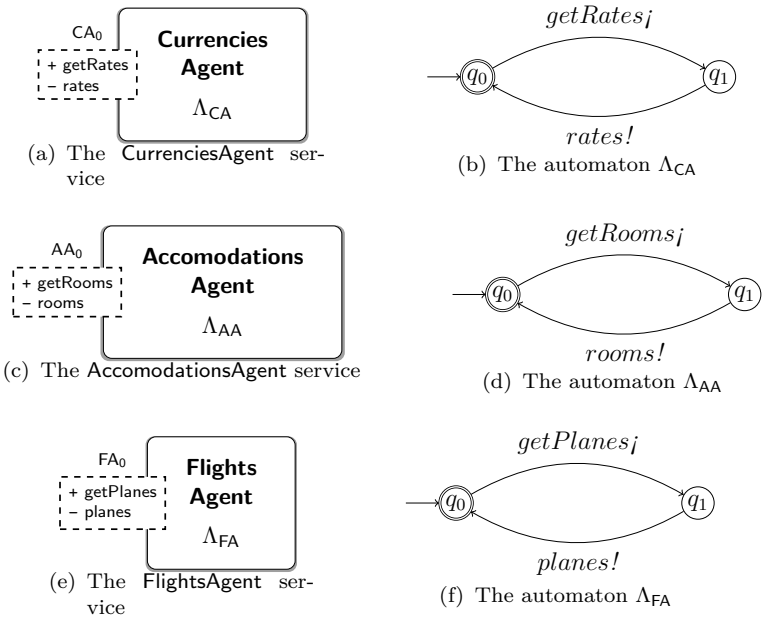
$$\langle \underset{\mathsf{TravelClient}}{\vdash\!\!-} \{\mathsf{CC}_1\}, q_0\rangle \xrightarrow{id,hotels!} \langle \underset{\mathsf{TravelClient}}{\vdash\!\!-} \{\mathsf{CC}_1\}, q_1\rangle$$

where q_0 and q_1 are the states (q_0, q_0) and (q_1, q_1) of the composed automaton $\Lambda_{TC} \times \Lambda_{CC}$ respectively. After this, the only plausible action in this run is the delivery of the message *hotels* by the communication channel CC. Since $\xi_{\mathsf{TravelClient}}(A_{M^+_{\mathsf{CC}_1}}) \cap \{hotels_i\} = \{hotels_i\}$ this action triggers a reconfiguration of the activity. In our example's repository, $\mathcal{R}$, the only service that can satisfy the requirement CC_1 is TravelAgent. Thus, the action $hotels_i$ leads us to the activity TravelClient′ shown in Figure 3.2. The prefix of the execution after this last transition is:

$$\cdots \xrightarrow{id,hotels!} \langle \underset{\mathsf{TravelClient}}{\vdash\!\!-} \{\mathsf{CC}_1\}, q_1\rangle \xrightarrow{\delta,hotels_i} \langle \underset{\mathsf{TravelClient}'}{\vdash\!\!-} \{\mathsf{H}_0, \mathsf{F}_0, \mathsf{CE}_0\}, q_2\rangle$$

where q_2 is the state $(q_1, q_0, q_1, q_0, q_0)$ of the automaton of TravelClient′. To see that the morphism $\delta : \mathsf{TravelClient} \to \mathsf{TravelClient}'$ exists is straightforward.

A continuation for this execution is obtained by the automaton Λ_{TA}, associated with TravelAgent, publishing the action *getHotels!* and the mandatory delivery *getHotels!* that comes after. This actions trigger a new reconfiguration of the activity on port H_0 of the communication channel



(a) The CurrenciesAgent service

(b) The automaton Λ_{CA}

(c) The AccomodationsAgent service

(d) The automaton Λ_{AA}

(e) The FlightsAgent service

(f) The automaton Λ_{FA}

Figure 3.1: Very simple services in $\mathcal{R}$

C_0; in this case, and considering once again our repository $\mathcal{R}$, the result of the reconfiguration should be the *attachment* of the service module AccomodationsAgent.

The following fact allows us to easily generalise Proposition 6 from open executions of ARNs to open executions of activities.

Fact 1. *There exists a (trivial) forgetful morphism of Muller automata $\mathcal{R}^\sharp \to \mathcal{A}^\sharp$ that maps every state $\langle \underset{\alpha}{\vdash\!\!-} R, q \rangle$ of $\mathcal{R}^\sharp$ to the state $\langle \alpha, q \rangle$ of $\mathcal{A}^\sharp$.*

Proposition 7. *For every (successful) execution*

$$\langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

of the activity quasi-automaton $\mathcal{R}^\sharp$, the infinite sequence

$$q_0 \xrightarrow{\iota_0} q_1 \restriction_{\delta_0} \xrightarrow{A^{-1}_{\delta_0}(\iota_1)} q_2 \restriction_{\delta_0;\delta_1} \xrightarrow{A^{-1}_{\delta_0;\delta_1}(\iota_2)} \cdots$$

is a (successful) execution of the automaton Λ_{α_0}.

Proof. This follows directly from Proposition 6. □

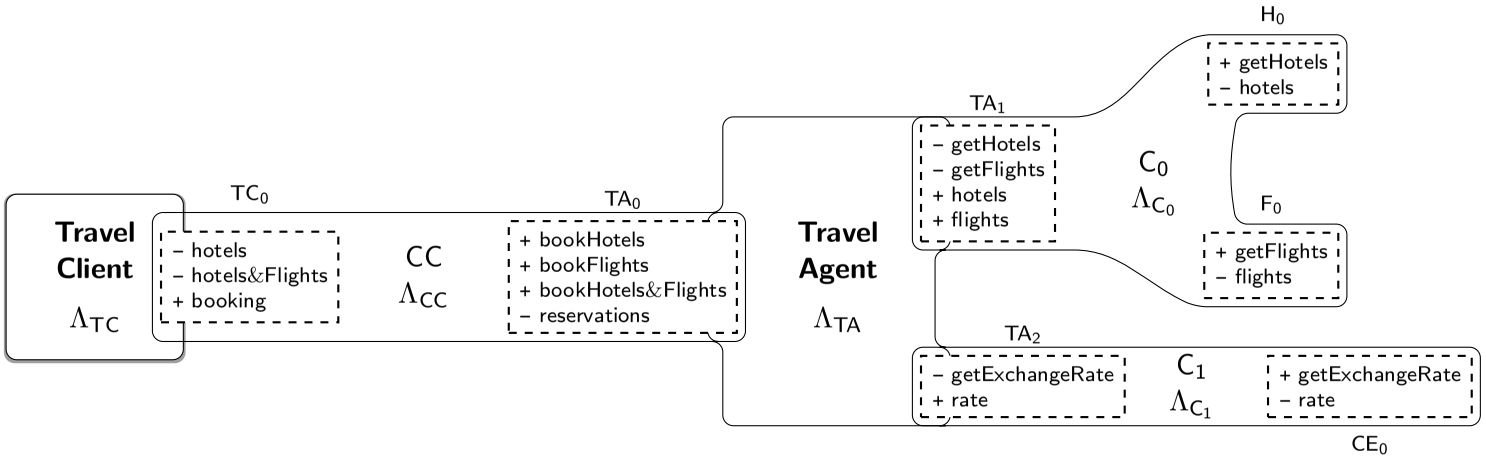


Figure 3.2: The TravelClient′ activity

Theorem 1 shows the relation that exists between the traces of an activity with respect to a respository and the automaton of each component of the activity. It shows that every (successful) open execution of an activity can be projected to a (successful) execution of each of the automata interveaning. In order to prove that open executions of activities give rise to "local" executions of Λ_{α_i} – for every $i \in \omega$, not only for $i = 0$ – we rely on a consequence of the fact that the functor $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$ preserves colimits and, in addition, we restrict the automata associated with the underlying ARNs of service modules.

Proposition 8. *The functor $\mathcal{A}\colon \mathbb{ARN} \to \mathbb{MA}$ preserves colimits. In particular, for every transition $\langle \underset{\alpha}{\longmapsto} R, q\rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\longmapsto} R', q'\rangle$ as in Definition 29, the Muller automaton $\Lambda_{\alpha'}$ is isomorphic with the product*

$$\Lambda_\alpha^\delta \times \prod_{\substack{r\in R\\ \xi_r(A_{M_r^+})\cap\iota\neq\varnothing}} \Lambda_{\alpha^r}^{\delta^r}$$

of the cofree expansions Λ_α^δ and $\Lambda_{\alpha^r}^{\delta^r}$, for $r \in R$ such that $\xi_r(A_{M_r^+}) \cap \iota \neq \varnothing$, of the automata Λ_α and Λ_{α^r} along the alphabet maps A_δ and A_{δ^r}, respectively.[3]

Consequently, a transition $p' \xrightarrow{\iota'} q'$ is defined in the automaton $\Lambda_{\alpha'}$ if and only if $p'\restriction_\delta \xrightarrow{A_\delta^{-1}(\iota')} q'\restriction_\delta$ is a transition of Λ_α and, for each $r \in R$ such that $\xi_r(A_{M_r^+}) \cap \iota \neq \varnothing$, $p'\restriction_{\delta^r} \xrightarrow{A_{\delta^r}^{-1}(\iota')} q'\restriction_{\delta^r}$ is a transition of Λ_{α^r}.

Definition 31 (Idle initial states)**.** *An automaton $\Lambda = \langle Q, 2^A, \Delta, I, \mathcal{F}\rangle$ is said to have* idle initial states *if for every initial state $q \in I$ there exists a transition $(p, \varnothing, q) \in \Delta$ such that p is an initial state too.*

Theorem 1. *If, for every service module $P \underset{\alpha}{\longleftarrow} R$ in $\mathcal{R}$, the automaton Λ_α has idle initial states, then for every (successful) execution*

$$\langle \underset{\alpha_0}{\longmapsto} R_0, q_0\rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\longmapsto} R_1, q_1\rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\longmapsto} R_2, q_2\rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

of $\mathcal{R}^\natural$ there exists a (successful) execution of Λ_{α_i}, for $i \in \omega$, of the form

$$q_0' \xrightarrow{\iota_0'} q_1' \xrightarrow{\iota_1'} \cdots q_{i-1}' \xrightarrow{\iota_{i-1}'} q_i \xrightarrow{\iota_i} q_{i+1}\restriction_{\delta_i} \xrightarrow{A_{\delta_i}^{-1}(\iota_{i+1})} q_{i+2}\restriction_{\delta_i;\delta_{i+1}} \xrightarrow{A_{\delta_i;\delta_{i+1}}^{-1}(\iota_{i+2})} \cdots$$

where, for every $j < i$, $q_j'\restriction_{\delta_j;\cdots;\delta_{i-1}} = q_j$ and $A_{\delta_j;\cdots;\delta_{i-1}}^{-1}(\iota_j') = \iota_j$

[3] We recall from [62] that the cofree expansion of an automaton $\Lambda = \langle Q, 2^A, \Delta, I, \mathcal{F}\rangle$ along a map $\sigma\colon A \to A'$ is the automaton $\Lambda' = \langle Q, 2^{A'}, \Delta', I, \mathcal{F}\rangle$ for which $(p, \iota', q) \in \Delta'$ if and only if $(p, \sigma^{-1}(X'), q) \in \Delta$.

Proof. This result can be proved by induction on i. The base case results directly from Proposition 7, while the induction step relies on condition 3 of Definition 29 and on Proposition 8. □

The reader should notice that all the automata used as examples in this work have *idle initial states* as a consequence of the hidden self loop, labelled with the empty set, that we assumed to exist in every state.

3.1.5 Satisfiability of Linear Temporal Logic Formulae

In this section we show how we can use the trace semantics we presented in the previous section to reason about Linear Temporal Logic (LTL for short) [42, 57] properties of activities. Next we define linear temporal logic by providing its grammar and semantics in terms of sets of traces.

Definition 32. *Let $\mathcal{V}$ be a set of proposition symbols, then the set of LTL formulae on $\mathcal{V}$, denoted as LTLForm($\mathcal{V}$), is the smallest set S such that:*

- $\mathcal{V} \subseteq S$, *and*
- *if* $\phi, \psi \in S$, *then* $\{\neg\phi, \phi \vee \psi, \mathbf{X}\phi, \phi\mathbf{U}\psi\} \subseteq S$.

We consider the signature of a repository to be the union of all messages of all the service modules in it. This can give rise to an infinite language over which it is possible to express properties refering to any of the services in the repository, even those that are not yet bound (and might never be). To achieve this we require the alphabets of the service modules in a repository $\mathcal{R}$ to be pairwise disjoint.

Definition 33. *Let $\mathcal{R}$ be a repository and $\underset{\alpha}{\vdash\!\!-} R$ an activity. We denote with $A_{\mathcal{R},\alpha}$ the set* $\left(\bigcup \{A_{\alpha'}\}_{P' \xleftarrow[\alpha']{} R' \in \mathcal{R}}\right) \cup A_\alpha$

Defining satisfaction of an LTL formula requires that we first define what is the set of propositions over which we can express the LTL formulae. We consider as the set of propositions all the actions in the signature of the repository or in the activity to which we are providing semantics. Thus, the propositions that hold in a particular state will be the ones that correspond to the actions in the label of the transition that took the system to that state.

In order to define if a run satisfies an LTL formula it is necessary to consider the suffixes of a run, thus let

$$r = \langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0, \iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1, \iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2, \iota_2} \cdots$$

be a succesful open execution of $\underset{\alpha}{\vdash\!\!-} R$ with respect to a repository $\mathcal{R}$ we denote with r_i the i^{th} suffix of r. That is:

$$r_i{=}\langle \underset{\alpha_i}{\vdash\!\!-} R_i, q_i\rangle \xrightarrow{\delta_i,\iota_i} \langle \underset{\alpha_{i+1}}{\vdash\!\!-} R_{i+1}, q_{i+1}\rangle \xrightarrow{\delta_{i+1},\iota_{i+1}} \langle \underset{\alpha_{i+2}}{\vdash\!\!-} R_{i+2}, q_{i+2}\rangle \xrightarrow{\delta_{i+2},\iota_{i+2}} \cdots$$

The thoughtful reader may notice that while our formulae are described over the union of the alphabet of the repository $\mathcal{R}$ and the alphabet of the activity $\underset{\alpha}{\vdash\!\!-} R$, the labels ι_i in a run belong to the alphabet A_{α_i}, that is the computed co-limit described in Definition 25. Therefore, we need to *translate* our formula accordingly with the modifications suffered by the activity during the particular run to be able to check if it holds. In order to define how the translation of the formula is carried out we rely on the result of Corollary 1. Two extra assumptions are needed: a) in the first place we assume that in a run r, $\iota_i \neq \varnothing$, b) in the second place we asssume that in a run a service module is not bound more than one time, meaning that we consider service modules in a repository as instances. Note that the assumption b) can be formalized by considering the repository as an extra element labelling the transitions in a run such that the repository remains unchanged along non-reconfiguration transitions and it is updated appropriately after reconfiguration transitions. The update of the repository is carried out in a way such that the services being bound in a reconfiguration step are removed from it. Then Definition 29 must be updated so the service modules considered in Condition 2 are only those that are available in the repository at that particular point in the execution.

Under the previous assumptions the following definition provides the required notation to define these translations:

Definition 34. *Let $\mathcal{R}$ be a repository and $\langle \underset{\alpha}{\vdash\!\!-} R, q\rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!-} R', q'\rangle$ a transition of $\mathcal{R}^\sharp$ then we define $A_{\hat{\delta}} : A_{\mathcal{R},\alpha} \to A_{\mathcal{R},\alpha'}$ as*

$$A_{\hat{\delta}}(a) = \begin{cases} A_\delta(a) & a \in A_\alpha \\ A_{\delta^{r_i}}(a) & a \in A_{\alpha^{r_i}} \\ a & \textit{otherwise} \end{cases}$$

Definition 35. *Let $\mathcal{R}$ be a repository and let $\underset{\alpha}{\vdash\!\!-} R$ be an activity. Also let $\mathcal{V} = A_{\mathcal{R},\alpha}$, $\phi, \psi \in LTLForm(\mathcal{V})$, $a \in \mathcal{V}$ and $v \subseteq \mathcal{V}$ then:*

- $\langle r, v, \tau\rangle \vDash \mathbf{true}$,
- $\langle r, v, \tau\rangle \vDash a$ *iff* $\tau(a) \in v$,

- $\langle r, v, \tau \rangle \vDash \neg\phi$ *iff* $\langle r, v, \tau \rangle \nvDash \phi$,
- $\langle r, v, \tau \rangle \vDash \phi \vee \psi$ *if* $\langle r, v, \tau \rangle \vDash \phi$ *or* $\langle r, v, \tau \rangle \vDash \psi$,
- $\langle r, v, \tau \rangle \vDash \mathbf{X}\phi$ *iff* $\langle r_1, \iota_0, \tau; A_{\hat{\delta}_0} \rangle \vDash \phi$, *and*
- $\langle r, v, \tau \rangle \vDash \phi \mathbf{U} \psi$ *iff there exists* $0 \leq i$ *such that* $\langle r_i, \iota_{i-1}, \tau; A_{\hat{\delta}_0}; ...; A_{\hat{\delta}_{i-1}} \rangle \vDash \psi$ *and for all* j, $0 \leq j < i$, $\langle r_j, \iota_{j-1}, \tau; A_{\hat{\delta}_0}; ...; A_{\hat{\delta}_{j-1}} \rangle \vDash \phi$ *where* $\iota_{-1} = \varnothing$ *and* $A_{\hat{\delta}_{-1}} = 1_{A_{\mathcal{R},\alpha}}$.

If $\mathcal{V}$ is a set of propositions, $\phi, \psi \in LTLForm(\mathcal{V})$, the rest of the boolean constants and operators are defined as usual as: $\mathbf{false} \equiv \neg\mathbf{true}$, $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\phi \implies \psi \equiv \neg\phi \vee \psi$, etc. We define $\Diamond\phi \equiv \mathbf{true}\mathbf{U}\phi$ and $\Box\phi \equiv \neg(\mathbf{true}\mathbf{U}\neg\phi)$.

Definition 36. *Let* $\mathcal{R}$ *be a repository and let*

$$r = \langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

be a successful open execution of $\mathcal{R}^\sharp$. *Then a formula* $\phi \in LTLForm(A_{\mathcal{R},\alpha_0})$ *is satisfied by* r $(r \vDash \phi)$ *if and only if* $\langle r, \varnothing, 1_{A_{\mathcal{R},\alpha_0}} \rangle \vDash \phi$.

Following the previous definitions, checking if an activity $\underset{\alpha}{\vdash\!\!-} R$ satisfies a proposition ϕ under a repository $\mathcal{R}$ is equivalent to checking if every successful open execution of $\underset{\alpha}{\vdash\!\!-} R$ with respect to $\mathcal{R}$ satisfies ϕ.

In the following we will show how the satisfaction relation in Definition 58 can be used to reason about properties of activities. We are particularly interested in asserting properties regarding the future execution of an activity with respect to a repository.

In order to exemplify, let us once again consider the activity TravelClient of Figure 2.3(a) and the repository $\mathcal{R}$ formed by the services TravelAgent, CurrenciesAgent, AccomodationAgent, and FlightsAgent described in Figure 2.4 and Figure 3.1. We are then interested in the open successful executions of the quasi-automaton $\mathcal{R}^\sharp$. Two examples of statements we could be interested in are the following properties:

1. Every execution of TravelClient requires the execution of CurrenciesAgent:

   For all successful open executions r of $\mathcal{R}^\sharp$,

$$r \vDash \Diamond \left(\bigvee_{a \in A_{M_{\mathsf{CurrenciesAgent}}}} a \right)$$

.

2. There exists an execution of TravelClient that does not require the execution of FlightsAgent:

   There exists a successful open execution r of $\mathcal{R}^\sharp$ such that

$$r \vDash \Box\left(\neg \bigvee_{a \in A_{M_{\mathsf{FlightsAgent}}}} a\right)$$

   .

The first property is true and it can be checked by observing that in the automaton Λ_{TA} no matter what is the choice for a transition made in the initial state ($bookHotels_{\text{¡}}$, $bookFlights_{\text{¡}}$, or $bookHotels\&Flights_{\text{¡}}$), the transition labelled with action $getExchangeRate!$ belongs to every path that returns to the initial state, that is the only accepting state. Therefore, the reconfiguration of the activity on port CE_0 is enforced in every successful execution.

The second one is also true as it states that there is an execution that does not requires the binding of a flights agent. Observing TravelClient, one can consider the trace in which no order on flights is placed never as the client always choose to order just accommodation.

3.2 Non monotonic reconfigurations

We argued that semantics given to ARNs in Section 3.1 provide a more realistic way of capturing SOC executions than previous versions, yet one should observe that reconfigurations of ARNs are considered only in a monotonic or incremental direction. This means that a reconfiguration always adds or binds a service component to an activity. From our perspective this is still not eniterly satisfactory for an execution model of services. One of the missing aspects of the semantics defined in Section 3.1 is the impossibility of considering non-incremental reconfigurations as the result of unbinding services. This situation may be the outcome of: a) the execution of a service whose goal is met in a finite number of steps, or b) the failure of a service to meet its goal.

Regarding the first situation, it is not uncommon for services to be atomic and stateless; and, going further in the conceptual bases of the paradigm, each binding resulting from a discovery triggered by the execution of an activity does not need to be served by the same service as it may not be available at that particular time, or it may not be the best suited for

the task. In general, the existing formalisations of services prescribe that services are reactive software artefacts, thus, once a service is bound, it remains bound and there is no notion of termination. To overcome this limitation we propose to model the computational aspects of terminating services as finite state automata such that they coexist in a coherent way with the non-terminating services whose computation is modelled by Muller automata.

The second situation emerges from the fact that the execution of communicating software systems, running over a falible communication infrastructure is likely to experience errors derived from the temporal unavailability of a given service bound to the executing software. SOC in the real world need to cope with this kind of errors as an absolute minimum fault tolerance criterion. Our proposal is to adapt the formal framework allowing the traces to suffer spontaneous "structure-decreasing" transitions from one state to another. These transitions are simple from the point of view of the transformation suffered by the network but, at the same time, they give rise to the question of to which state of the system it should transition to after the detection of the failure.

In this section we address these two scenarios by considering a modified version of ARNs that we call *Hybrid ARNs*. In this new version a new kind of process and a new kind of connections are introduced to model finite services. The difference between the new kinds of processes and connections and the old ones is that the new ones are labelled with finite state automata instead of Muller automata. Therefore they accept finite strings. In order to provide a uniform view for executions of services we encode this finite atuomata into Muller automata that accept the same language modulo an infinite ϵ-suffix. In this way we can extend definitions of ARNs with minimum changes to consider finite services.

3.2.1 Encoding FDAs into Muller Automata

Definition 37 (Finite State Automaton)**.** *A* finite state automaton Λ *is a tuple* $\langle Q, A, \Delta, q_0, \mathcal{F}\rangle$ *where*

- Q *is the set of* states *of* Λ*,*
- $\Delta \subseteq Q \times A \times Q$ *is the* transition relation *of* Λ*, with transitions* $(p, \iota, q) \in \Delta$ *usually denoted by* $p \xrightarrow{\iota} q$*,*
- $q_0 \in Q$ *is the* initial state *of* Λ*, and*

- $\mathcal{F} \subseteq Q$ *is the set of* final states *of* Λ.

Definition 38 (Equivalent Muller automaton)**.** *Given an FDA* $\Lambda = \langle Q, A, \Delta, q_0, \mathcal{F} \rangle$ *we define the* equivalent Muller automaton $\Lambda^\omega = \langle Q^\omega, A^\omega, \Delta^\omega, I^\omega, \mathcal{F}^\omega \rangle$ *such that:*

- $Q^\omega = Q \cup \{q_f\}$
- $A^\omega = A$
- $I^\omega = \{q_0\}$
- $\mathcal{F}^\omega = \{\{q_f\}\}$
- $\Delta^\omega = \Delta \cup \{(q, \epsilon, q_f) \mid q \in \mathcal{F}\} \cup \{(q_f, \epsilon, q_f)\}$

Then we can observe that the only strings accepted by Λ^ω are those that pass inifinite many times through the newly introduced sate q_f. The only output transition from this state is a self ϵ-loop and all the states that were marked as final in Λ have an ϵ-transition to q_f. Therefore the strings accepted by Λ^ω are the strings accepted by Λ plus an infinite ϵ suffix.

Definition 38 give us a way to encode FDA into Muller automata thus making them more easily and clearly handled in the rest of this work. Note that although in Definition 38 we consider a Muller automaton over an alphabet A in ARNs (and in HARNs too) we use Muller automata defined over the powerset of the alphabet, i.e. where the transitions are labelled with sets of tags. The enconding defined Definition 38 is trivially extended to that case by considering singletons as labels for transitions and the $\varnothing$-transition as a replacement for the ϵ-transition.

3.2.2 Semantics

Whereas in the previous semantics we had distinguished two types of hyperarchs, processes and connections, here we distinguish processes and connections even further in order to consider two different kinds of processes, those that represent services that *never end* (reactive processes) and another kind (processes) that we think is more suitable to model, for example, atomic/non-persistent services. Therefore here we rename Processes from Definition 18 to Reactive Process and we redefine Processes to be labeled with finite state automata.

Definition 39 (Reactive Process)**.** *A* reactive process $\langle \gamma, \Lambda \rangle$ *consists of a set* γ *of pairwise disjoint ports and a Muller automaton* Λ *over the set of actions* $A_\gamma = \bigcup_{M \in \gamma} A_M$*, where* $A_M = \{m! \mid m \in M^-\} \cup \{m¡ \mid m \in M^+\}$*.*

Definition 40 (Process)**.** *A* process $\langle \gamma, \Lambda \rangle$ *consists of a set* γ *of pairwise disjoint ports and a finite state automaton* Λ *over the set of actions* $A_\gamma = \bigcup_{M \in \gamma} A_M$*, where* $A_M = \{m! \mid m \in M^-\} \cup \{m¡ \mid m \in M^+\}$*.*

We follow the same modifications and notation with connections.

Definition 41 (Reactive Connection)**.** *Let* γ *be a set of pairwise disjoint ports. A* reactive connection $\langle M, \mu, \Lambda \rangle$ *between the ports of* γ *consists of a set* M *of messages, a partial attachment injection* $\mu_i \colon M \rightharpoonup M_i$ *for each port* $M_i \in \gamma$*, and a Muller automaton* Λ *over* $A_M = \{m! \bigm| m \in M\} \cup \{m¡ \bigm| m \in M\}$ *such that*

$$(a)\ \ M = \bigcup_{M_i \in \gamma} \mathrm{dom}(\mu_i) \quad \textit{and} \quad (b)\ \ \mu_i^{-1}(M_i^{\mp}) \subseteq \bigcup_{M_j \in \gamma \setminus \{M_i\}} \mu_j^{-1}(M_j^{\pm}).$$

Definition 42 (Connection)**.** *Let* γ *be a set of pairwise disjoint ports. A* connection $\langle M, \mu, \Lambda \rangle$ *between the ports of* γ *consists of a set* M *of messages, a partial attachment injection* $\mu_i \colon M \rightharpoonup M_i$ *for each port* $M_i \in \gamma$*, and a finite state automaton* Λ *over* $A_M = \{m! \bigm| m \in M\} \cup \{m¡ \bigm| m \in M\}$ *such that*

$$(a)\ \ M = \bigcup_{M_i \in \gamma} \mathrm{dom}(\mu_i) \quad \textit{and} \quad (b)\ \ \mu_i^{-1}(M_i^{\mp}) \subseteq \bigcup_{M_j \in \gamma \setminus \{M_i\}} \mu_j^{-1}(M_j^{\pm}).$$

Next, we redefine ARNs to what we call *Hybrid ARNs* in order to consider the existence of both kinds of processes and connections.

Definition 43 (Hybrid Asynchronous Relational Net)**.** *A* hybrid asynchronous relational net $\alpha = \langle X, P \cup RP, C \cup RC, \gamma, M, \mu, \Lambda \rangle$ *consists of*

- *a hypergraph* $\langle X, E \rangle$*, where* X *is a (finite) set of* points *and* $E = P \cup RP \cup C \cup RC$ *is a set of* hyperedges *(non-empty subsets of* X*) partitioned into* computation hyperedges $cp \in P \cup RP$ *and* communication hyperedges $c \in C \cup RC$ *such that no adjacent hyperedges belong to the same partition, and*

- *four labeling 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* $p \in P$*,* (c) *a reactive process* $\langle \gamma_{rp}, \Lambda_{rp} \rangle$ *to each hyperedge* $rp \in RP$*,* (d) *a connection* $\langle M_c, \mu_c, \Lambda_c \rangle$ *to each hyperedge* $c \in C$*, and* (e) *a reactive connection* $\langle M_{rc}, \mu_{rc}, \Lambda_{rc} \rangle$ *to each hyperedge* $rc \in RC$

We consider two special cases of HARNS that we call "pure". The first one when $C = P = \varnothing$ that we call "purely reactive HARN" and the second one when $RC = RP = \varnothing$ that we call "purely finite HARN". This two special cases will come in handy when we consider semantics of activites with respect to a repository.

The alphabet of a HARN is defined in a the same way as the alphabet of an ARN.

Definition 44 (Alphabet of a HARN)**.** *The* alphabet associated with an HARN α *is the vertex* A_α *of the colimit* $\xi\colon D_\alpha \Rightarrow A_\alpha$ *of the functor* $D_\alpha\colon \mathbb{J}_\alpha \to \mathbb{S}\mathrm{et}$*, where*

- $\mathbb{J}_\alpha$ *is the preordered category whose objects are points* $x \in X$*, hyperedges* $e \in P \cup RP \cup C \cup RC$*, or "attachments"* $\langle c, x\rangle$ *of* α*, with* $c \in C \cup RC$ *and* $x \in \gamma_c$*, and whose arrows are given by* $\{x \to p \mid p \in P \cup RP, x \in \gamma_p\}$*, for computation hyperedges, and* $\{c \leftarrow \langle c, x\rangle \to x \mid c \in C \cup RC, x \in \gamma_c\}$*, for communication hyperedges;*

- D_α *defines the sets of actions associated with the ports, processes and channels, together with the appropriate mappings between them.*

Definition 45 (Automaton of a HARN)**.** *Given a* hybrid asynchronous relational net $\alpha = \langle X, P \cup RP, C \cup RC, \gamma, M, \mu, \Lambda\rangle$ *we define the automaton of* α *as a Muller automaton* $\Lambda_\alpha = \langle Q_\alpha, A_\alpha, \Delta_\alpha, I_\alpha, \mathcal{F}_\alpha\rangle$ *such that:*

$$
\begin{aligned}
Q_\alpha &= \textstyle\prod_{e\in RP\cup RC} Q_e \times \prod_{e\in P\cup C} Q_e^\omega \\
\Delta_\alpha &= \{(p,\iota,q) \;\big|\; (\pi_e(p), \xi_e^{-1}(\iota), \pi_e(q)) \in \Delta_e \text{ for each } e \in RP \cup RC \\
&\qquad \text{and } (\pi_e(p), \xi_e^{-1}(\iota), \pi_e(q)) \in \Delta_e^\omega \text{ for each } e \in P \cup C\}, \\
I_\alpha &= \{(q_e)_{e\in P\cup RP\cup C\cup RC} \mid q_e \in I_e \text{ if } e \in RP \cup RC \text{ and } q_e = q_{0_e} \text{ if } e \in P \cup C\} \\
\mathcal{F}_\alpha &= \{qs \subseteq Q_\alpha \mid \forall e \in RP \cup RC, \pi_e(qs) \in \mathcal{F}_e \text{ and } \forall e \in P \cup C, \pi_e(qs) \in \mathcal{F}_e^\omega\}
\end{aligned}
$$

According to Definition 45 the automaton of the HARN formed by a process p is not the automaton Λ_p that labels the hyperarch p but the equivalent Muller automaton Λ_p^ω. Note that because of how the set $\mathcal{F}_\alpha$ is defined, the finite components are required to remain in their final state. This is sound since the equivalent Muller automata, by definition, have idle final states.

Fact 2. *Under the notations of Definition 45, for every hyperedge* $e \in RP \cup RC$ *of* α*, the maps* ξ_e *and* π_e *define an* $\mathbb{MA}$*-morphism* $\langle \xi_e, \pi_e\rangle\colon \langle A_{M_e}, \Lambda_e\rangle \to \langle A_\alpha, \Lambda_\alpha\rangle$*, and for every hyperedge* $e \in P \cup C$ *of* α*, the maps* ξ_e *and* π_e *define an* $\mathbb{MA}$*-morphism* $\langle \xi_e, \pi_e\rangle\colon \langle A_{M_e}, \Lambda_e^\omega\rangle \to \langle A_\alpha, \Lambda_\alpha\rangle$

Proposition 9. *For every HARN* $\alpha = \langle X, P \cup RP, C \cup RC, \gamma, M, \mu, \Lambda \rangle$*, the* $\mathbb{MA}$*-morphisms* $\langle \xi_e, \pi_e \rangle : \langle A_{M_e}, \Lambda_e \rangle \to \langle A_\alpha, \Lambda_\alpha \rangle$ *associated with hyperedges* $e \in P \cup RP \cup C \cup RC$ *form colimit injections for the functor* $G_\alpha : \mathbb{J}_\alpha \to \mathbb{MA}$ *that maps*

- *every computation or communication hyperedge* $e \in P \cup RP \cup C \cup RC$ *to* $\langle A_{M_e}, \Lambda_e \rangle$ *and*
- *every point* $x \in X$ *(or attachment* $\langle c, x \rangle$*) to* $\langle A_{M_x}, \Lambda_x \rangle$*, where* Λ_x *is the Muller automaton* $\langle \{q\}, 2^{A_{M_x}}, \{(q, \iota, q) \mid \iota \subseteq A_{M_x}\}, \{q\}, \{\{q\}\} \rangle$ *with only one state, which is both initial and final, and with all possible transitions.*

Therefore, both the alphabet and the automaton of an HARN α *are given by the vertex* $\langle A_\alpha, \Lambda_\alpha \rangle$ *of a colimiting cocone of the functor* $G_\alpha : \mathbb{J}_\alpha \to \mathbb{MA}$*.*

Proof. As in Proposition 5. □

Definition 46 (Morphism of HARNs)**.** *A* morphism $\delta : \alpha \to \alpha'$ *between two HARNs* $\alpha = \langle X, P \cup RP, C \cup RC, \gamma, M, \mu, \Lambda \rangle$ *and* $\alpha' = \langle X', P' \cup RP', C' \cup RC', \gamma', M', \mu', \Lambda' \rangle$ *consists of*

- *an injective map* $\delta : X \to X'$ *such that* $\delta(P) \subseteq P'$*,* $\delta(RP) \subseteq RP'$*,* $\delta(C) \subseteq C'$ *and* $\delta(RC) \subseteq RC'$*, that is an injective homomorphism between the underlying hypergraphs of* α *and* α' *that preserves the computation and communication hyperedges, and*
- *a family of polarity-preserving injections* $\delta^{pt}_x : M_x \to M'_{\delta(x)}$*, for* $x \in X$*,*

such that

- *for every computation hyperedge* $p \in P \cup RP$*, for every point* $x \in \gamma_p$*,* $\delta^{pt}_x = 1_{M_x}$*,*
- *for every computation hyperedge* $p \in P \cup RP$*,* $\Lambda_p = \Lambda'_{\delta(p)}$*, and*
- *for every communication hyperedge* $c \in C \cup RC$*,* $M_c = M'_{\delta(c)}$*,* $\Lambda_c = \Lambda'_{\delta(c)}$ *and, for every point* $x \in \gamma_c$*,* $\mu_{c,x}; \delta^{pt}_x = \mu'_{\delta(c),\delta(x)}$*.*

Proposition 10. *The class of HARNS together with the morphisms in Definition 46 form a category that we denote by* $\mathbb{HARN}$*.*

Proof. It is straightforward to verify that the morphisms of HARNs can be composed in terms of their components. Their composition is associative and has left and right identities given by morphisms that consist solely of set-theoretic identities. □

Definition 47. *The "flattened" automaton* $\mathcal{A}^\sharp = \langle Q^\sharp, A^\sharp, \Delta^\sharp, I^\sharp, \mathcal{F}^\sharp \rangle$ *induced by the functor* $\mathcal{A}\colon \mathbb{HARN} \to \mathbb{MA}$[4] *is defined as follows:*

$$
\begin{aligned}
Q^\sharp =& \{\langle \alpha, q\rangle \mid \alpha \in |\mathbb{HARN}| \text{ and } q \in Q_\alpha\},\\
A^\sharp =& \{\langle \delta, \iota\rangle \mid \delta\colon \alpha \to \alpha' \text{ or } \delta\colon \alpha' \to \alpha \text{ and } \iota \subseteq A_\alpha\},\\
\Delta^\sharp =& \{(\langle \alpha, q\rangle, \langle \delta, \iota\rangle, \langle \alpha', q'\rangle) \mid \delta = 1_\alpha \text{ or } \delta\colon \alpha \to \alpha' \text{ and } (q, \iota, q'\restriction_\delta) \in \Delta_\alpha\\
& \quad \text{or } \delta\colon \alpha' \to \alpha \text{ and } (q, \iota, q'') \in \Delta_\alpha \text{ and } q''\restriction_\delta = q'\},\\
I^\sharp =& \{\langle \alpha, q\rangle \mid \alpha \in |\mathbb{HARN}| \text{ and } q \in I_\alpha\}, \text{ and}\\
\mathcal{F}^\sharp =& \{\{\langle \alpha, q\rangle \mid q \in F\} \mid \alpha \in |\mathbb{HARN}| \text{ and } F \in \mathcal{F}_\alpha\}.
\end{aligned}
$$

The "flattened" automaton of Definition 47 amalgamates both the configuration and the state of the system in a similar way as Definition 27 does for ARNs. The main difference is that here we need to take into account structural modifications that can now remove parts of the structure. Is due to this aspect that here we consider morphisms in $\mathbb{HARN}$ in both directions, i.e. going from the source HARN to the target HARN and the other way around. Notice that in Definition 47 we do not restrict the structural changes in any way. This allows us to capture every possible reconfiguration but on the the other hand this gives place to new situations that need to be considered.

First, some of these reconfigurations may be considered valid and some other invalid. For example, a reconfiguration that removes a process hyperarch p would be considered valid if p had arrived to a final state of its underlying FDA, while it would be considered invalid otherwise. This distinction is interesting from a conceptual point of view as it allows us to distinguish *termination* from *failures*. These two notions will be formalized afterwards in this document.

Second, in Definition 47 there is no restriction to the kind of reconfigurations that may occur. Therefore, when considering executions of a HARN there are no warranties that the initial HARN (that of the initial state of the execution) will be kept along the run. This constitutes a problem from the point of view of characterizing executions of activities. Thus we need to restrict transitions even further in order to get a proposition equivalent to Proposition 6 but for HARNs.

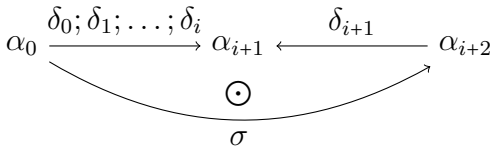
In order to achieve this we will require an extra property in the case of remotions. Note that the path up to the first remotion in an execution of $\mathcal{A}^\sharp$ is a sequence of morphism (and labels) that can be regarded as one (the

[4]Note that $\Lambda^\sharp$ is in fact a quasi-automaton, because its components are proper classes.

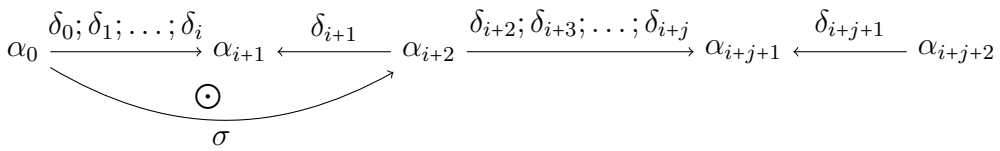
composotion of all of them) yielding a cospan like the following:

$$\alpha_0 \xrightarrow{\delta_0;\delta_1;\ldots;\delta_i} \alpha_{i+1} \xleftarrow{\delta_{i+1}} \alpha_{i+2}$$

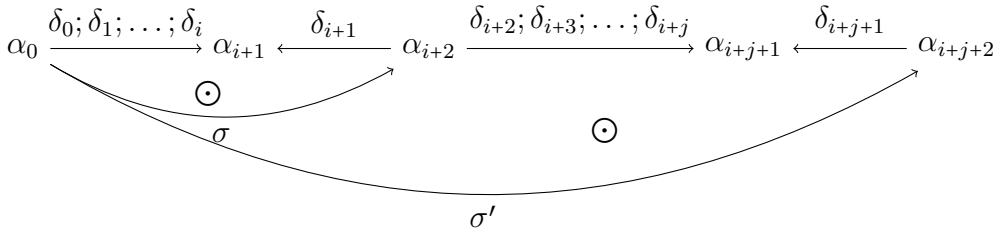
. We want to characterize traces in which α_0 is kept along the run. We can do that by requiring that in a situation like this one there is a morphism $\sigma : \alpha_0 \to \alpha_{i+2}$ such that $\sigma;\delta_{i+1} = \delta_0;\delta_1;\ldots;\delta_i$, i.e. the following diagram commutes:



If the above property is valid for the first remotion then one can enunciate a similar property for the second remotion noting that the second remotion has the following form:



Then one could ask that a $\sigma' : \alpha_0 \to \alpha_{i+j+2}$ exists such that $\sigma;\delta_{i+2};\delta_{i+3};\ldots;\delta_{i+j} = \sigma';\delta_{i+j+1}$, i.e. the following diagram commutes:



Note that if the above property holds for all the remotions up to the i^{th}-one then by conmmutativity of the diagram we can guarantee that α_0 remained after each of this remotions.

Definition 48 (Open execution of a HARN)**.** *An* open execution *of a HARN α is an execution of $\mathcal{A}^\sharp$ that starts from an initial state of Λ_α, i.e. a sequence*

$$\langle \alpha_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \alpha_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \alpha_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

such that $\alpha_0 = \alpha$, $q_0 \in I_\alpha$ and, for every $i \in \omega$

a) $\langle \alpha_i, q_i \rangle \xrightarrow{\langle \delta_i, \iota_i \rangle} \langle \alpha_{i+1}, q_{i+1} \rangle$ *is a transition in* $\Delta^\sharp$ *and*

b) for all $i \geq 0$ *there exists* $\sigma_i : \alpha_0 \to \alpha_i$ *such that:*

- $\sigma_{i-1}; \delta_{i-1} = \sigma_i$ *if* $\delta_{i-1} : \alpha_{i-1} \to \alpha_i$ *or*
- $\sigma_{i-1} = \sigma_i; \delta_{i-1}$ *if* $\delta_{i-1} : \alpha_i \to \alpha_{i-1}$

where $\sigma_{-1} = \delta_{-1} = 1_{\alpha_0}$.

An open execution as above is successful *if it is successful with respect to the automaton* $\mathcal{A}^\sharp$.

Condition b) of Definition 48 guarantees that α_0 is preserved along the execution. As a consequence of this property, we can take the reduct via σ_i at any i^{th} step of the execution to obtain a state/label of Λ_{α_0}. With this in hand we can now again assure that a successful open execution of a HARN α gives a successful execution of the automaton of α. Again we base our reasoning on the definition of the transitions of $\mathcal{A}^\sharp$ and on the functoriality of $\mathcal{A}{:}\mathbb{HARN} \to \mathbb{MA}$. Then, it is easy to see that, for every HARN α, every successful open execution of α gives a successful execution of its underlying automaton Λ_α.

Proposition 11. *For every (successful) open execution*

$$\langle \alpha_0, q_0 \rangle \xrightarrow{\delta_0, \iota_0} \langle \alpha_1, q_1 \rangle \xrightarrow{\delta_1, \iota_1} \langle \alpha_2, q_2 \rangle \xrightarrow{\delta_2, \iota_2} \cdots$$

of the quasi-automaton $\mathcal{A}^\sharp$*, the infinite sequence*

$$q_0 \xrightarrow{(\iota_0)} q_1 \restriction_{\sigma_1} \xrightarrow{A^{-1}_{\sigma_1}(\iota_1)} q_2 \restriction_{\sigma_2} \xrightarrow{A^{-1}_{\sigma_2}(\iota_2)} \cdots$$

corresponds to a (successful) execution of the automaton Λ_{α_0}.

Proof. By resorting to the same reasoning used to prove Proposition 6. Note that the only differences are that now we take the σ_i-reducts at the i^{th} step and, contrary to what was prescribed for ARNs, the structure does not need to remain fixed from certain step on (i.e. it can change infinitely). In ARNs this requirement was added because allowing for infinite reconfigurations could yield to an empty set of infinitely often visited states. Notice that in HARNs infinite reconfigurations can still yield a non-empty set of infinitely often visited states and therefore are allowed. □

3.2.3 Open execution of hybrid activities

We first mimic the definition of *activites* using HARNs instead of ARNs and we call them simply *hybrid activities*. For the rest of this section we assume that $\mathcal{R}$ is an arbitrary but fixed repository of service modules. Note that in this section we only consider as a valid remotion reconfiguration those that occur due to termination. That means that we'll restrict these reconfigurations in a way that only transitions that remove processes when they've reached a final state will be considered valid. Other kinds of remotion, such as failures, will be discussed in the next section. Additionaly we require that every service module is defined by a purely finite or purely reactive HARN. This is required to keep semantics consistent since termination is considered in a service-wide manner, meaning that a service module has to be removed entirely or not at all.

Definition 49. *The* hybrid activity (quasi-)automaton $\mathcal{R}^\sharp = \langle Q^{\mathcal{R}}, A^{\mathcal{R}}, \Delta^{\mathcal{R}}, I^{\mathcal{R}}, \mathcal{F}^{\mathcal{R}} \rangle$ *generated by the service repository* $\mathcal{R}$ *is defined as follows:*

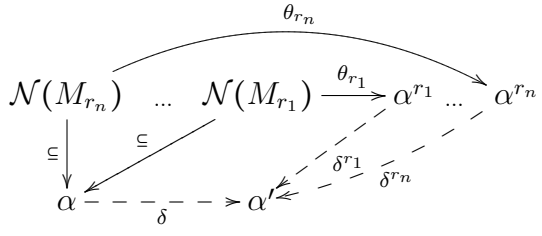
The states *in* $Q^{\mathcal{R}}$ *are pairs* $\langle \underset{\alpha}{\vdash\!\!\!-} R, q \rangle$*, where* $\underset{\alpha}{\vdash\!\!\!-} R$ *is a hybrid activity – i.e.* α *is a HARN and* R *is a finite set of requires-points of* α *– and* q *is a state of* Λ_α*.*

The alphabet $A^{\mathcal{R}}$ *is given by pairs* $\langle \delta, \iota \rangle$*, where* $\delta \colon \alpha \to \alpha'$ *is a morphism of HARNs and* ι *is a set of* α*-actions; thus,* $A^{\mathcal{R}}$ *is just the alphabet of* $\mathcal{A}^\sharp$*.*

There exists a transition $\langle \underset{\alpha}{\vdash\!\!\!-} R, q \rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!\!-} R', q' \rangle$ *whenever:*

1. $\langle \alpha, q \rangle \xrightarrow{\delta,\iota} \langle \alpha', q' \rangle$ *is a transition of* $\mathcal{A}^\sharp$*;*
2. *if* $\delta : \alpha \to \alpha'$ *then, for each requires-point* $r \in R$ *such that* $\xi_r(A_{M_r^+}) \cap \iota \neq \varnothing$ *there exists*
   - *a service module* $P^r \underset{\alpha^r}{\longleftarrow} R^r$ *in* $\mathcal{R}$ *and*
   - *a polarity-preserving injection* $\theta_r : M_r \to M_{P^r}$

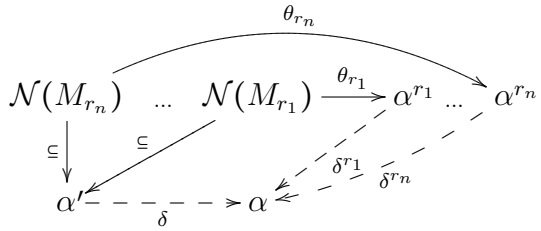
   such that the following colimit can be defined in the category of HARNs



where $\{r_1, \ldots, r_n\}$ *is the biggest subset of* R *such that* $\xi_{r_i}(A_{M^+_{r_i}}) \cap \iota \neq \varnothing$ *for all* $1 \leq i \leq n$ *and* $\mathcal{N}(M_{r_i})$ *is the atomic HARN that consists of only one point, labelled with the port* M_{r_i}*, and no hyperedges;*

3. *there exists a transition* $p' \xrightarrow{\iota'} q'$ *of* $\Lambda_{\alpha'}$ *such that* $p'\restriction_\delta = q$*,* $A_\delta^{-1}(\iota') = \iota$ *and, for each requires-point* $r \in R$ *as above,* $p'\restriction_{\delta^r}$ *is an initial state of* Λ_{α^r}*;*
4. *if* $\delta : \alpha' \to \alpha$ *then, for each requires-point* $r \in R'$ *and* $r \notin R$ *there exists*
   - *a service module* $P^r \xleftarrow[\alpha^r]{} R^r$ *in* $\mathcal{R}$ *and*
   - *a polarity-preserving injection* $\theta_r : M_r \to M_{P^r}$

   such that the following colimit can be defined in the category of HARNs



   where $\{r_1, \ldots, r_n\}$ *is the biggest subset of* R *such that* $r_i \in R'$ *and* $r_i \notin R$ *for all* $1 \leq i \leq n$*;*
5. *there exists a transition* $q \xrightarrow{\iota} p$ *of* Λ_α *such that* $p\restriction_\delta = q'$ *and, for each requires-point* $r \in R$ *as above,* $p\restriction_{\delta^r}$ *is a final state of the FDA* Λ_{α^r}*;*

The states *in* $I^{\mathcal{R}}$ *are those pairs* $\langle \xrightarrow[\alpha]{} R, q \rangle$ *for which* $q \in I_\alpha$*.*

The final-state sets *in* $\mathcal{F}^{\mathcal{R}}$ *are those sets* $\{\langle \xrightarrow[\alpha]{} R, q\rangle \mid q \in F\}$ *for which* $F \in \mathcal{F}_\alpha$*.*

Open executions of hybrid activities can be defined relative to the automaton $\mathcal{R}^\sharp$ in a similar way to the open executions of HARNs (see Definition 48).

Definition 50 (Open execution of a hybrid activity)**.** *An* open execution *of a hybrid activity* $\xrightarrow[\alpha]{} R$ *with respect to* $\mathcal{R}$ *is an execution of the quasi-automaton* $\mathcal{R}^\sharp$ *that starts from an initial state of* Λ_α*, i.e. a sequence of transitions of* $\mathcal{R}^\sharp$

$$\langle \xrightarrow[\alpha_0]{} R_0, q_0\rangle \xrightarrow{\delta_0,\iota_0} \langle \xrightarrow[\alpha_1]{} R_1, q_1\rangle \xrightarrow{\delta_1,\iota_1} \langle \xrightarrow[\alpha_2]{} R_2, q_2\rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

such that $\alpha_0 = \alpha$*,* $R_0 = R$*,* $q_0 \in I_\alpha$*, and, for every* $i \in \omega$

a) $\langle \vdash\!\!\!\underset{\alpha_i}{\longrightarrow} R_i, q_i \rangle \xrightarrow{\delta_i, \iota_i} \langle \vdash\!\!\!\underset{\alpha_{i+1}}{\longrightarrow} R_{i+1}, q_{i+1} \rangle$ *is a transition in* $\mathcal{R}^\sharp$ *and*

b) for all $i \geq 0$ *there exists* $\sigma_i : \alpha_0 \to \alpha_i$ *such that:*

- $\sigma_{i-1}; \delta_{i-1} = \sigma_i$ *if* $\delta_{i-1} : \alpha_{i-1} \to \alpha_i$ *or*
- $\sigma_{i-1} = \sigma_i; \delta_{i-1}$ *if* $\delta_{i-1} : \alpha_i \to \alpha_{i-1}$

where $\sigma_{-1} = \delta_{-1} = 1_{\alpha_0}$.

. An open execution as above is successful *if it is successful with respect to* $\mathcal{R}^\sharp$.

3.2.4 Aborting communications

In Section 3.2.3 we characterized executions of activities which made use of terminating service modules. There we allowed reconfigurations that removed service modules when these modules arrived to a final state. But there is another possible cause for modules' removal, failures. The nature of a failure is diverse. For example, there are recoverable failures such as temporary unavailability or rejection of an operation due to data validation. This kind of failure may be handled by an activity inside its logic. What we are trying to characterize here in our semantics are unrecoverable failues. An unrecoverable failure may be a consequence of a hardware failure, such as a service module becoming unreachable due to a network or server downtime, or an unexpected critical software error. When a failure such as the latter occur some previously bound service modules could be made unavailable in a sudden and unexpected time. Thus, to capture this kind of situation in our semantics we need to give a step backwards from the semantics given in Section 3.2.3 in order to allow reconfigurations that remove parts of a HARN at any point of the execution. Furthermore we want to give a property that allows us to describe a subset of the executions of an activity that may still constitute a successful execution even though it may have suffered some failures along the way.

In order to be able to define a formal semantics for aborting communications or failures we'll need to rely on the concept of *binding point* of a service module. Intuitively *the* binding point of a service module $P^r \underset{\alpha^r}{\longleftarrow} R^r$ with respect to a requires point of an activity's execution is the state in the execution immediately before the binding of $P^r \underset{\alpha^r}{\longleftarrow} R^r$. From the point of view of the actions, the binding point is the state immediately before the execution of a trigger associated with the requires point where $P^r \underset{\alpha^r}{\longleftarrow} R^r$ was bound to.

Definition 51. *In an open execution*

$$\langle \underset{\alpha_0}{\vdash\!\!-\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

of a hybrid activity $\underset{\alpha_0}{\vdash\!\!-\!\!-} R_0$*, a binding point of a service module* $P \underset{\alpha}{\leftarrow} R$ *in* $\mathcal{R}$ *with respect to a requires point* r *is a state*

$$\langle \underset{\alpha_i}{\vdash\!\!-\!\!-} R_i, q_i \rangle$$

such that:

a) $r \in R_i$ *and* $r \notin R_{i+1}$

b) $\delta^r : \alpha \to \alpha_{i+1}$ *where* δ^r *is one of the morphisms of the colimit in Condition 2 of Definition 49.*

Note that in an open execution of a hybrid activity there may be more than one binding point for a service module with respect to a requires point. Even more, the binding points for a given service module and a given requires point may be infinite. Nevertheless we are interested in looking at the binding points of a service module with respect to a requires point at a particular point in the execution of the activity. That means that we are going to look at the binding points in a prefix of an execution, and prefixes are finite. Therefore we are going to be looking at a finite amount of binding points at a time and thus we can define the nearest binding point which is the binding point

$$\langle \underset{\alpha_i}{\vdash\!\!-\!\!-} R_i, q_i \rangle$$

such that $i \geq j$ for all j such that

$$\langle \underset{\alpha_j}{\vdash\!\!-\!\!-} R_j, q_j \rangle$$

is a binding point in the prefix we are looking at.

Next we define a rollback state. When a failure occurs one way to safely continue executing is to rollback. Rolling back is a concept inherited from the transactional world such as that of database systems.

Definition 52. *In an open execution*

$$\langle \underset{\alpha_0}{\vdash\!\!-\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

of a hybrid activity $\underset{\alpha_0}{\vdash\!\!-\!\!-} R_0$ *a state*

$$\langle \underset{\alpha_{i+1}}{\vdash\!\!-\!\!-} R_{i+1}, q_{i+1} \rangle$$

is a rollback state *if*

a) $\delta_i : \alpha_{i+1} \to \alpha_i$ *and there exists* $0 \leq j < i+1$ *such that*

$$\langle \vdash\!\!\!\underset{\alpha_j}{\!\!\!\!\!\!-\!\!\!\!\!\!-} R_j, q_j \rangle \equiv \langle \vdash\!\!\!\underset{\alpha_{i+1}}{\!\!\!\!\!\!-\!\!\!\!\!\!-} R_{i+1}, q_{i+1} \rangle$$

and

b) for all $r \in R_{i+1}$ *and* $r \notin R_i, j \leq b_r$ *where* $\langle \vdash\!\!\!\underset{\alpha_{b_r}}{\!\!\!\!\!\!-\!\!\!\!\!\!-} R_{b_r}, q_{b_r} \rangle$ *is the nearest binding point of the service module* $P_r \xleftarrow[\alpha_r]{} R_r$ *with respect to the requires point* r*.*

Condition a) of Definition 52 prescribes that a rollback state is the result of a remotion reconfiguration. This implies that α_{i+1} is smaller (in terms of HARN morphisms) than α_i. Condition b) says that a rollback state is in essence any state that occurred previously to the binding of every service module that was removed due to the failure in this execution. Notice that there is always a state that complies with these conditions which is the initial state of the execution.

Note that this approach is a conservative one. We rollback to a safe state in the sense that we "undo" everything that was done from the binding of the component that failed up to the failure point in the trace. This of course is suboptimal since between the binding of a service and its failure many tasks could have been done successfuly. At this point we envision the possibility of considering serveral different rollback policies. For example in WS-BPEL [52] compensations are defined. Compensations are explicit "undo" sequences of operations that take place in the event of an error. This could be incorporated into our model although explicit support for compensations should be added. Other ways of defininig rollback policies could be by different equivalence criterions between traces. For example one could consider as a valid rollback state any state whose continuations are included into the successful continuations of the trace that failed. Implementing such criterions would generally imply computationally expensive checks.

With this in hand we can revisit Definition 49 in order to incorporate the notion of rollback to the execution of an activity.

Definition 53 (Hybrid activity (quasi-)automaton revisited)**.** *The* hybrid activity (quasi-)automaton $\mathcal{R}^\sharp = \langle Q^{\mathcal{R}}, A^{\mathcal{R}}, \Delta^{\mathcal{R}}, I^{\mathcal{R}}, \mathcal{F}^{\mathcal{R}} \rangle$ *generated by the service repository* $\mathcal{R}$ *is defined as follows:*

The states *in* $Q^{\mathcal{R}}$ *are pairs* $\langle \vdash\!\!\!\underset{\alpha}{\!\!\!\!\!\!-} R, q \rangle$*, where* $\vdash\!\!\!\underset{\alpha}{\!\!\!\!\!\!-} R$ *is a hybrid activity – i.e.* α *is a HARN and* R *is a finite set of requires-points of* α *– and* q *is a state of* Λ_α*.*

The alphabet $A^{\mathcal{R}}$ *is given by pairs* $\langle \delta, \iota \rangle$*, where* $\delta : \alpha \to \alpha'$ *is a morphism of HARNs and* ι *is a set of* α*-actions; thus,* $A^{\mathcal{R}}$ *is just the alphabet of* $\mathcal{A}^\sharp$*.*

There exists a transition $\langle \underset{\alpha}{\vdash\!\!-} R, q \rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!-} R', q' \rangle$ *whenever Conditions 1, 2 and 3 of Definition 49 hold and if* $\delta : \alpha' \to \alpha$ *then either Conditions 4 and 5 hold or* $\iota = \varnothing$ *and* $\langle \underset{\alpha'}{\vdash\!\!-} R', q' \rangle$ *is a rollback state.*

Definition 54 (Open execution of a hybrid activity with failures)**.** *An* open execution *of a hybrid activity* $\underset{\alpha}{\vdash\!\!-} R$ *with respect to* $\mathcal{R}$ *is an execution of the quasi-automaton* $\mathcal{R}^\sharp$ *of Definition 53 that starts from an initial state of* Λ_α*, i.e. a sequence of transitions of* $\mathcal{R}^\sharp$

$$\langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

such that $\alpha_0 = \alpha$*,* $R_0 = R$*,* $q_0 \in I_\alpha$ *for every* $i \in \omega$

a) $\langle \underset{\alpha_i}{\vdash\!\!-} R_i, q_i \rangle \xrightarrow{\delta_i,\iota_i} \langle \underset{\alpha_{i+1}}{\vdash\!\!-} R_{i+1}, q_{i+1} \rangle$ *is a transition in* $\mathcal{R}^\sharp$ *and*

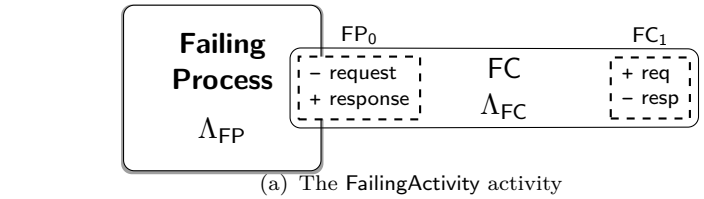
b) for all $i \geq 0$ *there exists* $\sigma_i : \alpha_0 \to \alpha_i$ *such that:*

- $\sigma_{i-1}; \delta_{i-1} = \sigma_i$ *if* $\delta_{i-1} : \alpha_{i-1} \to \alpha_i$ *or*
- $\sigma_{i-1} = \sigma_i; \delta_{i-1}$ *if* $\delta_{i-1} : \alpha_i \to \alpha_{i-1}$

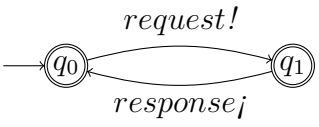
where $\sigma_{-1} = \delta_{-1} = 1_{\alpha_0}$*.*

An open execution as above is successful *if it is successful with respect to* $\mathcal{R}^\sharp$*.*

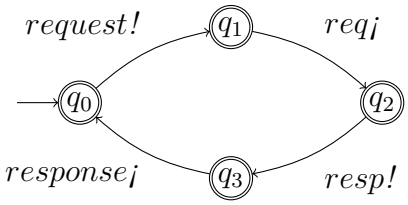
According to Definition 53 and Definition 54 an execution that fails infinitely often may be considered successful, depending on exactly how the automata of the HARN were defined. This is certainly not a desireable property thus in the next section we give a tool that allow us to restrict executions even further and will allow us to get rid of such undesireable executions throughout the use of LTL sentences. An example of this situation can be seen in Figure 3.3. There a simple activity is depicted, it makes a request throughout a connection that simply forwards the request and awaits the response to, once again, forward it to the original requestor. In this example $\mathcal{F}^{\Lambda_{\mathsf{FP}}} = \{\{q_0, q_1\}\}$ and $\mathcal{F}^{\Lambda_{\mathsf{FC}}} = \{\{q_0, q_1, q_2, q_3\}\}$ meaning that in order for an execution of this activity to be successful it has to visit every state infnitely many times. Note that even so, an execution that fails infinitely often after reaching the automaton state (q_1, q_3) rolling back to the initial state would still be considered successful in terms of the flattened automaton $\mathcal{R}^\sharp$ since the set of infinitely often visited states would be $\{(q_0, q_0), (q_1, q_1), (q_1, q_2), (q_1, q_3)\}$.



(a) The FailingActivity activity



(b) The automaton Λ_{FP}



(c) The automaton Λ_{FC}

Figure 3.3: Failing activity with successful executions

3.2.5 Satisfiability of LTL Formulae in HARNS

In Section 3.1.5 we showed how the trace semantics given to ARNs could be used to reason about temporal properties. Here we adapt definitions in Section 3.1.5 in order to consider the differences between ARNs and HARNs. The main difference, from a formal point of view, is that now the execution does not yield a signature morphism directly since a step in the execution can be a remotion. Therefore a new way of translating predicate names must be devised. To this end we now consider sets of propositions (labels). Towards the end of the section we show how this LTL formulae can be used to characterize a subset of the successful executions of the example in Figure 3.3 such that infinitely often failing executions are excluded.

As before in order to check the satisfaction of propositions in a particular state of a run we need to translate propositions, that are elements of the signature of the repository or the signature of the activity, to the signature of the HARN in consideration at a particular state.

Note 1. *The reader should note that there are not many requirements over the structure of ARNs or HARNs. In particular we do not require for them to be connected. This permits some unreasonable situations. For example, an activity or a service module could be described by a non-connected ARN or HARN. This situation does not seem realistic but still we choose not to restrict the formalism any further and to admit these awkward configurations.*

This setting suppose an additional complexity for considering appropri-

ate translations of propositions along the execution of a HARN. In the case of ARNs as the execution evolves solely by, at most, adding new service modules to the activity, a proposition p can always be translated along by composing the morphisms used to reconfigure the system and sice morphisms are injections we ended up always with a new proposition p' in the signature of the resulting ARN. When considering executions of HARNs the situation is slightly different because in the event of a remotion, considering the translation of a proposition "backwards" (i.e. looking at a morphism in the opposite direction) can lead from a proposition p to a set of propositions $\{p_1, \ldots, p_n\}$ that were collapsed together by means of the morphism we are looking and the injective mappings in the connections.

Notice that if we add, as an extra requirement, that the structure is connected, then as we always add/remove service modules as a whole during executions of activities we could guarantee that, even in the event of a remotion, the translation of a proposition would yield a single proposition. In the following definitions we consider the formalization as it is now and we give a satsifaction criterion that deals with the fact that translating a proposition can yield a set of propositions.

Definition 55. *Let $\mathcal{R}$ be a repository and $\langle \underset{\alpha}{\vdash\!\!-} R, q\rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!-} R', q'\rangle$ a transition of $\mathcal{R}^\sharp$ such that $\delta : \alpha \to \alpha'$ then we define $A_{\hat{\delta}} : A_{\mathcal{R},\alpha} \to A_{\mathcal{R},\alpha'}$ as*

$$A_{\hat{\delta}}(a) = \begin{cases} A_\delta(a) & a \in A_\alpha \\ A_{\delta^{r_i}}(a) & a \in A_{\alpha^{r_i}} \\ a & otherwise \end{cases}$$

Definition 56. *Let $\mathcal{R}$ be a repository and $\langle \underset{\alpha}{\vdash\!\!-} R, q\rangle \xrightarrow{\delta,\iota} \langle \underset{\alpha'}{\vdash\!\!-} R', q'\rangle$ a transition of $\mathcal{R}^\sharp$ then we define $A_{\hat{\Delta}} : 2^{A_{\mathcal{R},\alpha}} \to 2^{A_{\mathcal{R},\alpha'}}$ as*

$$A_{\hat{\Delta}}(ACT) = \begin{cases} \{A_{\hat{\delta}}(a) \mid a \in ACT\} & \delta : \alpha \to \alpha' \\ \bigcup_{a \in ACT} a \restriction A_{\hat{\delta}} & \delta : \alpha' \to \alpha \end{cases}$$

Definition 57. *Let $\mathcal{R}$ be a repository and let $\underset{\alpha}{\vdash\!\!-} R$ be an activity. Also let $\mathcal{V} = A_{\mathcal{R},\alpha}$, $\phi, \psi \in LTLForm(\mathcal{V})$, $a \in \mathcal{V}$ and $v \subseteq \mathcal{V}$ then:*

- $\langle r, v, \tau\rangle \vDash \mathbf{true}$,
- $\langle r, v, \tau\rangle \vDash a$ *iff* $\tau(\{a\}) \subseteq v$,
- $\langle r, v, \tau\rangle \vDash \neg\phi$ *iff* $\langle r, v, \tau\rangle \nvDash \phi$,

- $\langle r, v, \tau \rangle \vDash \phi \vee \psi$ *if* $\langle r, v, \tau \rangle \vDash \phi$ *or* $\langle r, v, \tau \rangle \vDash \psi$,
- $\langle r, v, \tau \rangle \vDash \mathbf{X}\phi$ *iff* $\langle r_1, \iota_0, \tau; A_{\hat{\Delta}_0} \rangle \vDash \phi$, *and*
- $\langle r, v, \tau \rangle \vDash \phi \mathbf{U} \psi$ *iff there exists* $0 \leq i$ *such that* $\langle r_i, \iota_{i-1}, \tau; A_{\hat{\Delta}_0}; ...; A_{\hat{\Delta}_{i-1}} \rangle \vDash \psi$ *and for all* j, $0 \leq j < i$, $\langle r_j, \iota_{j-1}, \tau; A_{\hat{\Delta}_0}; ...; A_{\hat{\Delta}_{j-1}} \rangle \vDash \phi$ *where* $\iota_{-1} = \varnothing$ *and* $A_{\hat{\Delta}_{-1}} = 1_{A_{\mathcal{R},\alpha}}$.

If $\mathcal{V}$ is a set of propositions, $\phi, \psi \in LTLForm(\mathcal{V})$, the rest of the boolean constants and operators are defined as usual as: $\mathbf{false} \equiv \neg\mathbf{true}$, $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\phi \implies \psi \equiv \neg\phi \vee \psi$, etc. We define $\Diamond\phi \equiv \mathbf{true}\mathbf{U}\phi$ and $\Box\phi \equiv \neg(\mathbf{true}\mathbf{U}\neg\phi)$.

Definition 58. *Let* $\mathcal{R}$ *be a repository and let*

$$r = \langle \underset{\alpha_0}{\vdash\!\!-} R_0, q_0 \rangle \xrightarrow{\delta_0,\iota_0} \langle \underset{\alpha_1}{\vdash\!\!-} R_1, q_1 \rangle \xrightarrow{\delta_1,\iota_1} \langle \underset{\alpha_2}{\vdash\!\!-} R_2, q_2 \rangle \xrightarrow{\delta_2,\iota_2} \cdots$$

be a successful open execution of $\mathcal{R}^\sharp$. *Then a formula* $\phi \in LTLForm(A_{\mathcal{R},\alpha_0})$ *is satisfied by* r ($r \vDash \phi$) *if and only if* $\langle r, \varnothing, 1_{A_{\mathcal{R},\alpha_0}} \rangle \vDash \phi$.

Following the previous definitions, checking if an activity $\underset{\alpha}{\vdash\!\!-} R$ satisfies a proposition ϕ under a repository $\mathcal{R}$ is equivalent to checking if every successful open execution of $\underset{\alpha}{\vdash\!\!-} R$ with respect to $\mathcal{R}$ satisfies ϕ.

Now we return to the example in Figure 3.3 to state the following very simple property:

$$\phi = \Box \Diamond response_j$$

In our example an execution will only satisfy ϕ if it traverses the transition that goes from (q_1, q_3) to (q_0, q_0) thus preventing the execution form infinitely failing at state (q_1, q_3) properly characterizing the subset of traces that we want to consider as valid.

3.3 Concluding remarks

The approach that we put forward in this chapter combines, in an integrated way, the operational semantics of processes and communication channels, and the dynamic reconfiguration of ARNs. As a result, it provides a full operational semantics of ARNs by means of automata on infinite sequences built from the local semantics of processes, together with the semantics of those ARNs that are selected from a given repository by means of stepwise execution, service discovery and binding. Another use for this semantics is

in identifying the differences between the non-deterministic behaviour of a component, reflected within the execution of an ARN, and the nondeterminism that arises from the discovery and binding to other ARNs.

In comparison with the logic-programming semantics of services described in [62], this gives us a more refined view of the execution of ARNs; in particular, it provides a notion of execution trace that reflects both internal actions taken by services that are already intervening in the execution of an activity, and dynamic reconfiguration events that result from triggering actions associated with a requires-point of the activity. In addition, by defining the semantics of an activity with respect to an arbitrary but fixed repository, it is also possible to describe and reason about the behaviour of those ARNs whose executions may not lead to ground networks, despite the fact that they are still sound and successful executions of the activity.

The proposed operational semantics allows us to use various forms of temporal logic to express properties concerning the behaviour of ARNs that surpasses those considered before. We showed this by defining a variant of the satisfaction relation for linear temporal logic, and exploiting the fact that reconfiguration actions are observable in the execution traces; thus, it is possible to determine whether or not a given service module of a repository is necessarily used, or may be used, during the execution of an activity formalised as an ARN.

In ARNs we were able to state that a successful execution of an activity yields a successful execution of each of its components. This result was not exteneded to HARNs. We leave this as further research.

Many directions for further research are still to be explored in order to provide an even more realistic execution environment for ARNs. Among them, in the current formalism, we did not consider any possible change on the repository during the execution which leads to a naive notion of distributed execution as simple technical problems can make services temporarily unavailable. Our proposal, as a future line of research, is to consider the repository as a dynamic environment which can change along the execution. Also we propose to consider explicit support for compensations.

3.4 Resumen

El objetivo de este trabajo es proporcionar una semántica operacional basada en trazas para diseños de software orientado a servicios que refleje fielmente la naturaleza dinámica de los procesos de *discovery* y *binding* de servicios

que se llevan a cabo en tiempo de ejecución. Para ello hacemos de la reconfiguración de una actividad un evento observable de su comportamiento. En SOC, los eventos de reconfiguración son desatados por acciones particulares asociadas a *requires-points*; cuando ocurre una reconfiguración, la capa intermedia (o *middleware*) debe encontrar un servicio que satisfaga los requerimientos de la actividad dentro de un repositorio de servicios ya conocido. Desde esta perspectiva nuestra propuesta es defininr la ejecución de trazas en las que las acciones puedan ser:

- acciones internas de la actividad; es decir accinoes que no están asociadas con *requires-points* y, por lo tanto, ejecutables sin la necesidad de reconfigurar la actividad, o
- acciones de reconfiguración; es decir acciones que sí están asociadas a un *requires-point* y, por lo tanto, disparan la reconfiguración del sistema por medio del proceso de *discovery* y *binding* de un servicio que provea el cómputo requerido.

Resumiendo, las contribuciones principales del capítulo son:

1. proporcionamos una semántica operacional basada en trazas para las ARNs que refleja tanto transiciones internas que ocurren en cualquiera de los servicios que ya participan del cómputo como acciones de reconfiguración dinámica que son el resultado del proceso de vincular *provides-points* de ARNs tomadas del repositorio con *requires-points* de la actividad, y
2. proporcionamos soporte para definir una técnica de *model-checking* que posibilite el análisis automático de propiedades de actividades en lógica temporal lineal.

En nuestro trabajo consideramos que la semántica se asigna con respecto a un repositorio de servicios dado, forzándonos a dejar de lado la asunción de que dada una ARN es posible encontrar una ARN *ground* hacia la cual la primera tiene un morfismo. Con respecto a trabajos precios, creemos que este enfoque da lugar a un entorno de ejecución más realista donde la potencial satisfacción de los requerimientos está limitada por los servicios registrados en un repositorio, y no por el universo completo de potenciales servicios.

Para definir nuestra semántica se realizan una serie de construcciones basadas en autómatas. Primero se define el autómata de una ARN. Intuitivamente el autómata de una ARN se obtiene tomando el producto de los

autómatas de cada una de sus componentes. Este producto se sincroniza sobre el alfabeto común de las componentes. Se debe notar que la noción de alfabeto común está dada por los mapeos definidos en las conexiones.

Mostramos que tanto el alfabeto como el autómata de una ARN pueden obtenerse como el colímite de un diagrama y que para todo morfismo entre dos ARNs α y α' existe un único morfimo de autómatas de Muller entre los autómatas de α y α' de modo que los diagramas conmuten. Esto nos permite definir un functor desde la categoría de las ARNs hacia la categoría de los autómatas de Muller. Se define entonces el autómata "aplanado" inducido por dicho functor. Este (quasi)-autómata amalgama en una única estructura tanto la configuración como el estado del sistema. Estos dos elementos se ven como un par $\langle \text{ARN}, \text{estado} \rangle$ y las transiciones son pares formados por un morfismo de ARNs y una acción del ARN origen del morfismo. Ahora las transiciones en este autómata puede representar tanto cambios en el estado como cambios estructurales. En este sentid el autómata "aplanado" nos proporciona una visión unificada de ambos aspectos de un sistema orientado a servicios.

Luego se define la ejecución de una ARN α como una ejecución del autómata "aplanado" tal que comienza en un estado inicial de α. La ejecución se dice exitosa si a partir de cierto punto la estructura permanece establece y el conjunto de estados infinitamente visitados pertenece al conjunto de estados finales de esa ARN estable. Mostramos que una ejecución exitosa de una ARN α proporciona una ejecución exitosa del autómata subyacente de α.

Como no hay restricciones sobre los morfismos de ARNs que se utilizan en las transiciones del autómata "aplanado" no es posible decir que una ejecución exitosa proporciona una ejecución exitosa para cada uno de los autómatas subyacentes a cada una de las componentes del sistema. Para ello se introducen restricciones adicionales al definir las ejecuciones de una actividad de modo de forzar que las reconfiguraciones tomen exclusivamente servicios de un repositorio dado tales que la acción que se está ejecutando corresponda con una acción realizable desde el estado inicial de cada uno de los servicios que se agregan. Adicionalmente se agrega la restricción técnica de que los servicios de un repositorio pueden permanecer indefinidamente en su estado inicial. De este modo ahora sí es posible mostrar que una ejecución exitosa de una actividad proporcional una ejecución exitosa de los autómatas subyacentes de cada una de las componentes que la forman.

Luego damos una relación de satisfacción para fórmulas de una lógica temporal lineal en las que las proposiciones son los elementos del alfabeto

de un repositorio. Las proposiciones son traducidas apropiadamente de acuerdo a las reconfiguraciones que sufre una actividad durante su ejecución.

A continuación extendemos la semántica proporcionada para considerar reconfiguraciones que puedan no solo agregar servicios sino también remover. En primera instancia se considera la posibilidad de modelar servicios con ejecución finita. En este caso la remoción es disparada por la finalización del cómputo de un servicio particular. Estos servicios son modelados mediante autómatas finitos y damos una forma de codificar estos autómatas en autómatas de Muller de modo que acepten el mismo lenguaje (módulo una secuencia infinita de transiciones vacías). Las definiciones dadas para la semántica monótona se extiende de manera directa para considerar esta nueva situación con el detalle de que, ahora, en las transiciones del autómata "aplanado" (que, recordamos, son pares de morfismo de ARN y acción) los morfismos considerados pueder ir tanto desde la ARN origen de la transición hacia la ARN destino de la misma o viceversa. Además damos una restricción que nos permite garantizar que el ARN inicial de una ejecución se mantiene a lo lrago de la misma. Esto nos permite extender el resultado de que una ejecución exitosa del autómata aplanado proporciona una ejecución exitosa del autómata subyacente del ARN inicial de la ejecución.

Por último consideramos la posibilidad de remociones debido a fallas. Estas remociones son modeladas como remociones que pueden ocurrir espontáneamente y damos una caracterización de *rollback*. Es decir que ante una remoción espontánea la ejecución es forzada a volver a un estado previo a la aparición de las componentes que fallaron. En esta caracterización una traza que falla infinitamente puede ser exitosa en términos del autómata y por lo tanto mostramos que esta caracterización no es suficiente. Por esto se extiende la noción de satisfacción de fórmulas temporales a la semántica con remociones y mostramos cómo estas fórmulas podrían utilizarse para mitigar este problema.

Chapter 4

On service binding

4.1 Introduction and motivation

Choreography and orchestration are the two main design principles for the development of distributed software (see e.g., [55]). Coordination is attained in the latter case by an *orchestrator*, specifying (and possibly executing) the distributed workflow. 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 workflow 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.

ARNs are accounted among the former type, with the automata labeling the communication hyperarcs playing the role of orchestrators. 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 coalesced, the nodes become "internal", that is they are no longer part of the interface and cannot be used for further bindings. In existing works

(e.g., [23]), the binding is subject to a *linear temporal logic* [57] entailment relation between theories attached to the provides- and requires-points. Such an entailment can be checked by resorting to any decision procedure for LTL (e.g., [37]).

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 automata labelling the communication hyperarcs), in the composition, together with the computational units (the automata labelling the process hyperarcs) increases the size of the resulting automaton making any potential analysis much more expensive.

Recently *choreographies* have been advocated as suitable alternatives for analysis of distributed applications (see e.g., [4, 35, 36, 43]). In a choreographed model, the distributed computational components are represented as partial view of the multiparty communication. The global behaviour can then be obtained by executing the processes participating in the communication in parallel. The main difficulty intrinsic to choreographed models is to check whether a set of participants willing to engage in a multiparty communication will succeed in executing in parallel, in other words, whether that set of participants can interoperate free of errors or not.

Communicating machines [9] is an automata-based formalism for describing those partial viewpoints in choreographies. They describe, for a given participant, its role in a multiparty communication, in terms of message exchange (either sent or received). For the case of communicating machines, this interoperability check was solved in [38] by means of an algorithm for synthesising choreographies from communicating machines. There, if the synthesis algorithm finishes yielding a choreography, then the interoperability is guaranteed by what is called multiparty compatibility so the composition is free from communication errors.

We represent choreographies as *global graphs*, to represent the global view, and *communicating finite state machines* (CFSMs) [9], to represent individual processes. A global graph [19] is basically a workflow graph representing the causality relations of interactions as well as the points of distributed choice and fork/join of threads. On the other hand, a CFSMs

is a finite state automaton where all the states are accepting and whose transitions are labelled on a set of *actions* where an action is either a *sending* action $\ell = \mathsf{pq!a}$, namely p writes message a in the buffer to q, or a *receiving* action $\ell = \mathsf{pq?a}$ from channel pq, namely q inputs message a from the channel from p.

A system consisting of *communicating finite state machines* evolves through communication interactions on channels represented pq from p to q; sent messages are kept in an unbound FIFO buffer (or queues) until the receivers input them.

In this thesis we propose *Communicating Relational Networks* (CRNs), a variant of ARNs whose binding is determined by a check relying on a blend of choreographies and orchestration as a possible way of overcoming the issues pointed out before. The solution we propose rests on that provides-points are labelled with *Communicating Finite State Machines* declaring the behaviour (from the communication perspective) exported by the service, and communication hyperarcs are labelled with *Global Graphs* [19] declaring the global behaviour of the communication channel. 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 chapter is organised as follows; in Section 4.2 we provide the formal definitions of most of the concepts used along the chapter. In Section 4.3 we introduce an example that will allow us to depict the concepts introduced in Section 4.4. Conlusions of this part are discussed in Section 4.5. Later on in Section 4.6 an extension to communicating machines is presented capable of handling data and conditions over the data guarding transitions. Afterwards in Section 4.7 and Section 4.8 two lines of further development are discussed with enough detail level to deserve their own sections.

4.2 Formal introduction

In this section we present the preliminary definitions used throughout the rest of the present chapter. We summarise communicating machines and global graphs borrowing definitions from [38] and from [19]. We also resort to definitions of ARNs; we direct the reader to definitions given in Chapter 2.

4.2.1 Communicating machines and global graphs

Communicating Finite State Machines (CFSMs for short) were introduced in [9] 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* ranged over by $\mathsf{a}, \mathsf{b}, \ldots$ and a finite set P of participants ranged over by $\mathsf{p}, \mathsf{q}, \ldots$. Then CFSMs are just finite state machines that communicate asynchronously through one-directional, unbounded, order-preserving buffers. Hence, a CFSM is a finite state machine whose transitions are labelled by actions denoting write and read operation over buffers. Then, the set of labels is defined by the following grammar:

$$Act_{\mathsf{Msg}} ::= \mathsf{pq!a} \mid \mathsf{pq?a}(\mathsf{p} \neq \mathsf{q})$$

An action $\ell \in Act_{\mathsf{Msg}}$ is a *sending* action $\ell = \mathsf{pq!a}$, namely p writes message $\mathsf{a} \in \mathsf{Msg}$ in the buffer to q or a *receiving* action $\ell = \mathsf{pq?a}$, namely q inputs a message a from the buffer from p.

Definition 59 (Communicating Fininte State Machine [9])**.** *A* communicating finite state machine *on* P *and* Msg *(CFSMs, for short) is a finite transition system* $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *where*

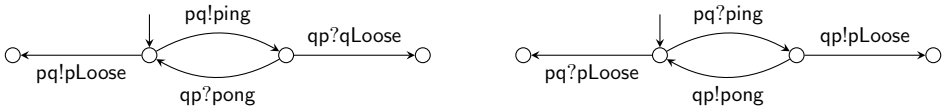
- Q *is a finite set of states;*
- $q_0 \in \mathsf{Q}$ *is an initial state;*
- $\delta \subseteq \mathsf{Q} \times Act_{\mathsf{Msg}} \times \mathsf{Q}$ *is a finite set of* transitions.

A communicating system *is a map S assigning a CFSM $S(\mathsf{p})$ to each $\mathsf{p} \in \mathsf{P}$ such that $\delta_{\mathsf{p}} \subseteq \mathsf{Q}_{\mathsf{p}} \times \{\mathsf{pq!a} \mid \mathsf{q} \in \mathsf{P}$ and $\mathsf{a} \in \mathsf{Msg}\} \cup \{\mathsf{qp?a} \mid \mathsf{q} \in \mathsf{P}$ and $\mathsf{a} \in \mathsf{Msg}\} \times \mathsf{Q}_{\mathsf{p}}$. We write $q \in S(\mathsf{p})$ when q is a state of the machine $S(\mathsf{p})$ and likewise $\tau \in S(\mathsf{p})$ when τ is a transition of $S(\mathsf{p})$.*

A CFSM $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ is deterministic if for all states $q \in \mathsf{Q}$ and all actions $\ell \in Act_{\mathsf{Msg}}$, if $(q_{\mathsf{p}}, \ell, q'_{\mathsf{p}}) \in \delta$ and $(q_{\mathsf{p}}, \ell, q''_{\mathsf{p}}) \in \delta$ then $q'_{\mathsf{p}} = q''_{\mathsf{p}}$. A CFSM M is minimal if there is no machine M' with fewer states than M such that $L(M) = L(M')$. Hereafter, we only consider deterministic and minimal CFSMs.

Notice that we consider only deterministic CFSMs in order to adjust ourselves to the conditions required in [38]. This will impose some restrictions when we consider data associated to transitions.

The semantics of a communicating system is given by a labelled transition system over *configurations*, which keep track of the state of each



(a) Particpant p (b) Particpant q

Figure 4.1: Example of CFSMs

machine and the content of each buffer in the system. Given a communicating system S, the buffers of S are $C = \{\mathsf{pq} \mid \mathsf{p}, \mathsf{q} \in \mathsf{P} \ \wedge \ \mathsf{p} \neq \mathsf{q}\}$. The content w_{pq} of the channel pq is given by a (possibly empty) sequence of action names, i.e., $w_{\mathsf{pq}} \in \mathsf{Msg}^*$. Then the execution of a system is defined in terms of transitions between configurations as follows:

Definition 60. *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 $w_{\mathsf{pq}} \in \mathsf{Msg}^*$. A configuration $s' = (\vec{q'}, \vec{w'})$ is* reachable *from another configuration $s = (\vec{q}, \vec{w})$ by the* firing of the transition τ *(written $s \xrightarrow{\tau} s'$) if there exists $\mathsf{m} \in \mathsf{Msg}$ such that either:*

Snd $\tau = (q_{\mathsf{p}}, \mathsf{pq!m}, q'_{\mathsf{p}}) \in \delta_{\mathsf{p}}$ *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* $w'_{\mathsf{p'q'}} = w_{\mathsf{p'q'}}$ *for all* $\mathsf{p'q'} \neq \mathsf{pq}$*; or*

Rcv $\tau = (q_{\mathsf{q}}, \mathsf{pq?m}, q'_{\mathsf{q}}) \in \delta_{\mathsf{q}}$ *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* $w'_{\mathsf{p'q'}} = w_{\mathsf{p'q'}}$ *for all* $\mathsf{p'q'} \neq \mathsf{pq}$

Rule Snd stands for the behaviour of the participant s putting the message a on buffer sr, while rule Rcv models the participant r consuming a message a from buffer sr. Note that a participant s cannot write messages on buffers w_{pq} when $\mathsf{p} \neq \mathsf{s}$. Similarly, r cannot read messages from w_{pq} when $\mathsf{q} \neq \mathsf{r}$. Figure 4.1 show two very simple CFSMs playing ping-pong. There both machines can play indefinitely until one of them looses.

As stated before, the global point of view of a multiparty commmunication is represented as a *global graph*. 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.

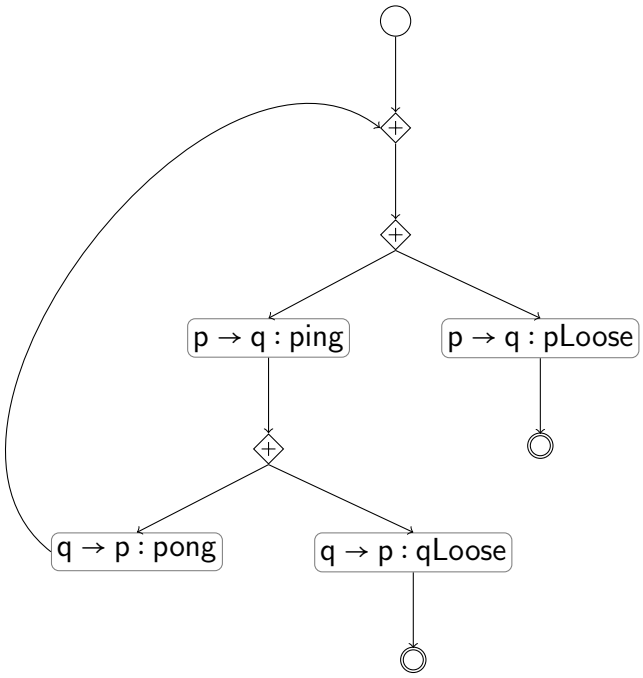


Figure 4.2: Global graph for the ping-pong protocol

Definition 61. *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. Figure 4.2.1 shows the global graph for the ping-pong protocol. Note that there are no fork/joins since there is concurrency in the protocol. On the contrary participants progress in turns, one at a time.

4.2.2 General Multiparty Compatibility (GMC)

In principle, given a communicating system, there is no guarantee that the participants will interoperate correctly (i.e., free of communication errors). The notion of correctness that we adopt requires that systems are deadlock-free and reach no *unspecified-reception* or *orphan message* configuration.

We say that a configuration $s = (\vec{q}, \vec{w})$ of a communicating system S is a *deadlock configuration* [16] if $\vec{w} = \vec{\epsilon}$, there is $\mathsf{r} \in \mathsf{P}$ such that $(q_\mathsf{r}, \mathsf{sr}?\mathsf{m}, q'_\mathsf{r}) \in \delta_\mathsf{r}$ and for ever $\mathsf{p} \in \mathsf{P}$, q_p is a receiving or final state. This means that all the buffers are empty and there is at least one machine waiting for a message and all of the other machines are in a receving or final state.

We say that a configuration $s = (\vec{q}, \vec{w})$ of a communicating system S is an *orphan message configuration* if all $q_\mathsf{p} \in \mathsf{P}$ are final but $\vec{w} \neq \vec{\epsilon}$. That means there is at least one non empty buffer and each machine is in a final state.

An *unspecified reception configuration* [16] is a configuration $s = (\vec{q}, \vec{w})$ of a communicating system S such that there exists $\mathsf{r} \in \mathsf{P}$ such that q_r is a receiving state and if $(q_\mathsf{r}, \mathsf{sr}?\mathsf{m}, q'_\mathsf{r}) \in \delta_\mathsf{r}$ then $w_\mathsf{sr} \neq \epsilon$ and $w_\mathsf{sr} \neq \mathsf{m} \cdot w'_\mathsf{sr}$. This means that q_r is prevented from receiving from any of its buffers while in a receiving state.

In [38] a condition is given to guarantee the absence of communication errors, which enables the automated transformation from *synchronous transition system* (as they are defined below in Definition 62) into choreography models. The condition resorts to checking two properties over the synchronous transition system of a communicating system. The two properties are: representability and branching property. Representability states that for each machine, each trace and each choice are represented in the synchronous transition system, while branching property states that whenever there is a choice there is a unique machine that takes the decission and the rest are either made aware of that decission or not involved in the choice at all. For formal definitions on these properties we refere the reader once again to [38]. If the synchronous transition system of a communicating system has these two properties then the communicating system is said to be *general multiparty compatible* (or GMC) and is guaranteed to be free of communication errors.

More recent works [28, 61] introduce a more general notion of well-formedness. There the authors generalise the notion of well-formedness using two different types of semantics, one based on pomsets and the other on hypergraphs representing partial orders. In this way they manage to

avoid the usual syntactic restrictions that limit expressiveness of global specifications.

The algorithm given in [38] synthesises a *global graph*, which specifies the distributed workflow of multiple CFSMs, in which interactions between participants are described as synchronous interaction labels of the form $\mathsf{s} \twoheadrightarrow \mathsf{r} : \mathsf{m}$, with $\mathsf{s}, \mathsf{r} \in \mathsf{P}$ and $\mathsf{m} \in \mathsf{Msg}$. We write Int_{Msg} for the set of all synchronous interaction labels. If choreography models are free from communication errors, then the communicating machines projected from the synthesised global graph are bisimilar to the original ones.

Definition 62 (Synchronous transition system)**.** *Let $S = (M_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ be a* communicating system*. The synchronous semantics of S is given by the LTS $\mathcal{S}(S) = ((Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}, Int_{\mathsf{Msg}}, \delta)$ with $\delta \subseteq (Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \times Int_{\mathsf{Msg}} \times (Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ defined by the following rule:*

$$\frac{\mathsf{s}, \mathsf{r} \in \mathsf{P} \qquad q_{\mathsf{s}} \xrightarrow{\mathsf{sr!m}} q'_{\mathsf{s}} \qquad q_{\mathsf{r}} \xrightarrow{\mathsf{sr?m}} q'_{\mathsf{r}} \qquad (\forall \mathsf{p} \notin \{\mathsf{s}, \mathsf{r}\})\ q'_{\mathsf{p}} = q_{\mathsf{p}}}{(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \xrightarrow{\mathsf{s} \twoheadrightarrow \mathsf{r}:\mathsf{m}} (q'_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}}\ \textsc{Int}$$

Each participant's CFSM can be recovered from either the synchronous transition system or from a global graph by means of projection [38]. We denote the projection with respect to a participant $\mathsf{p} \in \mathsf{P}$ by $\upharpoonright_{\mathsf{p}}$. The set of *reachable configurations of $\mathcal{S}(S)$* is

$$\mathbf{RS}(\mathcal{S}(S)) = \{s | (q_{0\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \rightarrow^* s\}$$

where $\rightarrow^*$ is the reflexive transitive closure of the relation given by rule Int in Definition 62. Given a state $(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ then $(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \upharpoonright_{\mathsf{r}} = q_{\mathsf{r}}$.

Definition 63. *Let $S = (M_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ be a communicating system then, the* projection of q from $\mathcal{S}(S)$ *is the CFSM $M \upharpoonright_{\mathsf{q}} = \langle Q \upharpoonright_{\mathsf{q}}, q_0 \upharpoonright_{\mathsf{q}}, \mathsf{Msg}, \delta \upharpoonright_{\mathsf{q}} \rangle$ where:*

- $Q \upharpoonright_{\mathsf{q}} = \left\{ q \upharpoonright_{\mathsf{q}} \;\middle|\; q \in \mathbf{RS}(\mathcal{S}(S)) \right\}$
- $\delta \upharpoonright_{\mathsf{q}}$ *is given by the following rules:*

$$\frac{(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \xrightarrow{\mathsf{s} \twoheadrightarrow \mathsf{r}:\mathsf{a}} (q'_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \in \mathcal{S}(S) \qquad \mathsf{q} = \mathsf{s}}{(q_{\mathsf{s}}, \mathsf{sr!a}, q'_{\mathsf{s}}) \in \delta \upharpoonright_{\mathsf{q}}}\ \textsc{E-Send}$$

$$\frac{(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \xrightarrow{\mathsf{s} \twoheadrightarrow \mathsf{r}:\mathsf{a}} (q'_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \in \mathcal{S}(S) \qquad \mathsf{q} = \mathsf{r}}{(q_{\mathsf{s}}, \mathsf{sr?a}, q'_{\mathsf{s}}) \in \delta \upharpoonright_{\mathsf{q}}}\ \textsc{E-Rcv}$$

Definition 64 (Simulation)**.** *We say that a machine* $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *simulates a machine* $(\mathsf{Q}', q_0', \mathsf{Msg}, \delta')$ *if and only if there exists a relation* $S \subseteq \mathsf{Q} \times \mathsf{Q}'$ *such that* $(q_0, q_0') \in S$ *and for all* $(q, q') \in S$ *if* $(q', \ell, q_\ell') \in \delta'$ *then* $(q, \ell, q_\ell) \in \delta$ *and* $(q_\ell, q_\ell') \in S$*. S is called a* simulation*. If both S and* S^{-1} *are simulations then S is a bisimulation.*

If there is a bisimulation between machines M and M′ they are regarded as bisimilar*.*

Proposition 12. *Let* $S = (M_\mathsf{p})_{\mathsf{p} \in \mathsf{P}}$ *be a communicating system then, the* $\mathcal{S}(S)\restriction_p \leq M_\mathsf{p}$ *(*M_p *simulates* $\mathcal{S}(S)\restriction_p$*) for all* $\mathsf{p} \in \mathsf{P}$*.*

Proof. It follows from observing that $Q\restriction_\mathsf{p} \subseteq Q_\mathsf{p}$, that $\delta\restriction_\mathsf{p} \subseteq \delta_\mathsf{p}$ and that q_0 is always in the projection. □

Note that the converse of Proposition 12 does not hold in general.

4.3 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.

Figure 4.3, Figure 4.4, and Figure 4.5 show the ARNs (including the automata), for the TravelClient, HotelsService, and PaymentProcessService respectively. The ARN in Figure 4.3(a) represents an activity composed with a communication channel. More precisely, TravelClient (in the solid box on the left) represents a process hyperedge whose Muller automaton is Λ_{TC} (depicted in Figure 4.3(b)). The solid "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 Muller automaton associated with the communication hyperarc that acts as the orchestrator of the services. In our running example, the communication hyperarc of Figure 4.3 is labeled with the automaton Λ_{CC} of Figure 4.3(c) where, for readability and conciseness, the dotted and dashed edges stand for the paths

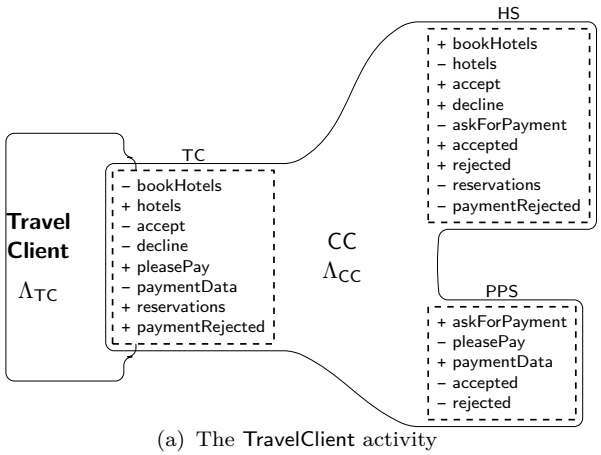
$$\xrightarrow{bookHotels!} \cdot \xrightarrow{bookHotels_{¡}} \cdot \xrightarrow{hotels!} \cdot \xrightarrow{hotels_{¡}}$$

and

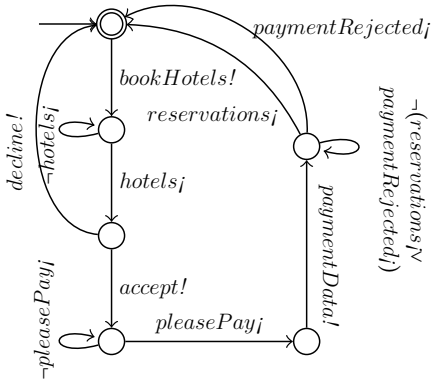
$$\xrightarrow{accept!} \cdot \xrightarrow{accept_{¡}} \cdot \xrightarrow{askForPayment!} \cdot \xrightarrow{askForPayment_{¡}} \cdot \xrightarrow{paymentData!} \cdot \xrightarrow{paymentData_{¡}}$$

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 $m_{¡}$ stands for the input of message m.

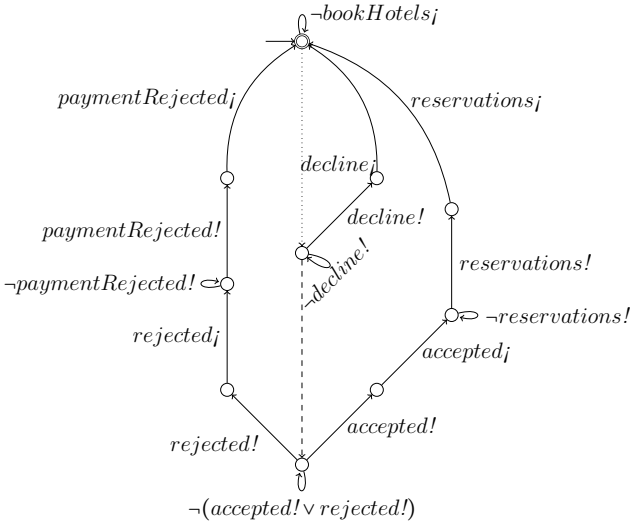
Figure 4.4 and Figure 4.5 represent two services with their automata (resp. Λ_{HS} and Λ_{PPS}) and their provides-point (resp. HS and PPS) not bound to any communication channel yet.



(a) The TravelClient activity

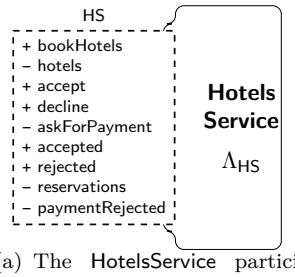


(b) Muller automaton Λ_{TC}

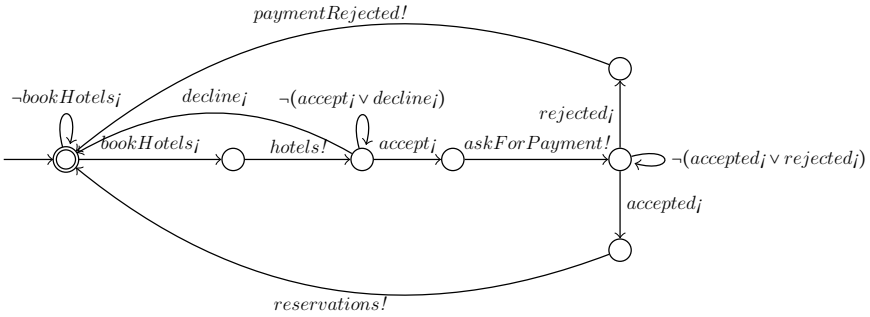


(c) Muller automaton Λ_{CC}.

Figure 4.3: The TravelClient activity together with the Muller automata.

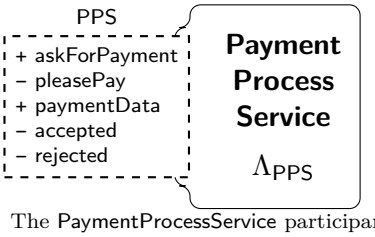


(a) The HotelsService participant

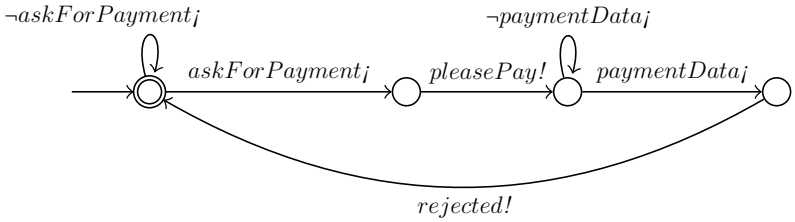


(b) Muller automaton Λ_{HS}

Figure 4.4: The HotelsService participant together with the machine Hs



(a) The PaymentProcessService participant



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

Figure 4.5: The PaymentProcessService participant.

The composition of ARNs yields a semantic definition of a binding mech-

anism of services in terms of "fusion" of provides-points and requires-points. More precisely, the binding is subject to an entailment relation between *linear temporal logic* [57] theories attached to the provides- and requires-points as illustrated in the following section.

4.4 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.

4.4.1 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 Figure 4.3(a). Likewise, one could specify a contract for the provides-point PPS of the ARN in Figure 4.5 (b) stating that payments are always rejected by including the formula[1]

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

It is easy to show that

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

by resorting to any decision procedure for LTL (see for instance, [37]). 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.

[1] In these 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.

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 guarantee that all possible outcomes will eventually be produced.

In order to overcome these limitations we present a modification to ARNs that we call *Communicating Relational Networks* (or CRNs for short). CRNs are defined exactly as ARNs but with the definition of *Connection* based on 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. Also we add an extra labelling function for ports in order to assign a CFSM to each port. These CFSMs models the declared behaviour over that port. For reasons that will be discussed afterwards in Section 4.5 we take Muller automata defined over sets of actions instead of over powersets of actions.

Definition 65 (Connection)**.** *Let γ be a set of pairwise disjoint ports. We say that $\langle M, \mu, \Gamma, \mathsf{P}\rangle$ is a* connection *on γ iff $\langle M, \mu\rangle$ is an attachment injection on γ and Γ is a global graph whose set of participants is $\mathsf{P} = \{\mathsf{p}_\pi\}_{\pi\in\gamma}$ exchanging messages in M such that:*

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

for each $\pi \in \gamma$.

Definition 66 (Process)**.** *A* process $\langle \gamma, \Lambda, \kappa, \{\mathsf{P}_M\}_{M\in\gamma}\rangle$ *consists of a set γ of pairwise disjoint ports, a familly $\{\mathsf{P}_M\}_{M\in\gamma}$ of pairwise disjoints sets of particpants, a set of CFSMs $\kappa = \{\kappa_M\}_{M\in\gamma}$ such that κ_M is defined over the set of messages M and the set of participants P_M, and a Muller automaton Λ over the set of actions $A_\gamma = \bigcup_{M\in\gamma} A_M$, where $A_M = \{m! \,\big|\, m \in M^-\} \cup \{m_i \,\big|\, m \in M^+\}$.*

Note that unlike CFSMs where a message is always sent over a particular channel, in ARNs connections may (partially) fuse the language of two or more ports indicating a concurrent send/reception. This can be modeled in global graphs using forks. Nevertheless, for simplicity in the presentation we will assume that the attachment injections are actually bijections. This means that messages from different ports (with the same direction) are never mapped to the same label in the connection.

Definition 67 (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* $\langle \gamma_p, \Lambda_p, \kappa_p, \{\mathsf{P}^p_M\}_{M\in\gamma_p}\rangle$ *to each hyperedge* $p \in P$*, and* (c) *a connection* $\langle M_c, \mu_c, \Gamma_c, \mathsf{P}^c\rangle$ *to each hyperedge* $c \in C$ *such that for all* $x \in X$ *adjacent to a process* p *and to a connection* c *(i.e.* $x \in \gamma_p$ *and* $x \in \gamma_c$ *with* $p \in P$ *and* $c \in C$*)* $\mathsf{P}^p_{M_x} = \mathsf{P}^c$*.*

A service cannot know in advance with whom it will interact. The use of CFSMs imposes a strong restriction in this aspect as a result of the need to have a prefixed set of participants over which a machine is defined. Nevertheless the reader should note that the set of participants acts simply as a set of names and besides being finite it plays no special role on the formalization. Therefore we can mitigate this restriction by relying on *renamings* of participans names in the following way:

Definition 68 (Renaming of CFSMs)**.** *Given a set of messages* Msg*, two sets of participants* P *and* P'*, a bijection between them* $ren\colon \mathsf{P} \leftrightarrow \mathsf{P}'$ *and two CFSMs* $\kappa = (\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *defined over* Msg *and* P *and* $\kappa' = (\mathsf{Q}', q'_0, \mathsf{Msg}, \delta')$ *defined over* Msg *and* P' *we say that* κ' is the renaming of κ along *ren if and only if:*

- $\mathsf{Q}' = \mathsf{Q}$
- $q'_0 = q_0$
- $\delta' = \{(q, ren(\ell), s) \mid (q, \ell, s) \in \delta\}$

. Where

$$ren(\ell) = \begin{cases} ren(\mathsf{s})ren(\mathsf{r})!\mathsf{m} & \ell = \mathsf{sr}!\mathsf{m} \\ ren(\mathsf{s})ren(\mathsf{r})?\mathsf{m} & \ell = \mathsf{sr}?\mathsf{m} \end{cases}$$

Now with renamings of CFSMs in hand we can define morphisms if CRNs in a similar way to morphisms of ARNs:

Definition 69 (Morphisms of CRNs)**.** *A* morphism $\delta\colon \alpha \to \alpha'$ *between two CRNs* $\alpha = \langle X, P, C, \gamma, M, \mu, \Lambda\rangle$ *and* $\alpha' = \langle X', P', C', \gamma', M', \mu', \Lambda'\rangle$ *consists of*

- *an injective map* $\delta: X \to X'$ *such that* $\delta(P) \subseteq P'$ *and* $\delta(C) \subseteq C'$*, that is an injective homomorphism between the underlying hypergraphs of* α *and* α' *that preserves the computation and communication hyperedges, and*
- *a family of polarity-preserving injections* $\delta_x^{pt}: M_x \to M'_{\delta(x)}$*, for* $x \in X$*,*
- *a family of participants bijections* $\delta_x^{part} : \mathsf{P}_x \leftrightarrow \mathsf{P}_{\delta(x)}$*,*

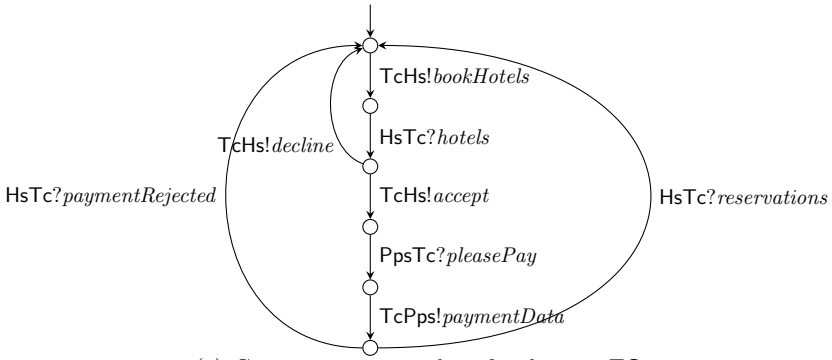
such that

- *for every point* $x \in \bigcup P$*,* $\delta_x^{pt} = 1_{M_x}$*,*
- *for every computation hyperedge* $p \in P$*,* $\Lambda_p = \Lambda'_{\delta(p)}$*, for every* $x \in \gamma_p$ $\kappa^{\delta(p)}_{M_{\delta(x)}}$ *is the renaming of* $\kappa^p_{M_x}$ *along* δ_x^{part}*, and*
- *for every communication hyperedge* $c \in C$*,* $M_c = M'_{\delta(c)}$*,* $\mathsf{P}^c = \mathsf{P}^{\delta(c)}$*,* $\Gamma_c = \Gamma'_{\delta(c)}$ *and, for every point* $x \in \gamma_c$*,* $\mu_{c,x}; \delta_x^{pt} = \mu'_{\delta(c),\delta(x)}$*.*

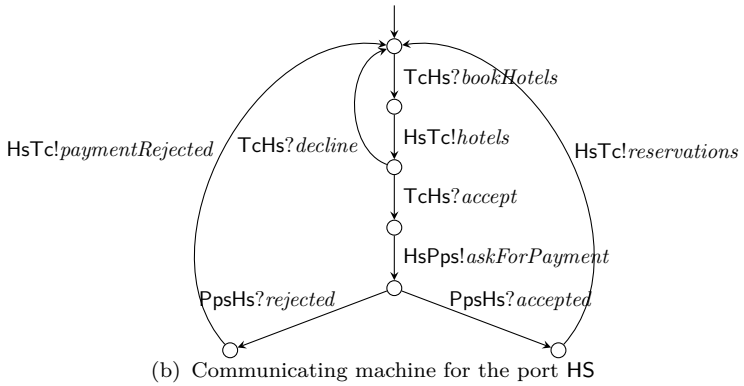
Morphisms of CRNs are in essence similar to morphisms of ARNs only that now we consider CFSMs labelling the ports and Global Graphs labelling the communication hyperedges. Moreover we also consider the possibility of renaming the set of participants over which the CFSMs that label the ports are defined. Note that there are no sanity checks as to whether a set of CFSMs labelling ports adjacent to a single connection form a communicating system or not. This assertion needs to be added to the binding check.

Figure 4.6 and Figure 4.7 show the communicating machines and global graphs that can be used to redefine the same services of the running example presented in Section 4.3, but as CRNs. To ease the presentation in the example we ommit the participants renamings.

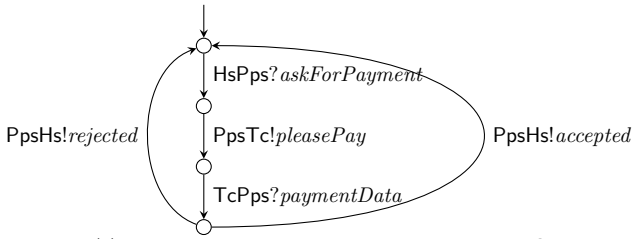
The machine in Figure 4.6(a) specifies that upon reception of a *bookHotel* message from the client, HotelsService sends back a list of *hotels*; if the client accepts then computation continues, otherwise the HotelsService returns to its initial state, etc.. Also, Figure 4.6(b) and (c) depict the communicating machines associated to the provides-points of services HotelsService and 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 Figure 4.7 (corresponding to the automaton in Figure 4.3(c)). The graph dictates that



(a) Communicating machine for the port TC



(b) Communicating machine for the port HS



(c) Communicating machine for the port PPS

Figure 4.6: Communicating machines labelling the ports TC, HS and PPS.

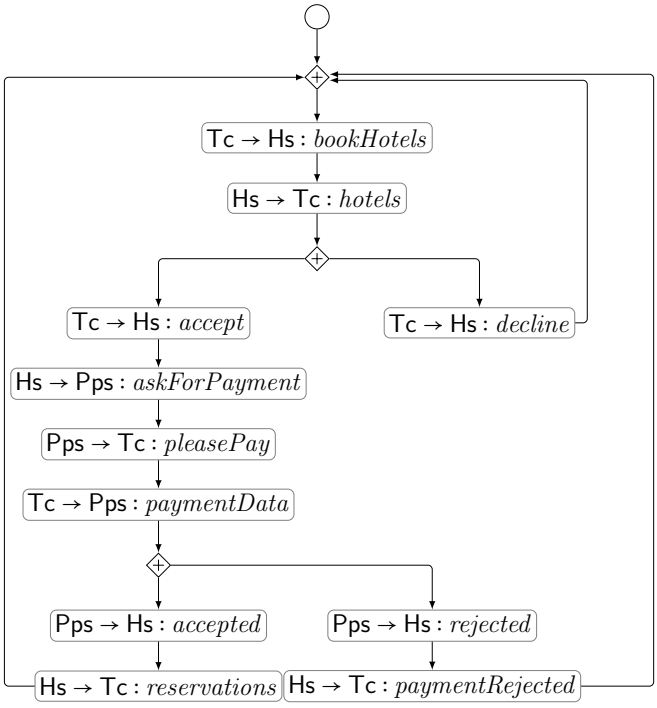


Figure 4.7: Global graph of the running example

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 Figure 4.7 branching points merge in the loop-back node underneath the initial node.

Binding of services' provides-points to an activity's requires-points is now subject to the ability to find, for each of the services provides-points, a participants bijection between the participants set used to define the CFSM labelling the provides-point of a service to the participants sets used to define the global graph labelling the connection in a way such that the resulting set of CFSMs form a communicating system and one of the following approaches can be asserted:

Based on Definition 67, 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 requires-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).

Bottom-Up The second mechanism is more flexible and it is based on a recent algorithm to synthesise choreographies out of communicating machines [38]. 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 Figure 4.7 with respect to the components HotelsService and PaymentProcessService yields the communicating machines in Figure 4.6(b) and Figure 4.6(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 [9]). Likewise, when adopting the second criterion, the binding is possible because the synthesis of the machines in Figure 4.6 yields the global graph of Figure 4.7.

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.

4.4.2 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 provides- and requires-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 4.1 shows a comparison of the procedure that has to be implemented for obtaining a binding mechanism for both of the approaches, the one based on ARNs, and the one based on CRNs.

Formalisation	Binding Mechanism
ARNs	$$\Gamma_\pi \vdash^{\mathsf{LTL}} \Gamma_\rho$$ 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	**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 [38] and $\equiv$ denotes isomorphism.

Table 4.1: Comparison of the procedures for the approaches based in ARNs and CRNs

4.5 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 requires-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 acceptance condition. Our approach provides a more symmetric mechanism built on top of the notion of bisimulation of CFSMs.

Notice that this approach requires the definition of a criterion to establish the coherence between the Muller automaton Λ of a process hyperedge and the communicating machines associated to its ports. 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. This projection, or in general, obtaining the observable behavior, is not trivial. For a more detailed discussion we refer the reader to Section 4.9 where we propose and discuss a different approach that consists in deriving the communication interface of an automaton directly from its definition.

We strived here for simplicity suggesting simple 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 [3]). 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. In Section 4.9 we draw a first attempt at ad-

dressing this problem by introducing a new kind of automata that internalize the asynchronous communication semantics when composed. Recently in [2] the authors address the problem of generalizing the notion of global type to describe open systems of CFSMs. There the authors propose global type with interface roles (GTIR) that denote a number of connected open systems of CFSMs where some participants are identified as interfaces rather than proper participants. A semantic compatibility condition is given such that if two GTIRs are connected through compatible interfaces then error-freeness (i.e. deadlock-freeness, no-orphan-message and no-unspecified-reception) is preserved.

4.6 Data-aware communicating finite state machines

4.6.1 Motivation

In Section 4.4 we proposed a way of integrating CFSMs within ARNs in a way that could allow us to rely on the algorithm presented in [38] in order to perform the binding check in a SOC system. One weakness of this approach is that while CFSMs are a suitable formalism to express communication protocols they are not intended for, and have no ability to express more complex functional contracts. In a service oriented system binding check should include a) communication interoperability check, b) functional contracts check and c) quality of service terms check. We argue that CRNs as presented in Section 4.4 are suitable for solving a) but not b). In the present section we introduce an extension of CFSMs that considers data sent and received within messages' payload and logical conditions associated to this data. We also present a data-aware multiparty compatibility property that allows us to extend results from [38] to this new kind of machines.

The necessity of considering values associated to communication models is not novel. In [18], Delzanno and Bultan use guards over a fixed set of global variables. This guards are associated to transitions and act as pre and post-conditions predicating over the variables allowing for a richer control structure in the communicating machines.

In [26], the authors propose a class of guarded communicating machines in which guards predicate over the values associated to messages but still act as pre- and post-conditions over these values. In some sense the guards can be interpreted as the specification of a transformation of the values

performed by the transtion to which the guard is associated.

Two works that are closer to the approach we are presenting in this section are [50, 51]. In the first one, the authors present a class of guarded choreographies and a class of guarded communicating finite state machines, and introduce a projection mechanisms which, given a data-aware choreography, behavioural skeletons for all the participants are produced. In the second one, the authors present a notion of conformance between a set of peers and a choreography; this is based on a definition of projection of a role from a choreography (the one presented in [51]), and an appropriate definition of bisimilarity between peers and roles.

In the present work we adopt the same view of guarded choreographies and guarded communicating state machines presented in [51]. The guards predicate over values associated to messages being exchanged by the participants. The rationale behind the formalism can be summarised as follows: - when an action $\mathsf{pq!m}\langle x\rangle$ is labelled with a guard φ, then the transition is understood as "Participant p sends message m to participant q such that the value x associated to m satisfies φ." being a sort of an ensures clause, and - when an action $\mathsf{pq?m}(x)$ is labelled with a guard φ, the transition is understood as "Participant q receives message m from participant p if the value x associated to m satisfies φ." being a sort of an assumes clause.

4.6.2 Guarded Communicating Machines: Compatibility

We fix the following syntax of a fragment of first-order logic, which we assume to be decidable on closed predicates.

$$\begin{array}{lcl} F, G & ::= & \mathsf{true} \,\big|\, \mathsf{false} \,\big|\, \phi(e_1, \ldots, e_n) \,\big|\, \neg F \,\big|\, F \wedge G \,\big|\, F \supset G \,\big|\, \exists x(F) \,\big|\, \ldots \\ e_1, e_2 & ::= & \mathsf{n} \,\big|\, x \,\big|\, \cdots \end{array}$$

We write $\mathbb{G}$ for the language generated from the above grammar. Above, ϕ ranges over pre-defined atomic predicates with fixed arities and types (e.g., Bool, Int, etc) [45, §2.8], x are the *interaction variables* representing the content of the messages exchanged by the participants; the (infinite) set of all interaction variables is denoted as $\mathcal{V}$; e_i ranges over expressions; we do not fix the language of expressions and just assume that it encompasses interaction variables, usual data types of programming languages, and constants (denoted with n). Hereafter we denote a finite vector of pair-wise disjoint interaction variables by $\vec{x}$. We denote the set of *free interaction variables* of G with $\mathrm{fv}\big(G\big)$, similarly for $\mathrm{fv}\big(e\big)$.

We extend the syntax of action labels to account for the values exchanged during communication:

$$Act_{\mathsf{Msg}} ::= \mathsf{pq!a}\langle\vec{\mathsf{x}}\rangle \mid \mathsf{pq?a}(\vec{\mathsf{x}})$$

We denote the set of *free interaction variables* of ℓ with $\mathrm{fv}(\ell)$.

Definition 70 (Guarded communicating finite state machines)**.** *A* guarded CFSM *is a 5-tuple* $G = (Q, q_0, \mathsf{Msg}, \delta, \mathcal{G})$ *where* $(Q, q_0, \mathsf{Msg}, \delta)$ *is a CFSM and* $\mathcal{G}$ *is a function that assigns a guard to each transition. We write* $q \xrightarrow[\alpha]{\ell} q'$ *when* $(q, \ell, q') \in \delta$ *and* $\mathcal{G}(q \xrightarrow{\ell} q') = \alpha$*. We require that* $\mathrm{fv}(\alpha) \subseteq \mathrm{fv}(\ell)$*. We denote with* $\mathrm{fv}(M)$ *the set of free variables of* M*, i.e.,* $\mathrm{fv}(M) = \bigcup_{q \xrightarrow[\alpha]{\ell} q'} \mathrm{fv}(\ell)$.

Without loss of generality we assume that no two transitions use the same variable as a sanity condition to avoid name clashes. That is that given $q \xrightarrow[\alpha]{\ell} q'$ and $s \xrightarrow[\alpha']{\ell'} s'$ then $\mathrm{fv}(\ell) \cap \mathrm{fv}(\ell') = \varnothing$.

A GCFSM $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta, \mathcal{G})$ is deterministic if for all states $q \in \mathsf{Q}$ and all actions $\ell \in Act_{\mathsf{Msg}}$

1. if $(q, \ell, q') \in \delta$ and $(q, \ell', q'') \in \delta$ such that $\ell \equiv \ell'$ then $q' = q''$ (i.e. arrows tagged with the same message have the same target state) and,
2. $\forall \ell' \in \delta$ such that $\ell \equiv \ell'$, $\mathrm{fv}(\ell) = \mathrm{fv}(\ell') \implies \mathcal{G}((q, \ell, q')) \iff \mathcal{G}((q, \ell', q''))$ (i.e. conditions associated with a message are equivalente along all the transitions with respect to a particular channel)[2].

Where $\ell \equiv \ell'$ if and only if they are both sending or receiving actions, the buffers are the same and the message labels in them are equal. Note that we take a conservative approach here as we require that machines are deterministic in the sense that the same message is always sent with the same guarantee or received under the same assumption. This simplifies the result presented in this section as it can be directly related to the result in [38].

We straightforwardly extend the definition of communicating systems to consider guarded CFSMs instead of just CFSMs, but we require any guarded CFSMs to be defined using just local variables, i.e., for $S = (G_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ it holds that $\mathsf{p} \neq \mathsf{q}$ implies $\mathrm{fv}(G_{\mathsf{p}}) \neq \mathrm{fv}(G_{\mathsf{q}})$.

[2]Here we are abusing $\mathrm{fv}()$. When applied over actions we assume it returs the **vector** of sent/received variables, while when applied in other contexts (for instance to conditions) we assume a **set** of variables is returned.

The asynchronous behaviour of a guarded CFSM is obtained by considering configurations only that now, buffers hold not only the message but also the free variables along with the condition that labeled the transition that enqueued such a message in the buffer.

Definition 71 (Semantics of guarded communicating systems)**.** *The* configuration *of a guarded 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 $w_{\mathsf{pq}} \in (\mathsf{Msg} \times 2^{\mathcal{V}} \times \mathbb{G})^*$. A configuration $s' = (\vec{q'}, \vec{w'})$ is* reachable *from another configuration $s = (\vec{q}, \vec{w})$ by the* firing of the transition τ *(written $s \xrightarrow{\tau} s'$) if there exists $\mathsf{m} \in \mathsf{Msg}$ such that either:*

Snd $\tau = (q_{\mathsf{p}}, \mathsf{pq}!\mathsf{m}\langle\vec{x}\rangle, q'_{\mathsf{p}}) \in \delta_{\mathsf{p}}$ *and*

(a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ *for all* $\mathsf{p}' \neq \mathsf{p}$*; and*

(b) $w'_{\mathsf{pq}} = w_{\mathsf{pq}} \cdot \langle \mathsf{m}\langle\vec{x}\rangle, \mathcal{G}(\tau)\rangle$ *and* $w'_{\mathsf{p'q'}} = w_{\mathsf{p'q'}}$ *for all* $\mathsf{p'q'} \neq \mathsf{pq}$*; or*

Rcv $\tau = (q_{\mathsf{q}}, \mathsf{pq}?\mathsf{m}(\vec{y}), q'_{\mathsf{q}}) \in \delta_{\mathsf{q}}$ *and*

(a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ *for all* $\mathsf{p}' \neq \mathsf{q}$*; and*

(b) $\langle \mathsf{m}\langle\vec{x}\rangle, \alpha\rangle \cdot w'_{\mathsf{pq}} = w_{\mathsf{pq}}$ *and* $w'_{\mathsf{p'q'}} = w_{\mathsf{p'q'}}$ *for all* $\mathsf{p'q'} \neq \mathsf{pq}$

(c) $\alpha \wedge \vec{x} = \vec{y} \vDash \mathcal{G}(\tau)$

Rules above are analogous to the ones in Definition 60 but now we check the satisfaction of guards (Rule Rcv(c)). Note that a participan can only consume a message from the buffer if the message is what it expects, it is the intended receiver for that message and the guarantee over the exchanged values is enough to ensure the satisfaction of the assumption that the receiver has over the expected values. From this observation we can derive that a guarded communicating system whose sending guards are all equivalent to $\bot$ or whose receiveing guard are all equivalent to $\top$ is semantically equivalent to a (non guarded) communicating system. In any of these cases Condition Rcv(c) would be trivially satisfied.

Note that even though determinism required over the transitions and the conditions is so strong that in a particular GCFSM conditions get identified by the message they label, GCFSM remain interesting enough as they consitute a more precise modelling language. An example can be seen in Figure 4.8. There we show a machine that is willing to accept any incoming values greater than zero and two other machines that send values equal to

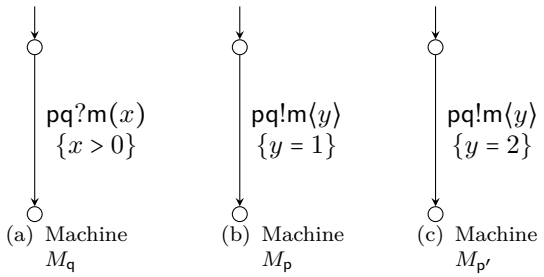


Figure 4.8: Participant q is able to compose with both versions of participant p

1 and 2 respectively meaning that machine M_{q} can be composed with both M_{p} and $M_{\mathsf{p}'}$. Once we know that data constrains are not an impediment to carry out communication we can abstract away from the details regarding data and check GMC condition on these new data spoiled systems. Note that this is possible due to the simplistic approach we took when considering values and conditions. The procedure goes as follows:

1st we check that for every action $\ell_{\mathsf{s}} = \mathsf{sr}!\mathsf{m}\langle\vec{x}\rangle \in \delta_{\mathsf{s}}$ the action $\ell_{\mathsf{r}} = \mathsf{sr}?\mathsf{m}(\vec{y})$ exists in δ_{r} and $\vec{x} = \vec{y} \wedge \mathcal{G}_{\mathsf{s}}(\ell_{\mathsf{s}}) \vDash \mathcal{G}_{\mathsf{r}}(\ell_{\mathsf{r}})$ and for every action $\ell_{\mathsf{r}} = \mathsf{sr}?\mathsf{m}(\vec{y}) \in \delta_{\mathsf{r}}$ the action $\ell_{\mathsf{s}} = \mathsf{sr}!\mathsf{m}\langle\vec{x}\rangle$ exists in δ_{s} and $\vec{x} = \vec{y} \wedge \mathcal{G}_{\mathsf{s}}(\ell_{\mathsf{s}}) \vDash \mathcal{G}_{\mathsf{r}}(\ell_{\mathsf{r}})$ and

2nd if the previous step succeeds then we project regular CFSM by removing all data and conditions and we check GMC over this new system.

Soundness of the previous procedure relies heavily on the fact that every time a message is sent or received over a channel the condition enqueued in the buffer and the reception condition will be the same. Thus allowing us to establish whether a message may be consumed according to data in a *static* way. Once all the conditions have been checked, if all the implications hold, then we are guaranteed that whenever the sytem arrives to a state where a machine would be able to consume a message from a buffer according to pure CFSMs semantics, then it will actually be able to do it as conditions will hold. On the contrary if some implication does not pass the static check then we know for sure at when the time comes, the receiver will not be able to consume that message from the buffer. Therefore this is sufficient to rule

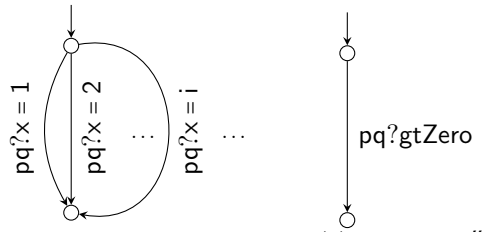
out this system as it is non GMC[3].

This rather simple mechanism allows us to obtain a class of machines strictly more expressive (from a modelling point of view) than pure CFSMs while maintainig the goodness of GMC check in terms of decidability and complexity. The reason why these GCFSM are strictly more expressive is that pure CFSMs are included into GCFSMs. As we said before, it suffices to set every reception condition to $\top$ and/or every sending condition to $\bot$ to obtain the exact same class of machines than pure CFSMs. On the other hand a machine like the one depicted in Figure 4.8(a) cannot be accurately modeled in pure CFSMs. An attempt to do it may end up in machines like the ones depicted in Figure 4.9(a) and Figure 4.9(b). The machine in Figure 4.9(a) mimics the semantic of Figure 4.8(a) by having an inifnite set of transitions. On the one hand this may not be a solution for any condition that one can imagine. Even when restricting the language of conditions to decideable fragments it is rather impractical to have to rely on an infinite structure. On the other hand, the machine in Figure 4.9(a) is not semmantically equivalent to the one in Figure 4.8(a). It is rather easy to see that M'_{q} will not validate GMC conditions when trying to compose it with a machine with just one send (for example a machine that sends $\mathsf{pq!x = 1}$). Another attempt may be to abstract away from the conditions and express them somehow in message names. An example of this is Figure 4.9(b). This abstraction forces us to abstract away conditions in both M_{p} and $M_{\mathsf{p}'}$ from Figure 4.8 in order to match the message name *gtZero*. This eliminates the differences between M_{p} and $M_{\mathsf{p}'}$ to once again leave us with an inaccurate model with respect to the original ones.

4.7 More complex ways of considering data

In this section we consider other possible ways of working with data by progressively lifting some of the restrictions imposed to GCFSMs. In the first place, as for basic communicating system, we introduce the notion of ideal, synchronous behaviour. Interactions between participants are described as synchronous interaction labels of the form $\mathsf{s} \twoheadrightarrow \mathsf{r} : \langle \mathsf{m}\langle\vec{x}\rangle(\vec{y}), \alpha \implies \beta\rangle$, with

[3]One of the conditions of GMC is representability. One of the requirements for representability is that the language of every machine is equal to the language of the projection of the synchronous system to that machine. One should note that if a reception cannot be excercised in the asynchronous system then it cannot be excercised in the synchronous system either. Therefore if a reception cannot be excercised in the asynchronous system, representability will not hold.



(a) Machine M'_{q} (b) Machine M''_{q}

Figure 4.9: Attempt to model GCFSMs with CFSMs

$\mathsf{s}, \mathsf{r} \in \mathsf{P}$, $\mathsf{m} \in \mathsf{Msg}$, $\vec{x} \subseteq \mathcal{V}$, $\vec{y} \subseteq \mathcal{V}$ and $\alpha, \beta \in \mathbb{G}$. We write Int_{Msg} for the set of all synchronous interaction labels.

Definition 72 (Synchronous transition system for guarded communication systems)**.** *Let $S = (G_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ be a* guarded communicating system. *The synchronous semantics of S is given by the LTS $\mathcal{S}(S) = ((Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}, Int_{\mathsf{Msg}}, \delta)$ with $\delta \subseteq (Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \times Int_{\mathsf{Msg}} \times (Q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ defined by the following rule:*

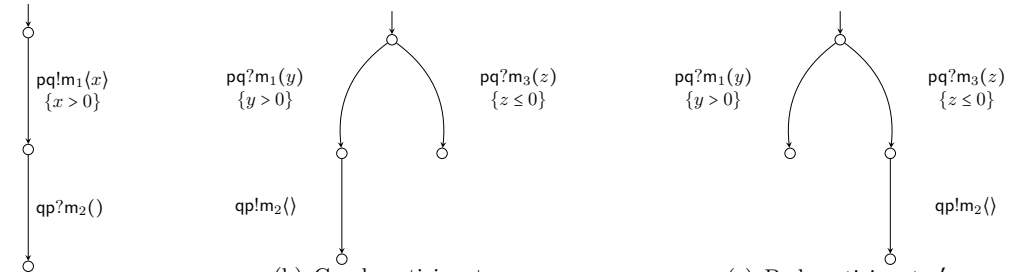
$$\frac{\mathsf{s}, \mathsf{r} \in \mathsf{P} \qquad q_{\mathsf{s}} \xrightarrow[\alpha]{\mathsf{sr!m}\langle \vec{x} \rangle} q'_{\mathsf{s}} \qquad q_{\mathsf{r}} \xrightarrow[\beta]{\mathsf{sr?m}(\vec{y})} q'_{\mathsf{r}} \qquad (\forall \mathsf{p} \notin \{\mathsf{s}, \mathsf{r}\})\ q'_{\mathsf{p}} = q_{\mathsf{p}} \qquad \alpha \wedge \left(\vec{x} = \vec{y}\right) \vDash \beta}{(q_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}} \xrightarrow[\alpha \Longrightarrow \beta]{\mathsf{s} \twoheadrightarrow \mathsf{r}:\mathsf{m}\langle \vec{x} \rangle (\vec{y})} (q'_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}}\ \textsc{Int}$$

The set of *reachable configurations of* $\mathcal{S}(S)$ is the reflexive transitive closure of δ denoted by

$$\mathbf{RS}(\mathcal{S}(S)) = \{s | (\mathbf{q_{0p}})_{\mathsf{p} \in \mathsf{P}} \rightarrow^* s\}.$$

Definition 73 defines the projection of a participan p from the synchronous system. This definition establishes that the states of a projected machine are those states from that machine that are reachable in the synchronous system. The transitions of the projection are given by those transitions that were able to synchronize.

Definition 73. *Let $S = (G_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ be a guarded communicating system then, the* projection of q from $\mathcal{S}(S)$ *is the GCFSM $G \upharpoonright_{\mathsf{q}} = \langle Q \upharpoonright_{\mathsf{q}}, q_0 \upharpoonright_{\mathsf{q}}, \delta \upharpoonright_{\mathsf{q}}, \mathcal{G} \upharpoonright_{\mathsf{q}} \rangle$ where:*



(a) Participant p

(b) Good participant q

(c) Bad participant q′

Figure 4.10: Example of wanted allowed and disallowed prunings.

- $Q\upharpoonright_{\mathsf{q}} = \left\{ q\upharpoonright_{\mathsf{q}} \;\middle|\; q \in \mathbf{RS}(\mathcal{S}(S)) \right\}$
- $\delta\upharpoonright_{\mathsf{q}}$ *is given by the following rules:*

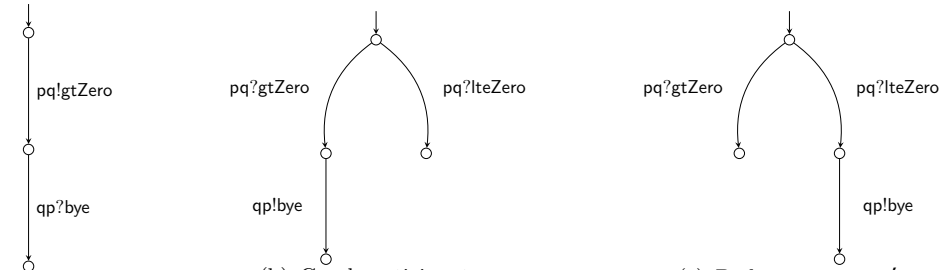
$$\frac{(q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \xrightarrow[\alpha \Longrightarrow \beta]{\mathsf{s}\twoheadrightarrow\mathsf{r}:\mathsf{a}\langle\vec{\mathsf{x}}\rangle(\vec{\mathsf{y}})} (q'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \in \mathcal{S}(S) \qquad \mathsf{q} = \mathsf{s}}{\tau = (q_{\mathsf{s}}, \mathsf{sr!a}\langle\vec{\mathsf{x}}\rangle, q'_{\mathsf{s}}) \in \delta\upharpoonright_{\mathsf{q}} \qquad \mathcal{G}\upharpoonright_{\mathsf{q}}(\tau) = \alpha}\ \text{E-Send}$$

$$\frac{(q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \xrightarrow[\alpha \Longrightarrow \beta]{\mathsf{s}\twoheadrightarrow\mathsf{r}:\mathsf{a}\langle\vec{\mathsf{x}}\rangle(\vec{\mathsf{y}})} (q'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \in \mathcal{S}(S) \qquad \mathsf{q} = \mathsf{r}}{\tau = (q_{\mathsf{r}}, \mathsf{sr?a}(\vec{\mathsf{x}}), q'_{\mathsf{r}}) \in \delta\upharpoonright_{\mathsf{q}} \qquad \mathcal{G}\upharpoonright_{\mathsf{q}}(\tau) = \beta}\ \text{E-Rcv}$$

Note 2. *The projected machines are submachines of the orginal ones. That means that the set of states of a projected machine is a subset of the set of states of the original machine and the set of transitions between two states is also a subset of the set of transitions between those two same states in the original machine.*

When analysing the ideal synchronous behaviour of a communicating system that involves guarded CFSMs, it becomes clear that some transitions of the GCFSMs may be forbidden by data constraints. Hence, the behaviour of a GCFSM that is relevant in a communicating system might be obtained by pruning those transitions that cannot be exercised during interaction due to data inconsistency.

Unlike what is prescribed by GMC conditions we would like to allow some transitions to not being excercised during communication due to data restrictions as long as this does not provoke communication errors. Because



(a) Participant p (b) Good participant q (c) Bad participant q′

Figure 4.11: Pure CFSM version of wanted allowed and disallowed prunnings.

of the asynchronous nature of the communication the only transitions that may fail to excercise without producing an immediate communication error are receptions. This is like that because if we would like to prevent a sending from happenning we would need to have some notion of what is expected in the receiver therefore breaking asynchroncity.

The question that we need to answer is when the impossibility to excercise some action does not imply a communication problem but rather a valid choice that could still lead to successful executions of the protocol. An example of this situation is given in Figure 4.10[45]. In this example participant p in Figure 4.10(a) should be allowed to interact with participant q of Figure 4.10(b) as it is clear that the interaction between them both can only lead to successful executions, while on the other hand we would like to be able to rule out the system formed by p and participant q′ in Figure 4.10(c) since it is obvious that the interaction between them would lead to a deadlock configuration. An equivalent situation can be formulated in terms of pure CFSMs as it is shown in Figure 4.11. The reader should be aware that neither the system formed by CFSMs in Figure 4.11(a) and Figure 4.11(c) nor the system formed by CFSMs in Figure 4.11(a) and Figure 4.11(b) pass the GMC check although it is clear that again the system formed by p and q has no communication problems.

[4]In Figure 4.10 and all subsequent figures where we show GCFSM we assume that when no condition is shown it is equivalente to $\top$ in the case of a reception transition and to $\bot$ in the case of a sending transition.

[5]In the figures of this section every sink state is considered final even though they are not marked.

This is why we want to introduce the notion of an error free choreography that takes into consideration valid prunnig of actions according to data. In order to do this we first present a similar notion but for pure CFSMs (i.e. without data).

4.7.1 Bisimulation up-to receptions

The intuitive situation that we want to capture is that in a communicating system, reception transitions may be cut off from communicating machines without modifying the behavior of the system. This situation is exhibited in Figure 4.11. Note that sending transitions may also be cut off without altering the behavior of the system when their source state becomes unreachable due to a reception transition been cut off. Summarizing sending transitions whose source state is reachable must not be cut off.

In this section we define *bisimulation up-to receptions*, a relation between two machines and we conjecture that when this relation holds between every machine of a communicating system and their projections obtained from the synchronous system then, the (asynchronous) behavior of the original system and the behavior of the system formed by the projections is equivalent (bisimilar).

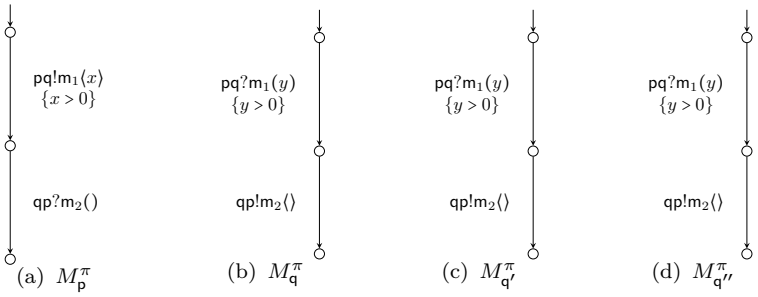
First we give some instrumental definitions.

Definition 74 (Reachable configurations)**.** *Given communicating system S we define $\mathbf{RS}_k$ as the maximal set such that $\mathbf{RS}_k = \{(\vec{q},\vec{w}) \mid \exists(\vec{q_0},\vec{\epsilon}) \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_k} (\vec{q_k},\vec{w_k}) = (\vec{q},\vec{w})\}$.*
We define $\mathbf{RS} = \bigcup_{k\geq 0} \mathbf{RS}_k$.

Note 3. *The set of 0-reachable configurations is the set that has as the only one element the initial configuration $\mathbf{RS}_0 = \{(\vec{q_0},\vec{\epsilon})\}$*

A configuration q is k-reachable (or $q \in \mathbf{RS}_k$) if there exists a path of length k from the initial configuration to q. Note that although the LTS of the asynchronous behavior is infinite every reachable configuration is k-reachable for some k (i.e. the state is reachable through a finite path). From now on when we consider configurations of a system we consider only reachable configurations.

An example of the situation we want to handle is depicted in Figure 4.15. There we show five machines. Let's first consider three different guarded communicating systems: (1) the one formed by p and q, (2) the one formed by p and q' and (3) the one formed by p and q''. The three of them present



(a) M_{p}^{π} (b) M_{q}^{π} (c) $M_{\mathsf{q}'}^{\pi}$ (d) $M_{\mathsf{q}''}^{\pi}$

Figure 4.12: Example of projected machines from systems (1)–(3).

the same *effective behaviour* (i.e. the three of them lead to the same asynchronous system). Application of the procedure described in Section 4.6.2 would rule out systems (2) and (3) because there is no action $\mathsf{pq!m_3}\langle u\rangle$ in M_{p}.

From the following systems: (4) the one formed by p' and q, (5) the one formed by p' and q' and (6) the one formed by p' and q'' the only one that is free from communication errors is (5). One should note that bisimulation up-to receptions is a relation that is parametric in the communicating system. This means that two machines may be bisimilar up-to receptions *with respect to a set of machines or participants* and not on their own.

In Figure 4.12 we show the projections of M_{p}, M_{q}, $M_{\mathsf{q}'}$ and $M_{\mathsf{q}''}$ from Figure 4.15 in the systems of items (1) through (3). Note that the three projections of q, q' and q'' are exactly the same. In Figure 4.13 we show the projected machines from the system in item (4). Note that these projections are exactly the same as the ones shown in Figure 4.12(a) and Figure 4.12(b). Yet it is rather clear that in this case the asynchronous behaviour of $M_{\mathsf{p}'}$ and $M_{\mathsf{p}'}^{\pi}$ differs. This situation should also be identified by the relation of bisimulation up-to receptions (i.e. $M_{\mathsf{p}'}$ and $M_{\mathsf{p}'}^{\pi}$ should not be regarded as bisimilar up-to receptions in this system). Finally in Figure 4.14 we show the projections of $M_{\mathsf{p}'}$, $M_{\mathsf{q}'}$ and $M_{\mathsf{q}''}$ in the systems of items (5) and (6). One should note that $M_{\mathsf{p}'} = M_{\mathsf{p}'}^{\pi}$ and $M_{\mathsf{q}'} = M_{\mathsf{q}'}^{\pi}$ but $M_{\mathsf{q}''} \not= M_{\mathsf{q}''}^{\pi}$. The prunning occured in the projection of $M_{\mathsf{q}''}$ should once again force us to rule out this system.

Definition 75 (Simulation up-to receptions)**.** *Given two communicating machines* $M = (\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *and* $M' = (\mathsf{Q}', q_0', \mathsf{Msg}, \delta')$ *we say that* $M \overset{?}{\leq} M'$ *(*M' *simulates* M *up-to receptions) if there exists* $R \subseteq \mathsf{Q} \times \mathsf{Q}'$ *such that*

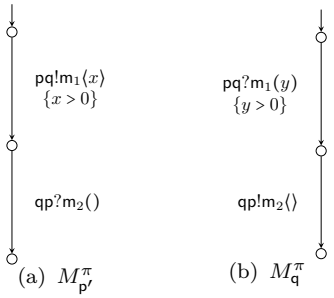


Figure 4.13: Example of projected machines from system (4).

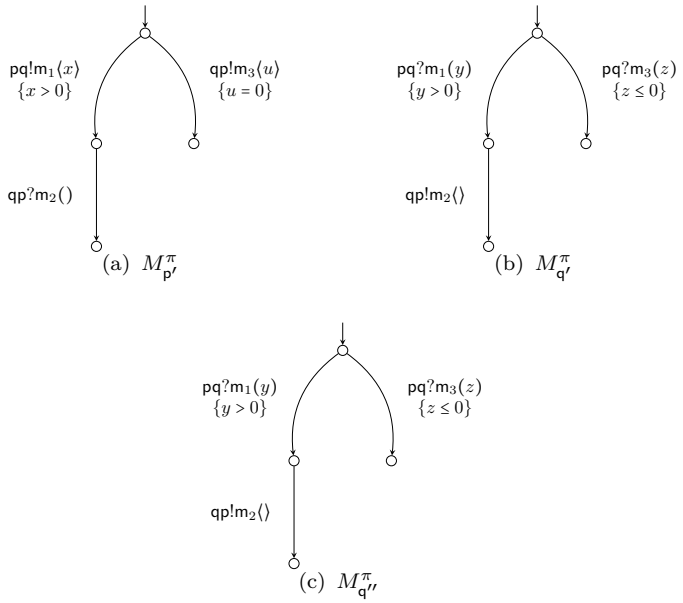


Figure 4.14: Example of projected machines from systems (5) and (6).

$(q_0, q_0') \in R$ *and for all* $(q, q') \in R$ *the following holds:*

$$\forall (q, \mathsf{pq!a}, q_1) \in \delta, (q', \mathsf{pq!a}, q_1') \in \delta' \wedge (q_1, q_1') \in R$$

and

$$\forall (q, \mathsf{pq?a}, q_1) \in \delta, \text{ if } (q', \mathsf{pq?a}, q_1') \in \delta' \text{ then } (q_1, q_1') \in R.$$

The relation R is said to be a simulation up-to receptions relation. If both R and R^{-1} are simulations up-to receptions then $M \overset{?}{\approx} M'$ (M and M' are bisimilar up-to receptions).

Note that we do not force every reception to be simulated, only those that are actually present.

Note 4. *A (plain) simulation relation is also a simulation up-to receptions. Therefore every machine M_p in a communicating system S simulates up-to receptions its own projection $M\!\upharpoonright_\mathsf{p}$.*

Now we enunciate the main property of this section. This property is left as a conjecture since some steps in the proof are missing.

Conjecture 1 (Bisimulation up-to receptions and GMC)**.** *Given a communicating system $(M_\mathsf{p})_{\mathsf{p} \in \mathsf{P}} = S$ and its synchronous system $\mathcal{S}(S)$ if for all $\mathsf{p} \in \mathsf{P}$ $M_\mathsf{p} \overset{?}{\approx} M\!\upharpoonright_\mathsf{p}$ and $S\!\upharpoonright$ is GMC then $\mathcal{A}(S) \approx \mathcal{A}(S\!\upharpoonright)$ where $S\!\upharpoonright \;=\; (M\!\upharpoonright_\mathsf{p})_{\mathsf{p} \in \mathsf{P}}$*

Sketch Proof. First remember that every machine M_p simulates its projection $M\!\upharpoonright_\mathsf{p}$ therefore it is immediate that $\mathcal{A}(S)$ simulates $\mathcal{A}(S\!\upharpoonright)$. Now we proceed by giving the relation R as the maximal relation between configurations of $\mathcal{A}(S)$ and $\mathcal{A}(S\!\upharpoonright)$ such that R is a simulation relation from $\mathcal{A}(S)$ to $\mathcal{A}(S\!\upharpoonright)$. Note that this relation exists since $\mathcal{A}(S)$ simulates $\mathcal{A}(S\!\upharpoonright)$. Next we need to prove that R^{-1} is also a simulation.

To do that we need to show that for all pair of configurations $\left((\vec{q}, \vec{w}), (\vec{q}\!\upharpoonright, \vec{w}\!\upharpoonright)\right)$ in R if $(\vec{q}, \vec{w}) \xrightarrow{\tau} (\vec{q'}, \vec{w'})$ in $\mathcal{A}(S)$ then $(\vec{q}\!\upharpoonright, \vec{w}\!\upharpoonright) \xrightarrow{\tau} (\vec{q'}\!\upharpoonright, \vec{w'}\!\upharpoonright)$ in $\mathcal{A}(S\!\upharpoonright)$ and $\left((\vec{q'}, \vec{w'}), (\vec{q'}\!\upharpoonright, \vec{w'}\!\upharpoonright)\right) \in R$

We split the proof in cases:

a) $\tau = (q, \mathsf{sr!m}, q')$ If this transition is present in $\mathcal{A}(S)$ then $(q_\mathsf{s}, \mathsf{sr!m}, q_\mathsf{s}') \in \delta_\mathsf{s}$. But then, since $M_\mathsf{s} \overset{?}{\approx} M\!\upharpoonright_\mathsf{s}$ and as $q\!\upharpoonright_\mathsf{s}$ is reachable in $M\!\upharpoonright_\mathsf{s}$, then $(q\!\upharpoonright_\mathsf{s}, \mathsf{sr!m}, q'\!\upharpoonright_\mathsf{s}) \in \delta\!\upharpoonright_\mathsf{s}$. Therefore $(\vec{q'}\!\upharpoonright, \vec{w'}\!\upharpoonright) \in \mathbf{RS}(\mathcal{A}(S\!\upharpoonright))$. Because determinism in the machines both $(\vec{q'}, \vec{w'})$ and $(\vec{q'}\!\upharpoonright, \vec{w'}\!\upharpoonright)$ are unique and for that reason we know the pair is in R.

b) $\tau = (q, \mathsf{sr}?\mathsf{m}, q')$ This case is subsequently splitted into two other cases. In the first one, when $(q\!\upharpoonright_{\mathsf{r}}, \mathsf{sr}?\mathsf{m}, q'\!\upharpoonright_{\mathsf{r}}) \in \delta\!\upharpoonright_{\mathsf{r}}$. In this case we argue in the same way than before that because of determinism, bisimulation up-to receptions and the fact that $\mathcal{A}(S)$ simulates $\mathcal{A}(S\!\upharpoonright)$ that $\left((\vec{q'}, \vec{w'}), (\vec{q'}\!\upharpoonright, \vec{w'}\!\upharpoonright)\right) \in R$.

The second case is when $(q\!\upharpoonright_{\mathsf{r}}, \mathsf{sr}?\mathsf{m}, q'\!\upharpoonright_{\mathsf{r}}) \notin \delta\!\upharpoonright_{\mathsf{r}}$ at this point we would like to show that this is an impossible situation as a consequence of the assumption that all the projections are bisimilar up-to receptions with the original ones. Unlike the sending case this is not direct since we are actually allowing some receptions to be missing in the projections. The argument would go by taking into account that we do not allow ANY reception to be missing but only those that failed to execute synchronously. Furthermore, we are only interested in proving this when the projected system is GMC. This assumption guarantees that that $\mathcal{A}(S\!\upharpoonright)$ is free from communication errors. We believe that this should allow us to argue that if the reception arrow is missing, then there is a reachable sending arrow in the sending machine that must have been cut off therefore contradicting the assumption that the machines are bisimilar up-to receptions.

□

If Conjecture 1 is valid then we could use it to allow pruning some transitions. We could relax the 1st step of the procedure given in Section 4.6.2 by not requiring that every reception transition has a corresponding sending transition. Then we could remove data and check GMC over the projections of the system without data.

Once again we bring to the attention of the reader that we are forced to deal with deterministic CFSMs (or GCFSMs) in order to restrain ourselves to the conditions given in [38]. As you may have notice this results in a weird modelling when considering data associated to transitions. For example, in Figure 4.15(d) we are forced to use different messages for each transition that leaves the initial state. A more interesting model would be one in which we can have some kind of non-determinism that is resolved (i.e. determinized) by data conditions. As an example we could think of replacing $\mathsf{pq}?\mathsf{m}_1(y)\{y > 0\}$ and $\mathsf{pq}?\mathsf{m}_3(z)\{z \leq 0\}$ with two transitions labelled with the same message m like so $\mathsf{pq}?\mathsf{m}(y)\{y > 0\}$ and $\mathsf{pq}?\mathsf{m}(z)\{z \leq 0\}$. Note that in this case the determinism would be imposed by the fact that the conditions are mutually exclusive (i.e. $(x > 0) \Leftrightarrow \neg(x \leq 0)$).

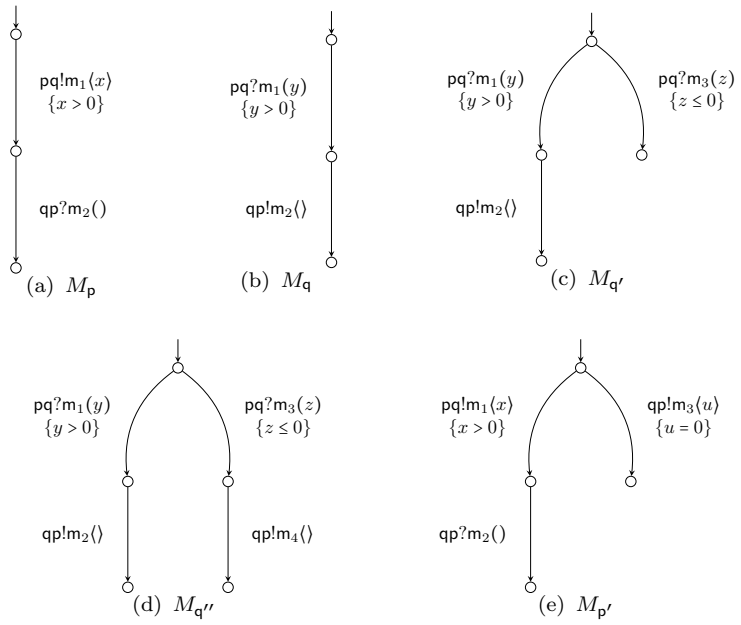


Figure 4.15: Example of bisimulation up-to receptions with data.

4.8 Moving away from local-only data

An even more complex and interesting use of data would be if some kind of relations between variables could be expressed. With this we could express richer contracts. For example a machine could send the result of applying a function to a previously received value. When considering this idea it is clear that one can rapidly fall into an undecidable model as a consequence of loops undermining the possibility to automatically determine binding in runtime. In this section we present a set of definitions trying to give the possibility to express some relations between variables while maintaining the decidability.

First we modify GCFSMs slightly by removing the restriction that conditions should predicate only over the values exchanged in their transitions. In this way we allow conditions to establish relations between variables exchanged in different transitions. The semantics in this new setting is given in terms of *incarnations* of variables. This means that each time a transition es executed we consider a new incarnation for each variable that is sent or received. In this context sentences are evaluated over the last incarnation of each variable. In order to prevent an infinite state explosion when consid-

ering the synchronous behavior we add the additional restriction that there are no transitive dependencies. This means that the "value" of a variable at any point of the execution is independent of the previous values that it may have taken in the past. This restriction is enforced even for transitive dependencies meaning that the value of an incarnation v_i does not depend on the value of v_j for any $j < i$ and there is no other incarnation v'_k such that the value of v_i depends on v'_k and the value of v'_k depends on the value of v_j for some $j < i$. Additionally we require that the validity of a sentence depends on, at most, one incarnation of each variable. The argument for these restrictions is that in this way one cannot build a counter with the conditions.

A set of auxiliary definitions is given in Definition 76–Definition 80 that will serve us to formally define the new semantics. First definition allows us to incarnate (rename) free variables in a sentence. The next two definitions formalize the notion of knowledge associated to a variable and its closure (Definition 77) and the notion of knowledge associated to a vector of variables and its closure (Definition 78)

Definition 76 (Instantiation function)**.** *The* instantiation *function* $inst : \mathbb{G} \times (\mathcal{V} \to \mathbb{N}) \to \mathbb{G}$ *is a function that, given a sentence* $\alpha \in \mathbb{G}$ *and an incarnation function* $\mathcal{I} : \mathcal{V} \to \mathbb{N}$ *returns the rewriting of the sentence* α *that is obtained by replacing each free variable* x *with* x_i *where* i *is the incarnation corresponding to variable* x *according to* $\mathcal{I}$*:*

$$inst(\alpha, \mathcal{I}) = \alpha[x_{\mathcal{I}(x)}/x]_{x \in \mathrm{fv}(\alpha)}$$

Definition 77. *Given a function* $G : \mathcal{V} \to \mathbb{G}$ *we define* $G^* : \mathcal{V} \to 2^{\mathbb{G}}$ *as*

$$G^*(x) = \bigcup_{i \in \mathbb{N}} G_i(x)$$

where

$$\begin{aligned} G_0(x) &= \{G(x)\} \\ G_i(x) &= \{G(x') \mid x' \in fv(G_{i-1}(x))\} \end{aligned}$$

Definition 78. *The function* $\mathcal{K} : 2^{\mathcal{V}} \times (\mathcal{V} \to \mathbb{N}) \times (\mathcal{V} \to \mathbb{G}) \to (\mathcal{V} \to \mathbb{G})$ *represents the knowledge closure function. Given a set of variables* $\vec{x}$ *and a mapping* $G : \mathcal{V} \to \mathbb{G}$ *it returns the sentences that are directly or indirectly involved in the definition of* $\vec{x}$*:*

$$\mathcal{K}(\vec{x}, \mathcal{I}, G) = \kappa : \mathcal{V} \to \mathbb{G}$$

such that

$$(\forall v \in \mathcal{V})\kappa(v) = \begin{cases} \top & v \notin \{x_{\mathcal{I}(x)} \mid x \in \vec{x}\} \\ \bigwedge G^*(v) & v \in \{x_{\mathcal{I}(x)} \mid x \in \vec{x}\} \end{cases}$$

Note 5. *We denote as* $len(\vec{x})$ *to the length of the vector of variables* $\vec{x}$*.*

Note 6. *We denote as* $\vec{x}[i]$ *(with* $0 \leq i < len(\vec{x})$*) to the* i*-th variable in* $\vec{x}$*.*

The following two definitions formalize the notion of mapping from variables to sentences and the notion of update-by-conjunction through adding a new sentence (Definition 79) or through the "union" of two different mappings (Definition 80). These definitions will be useful to express how the context evolves as a result of the execution of the GCFSMs.

Definition 79. *Given a function* $G : \mathcal{V} \to \mathbb{G}$ *and a sentence* $\gamma \in \mathbb{G}$*, we define the function* $\wedge : (\mathcal{V} \to \mathbb{G}) \times \mathbb{G} \to (\mathcal{V} \to \mathbb{G})$ *as:*

$$\wedge(G, \gamma) = G'$$

such that

$$(\forall v \in \mathcal{V})G'(v) = \begin{cases} G(v) & v \notin \mathrm{fv}(\gamma) \\ G(v) \wedge \gamma & v \in \mathrm{fv}(\gamma) \end{cases}$$

Definition 80. *Given a function* $G : \mathcal{V} \to \mathbb{G}$ *and a function* $\kappa : \mathcal{V} \to \mathbb{G}$*, we define the function* $\wedge : (\mathcal{V} \to \mathbb{G}) \times (\mathcal{V} \to \mathbb{G}) \to (\mathcal{V} \to \mathbb{G})$ *as:*

$$\wedge(G, \kappa) = G'$$

such that

$$(\forall v \in \mathcal{V})G'(v) = G(v) \wedge \kappa(v)$$

We are now in position to give the semantics of a *communicating system* for GCFSMs in this new setting.

As before we straightforwardly extend the definition of *communicating system* to consider GCFSMs. Note that we keep the restriction that a GCFSM use only local variables. That means that the sets of free variables of every pair of GCFSMs in a system is disjoint.

Now the asynchronous behavior of a GCFSM is obtained by considering configurations with two extra components. One of them describes the conditions of the transitions already executed in the path to a particular

configuration. This extra component is called *context*. The other component maintains the last incarnation used of each variable. Note that since a variable can only be sent or received in one transition, the incarnation index of a variable also denotes the number of times a particular transition was executed.

Definition 81 (Semantics of *communicating system* for GCFSMs)**.** *Let* $S = (M_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}$ *be a communicating system. A* configuration *of* S *is a 4-uple* $s = \langle \mathbf{G} \ ; \ \mathbf{q} \ ; \ \mathbf{w} \ ; \ I \rangle$ *where* $\mathbf{G} = (G_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}$ *with* $G_{\mathsf{p}} : \mathcal{V} \to \mathbb{G}$, $\mathbf{q} \in (Q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}$, $\mathbf{w} = (w_{\mathsf{pq}})_{\mathsf{pq}\in C}$ *with* $w_{\mathsf{pq}} \in \Sigma^*$ *and* $\mathcal{I} : \mathcal{V} \to \mathbb{N}$. *We write* C_S *to denote the set of all possible configurations of* S. *The semantics of* S *is given by the LTS* $\mathcal{A}(S) = (\mathsf{C}_S, Act_\Sigma, \delta)$ *with* $\delta \subseteq \mathsf{C}_S \times Act_\Sigma \times \mathsf{C}_S$ *defined by the following rules:*

$$\frac{\begin{array}{l}\mathsf{s} \in \mathsf{P} \quad q_{\mathsf{s}} \xrightarrow[\alpha]{\mathsf{sr!a}\langle\vec{\mathsf{x}}\rangle} q'_{\mathsf{s}} \qquad w'_{\mathsf{sr}} = w_{\mathsf{sr}}.\langle a\langle\vec{x}\rangle, \mathcal{K}(\vec{x}, \mathcal{I}', G'_{\mathsf{s}})\rangle \\ G'_{\mathsf{s}} = G_{\mathsf{s}} \wedge \iota_\alpha \qquad (\forall v \in \mathcal{V}) G'_{\mathsf{s}}(v) \not\equiv \bot \qquad \mathcal{I}' = \mathcal{I}[x \mapsto \max(\mathcal{I}) + 1]_{x\in\vec{x}} \\ (\forall \mathsf{p} \neq \mathsf{s}) q'_{\mathsf{p}} = q_{\mathsf{p}} \qquad (\forall \mathsf{p} \neq \mathsf{s}) G'_{\mathsf{p}} = G_{\mathsf{p}} \qquad (\forall c \neq \mathsf{sr}) w'_c = w_c\end{array}}{\langle (\mathbf{G}_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{q}_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{w}_{\mathbf{c}})_{\mathbf{c}\in\mathbf{C}} \ ; \ I\rangle \xrightarrow{\mathsf{sr!a}\langle\vec{\mathsf{x}}\rangle} \langle (\mathbf{G}'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{q}'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{w}'_{\mathbf{c}})_{\mathbf{c}\in\mathbf{C}} \ ; \ I'\rangle}\ \textsc{Send}$$

$$\frac{\begin{array}{l}\mathsf{r} \in \mathsf{P} \quad q_{\mathsf{r}} \xrightarrow[\beta]{\mathsf{sr?a}(\vec{\mathsf{y}})} q'_{\mathsf{r}} \qquad w_{\mathsf{sr}} = \langle a\langle\vec{x}\rangle, \kappa\rangle.w'_{\mathsf{sr}} \qquad \mathcal{I}' = \mathcal{I}[y \mapsto \max(\mathcal{I}) + 1]_{y\in\vec{y}} \\ G'_{\mathsf{r}} = (G_{\mathsf{r}} \wedge \kappa) \wedge \left(\vec{x} = \vec{y}\right) \qquad (\forall v \in \mathcal{V}) G'_{\mathsf{r}}(v) \not\equiv \bot \qquad \gamma \vDash \iota_\beta \\ (\forall \mathsf{p} \neq \mathsf{r}) q'_{\mathsf{p}} = q_{\mathsf{p}} \qquad (\forall \mathsf{p} \neq \mathsf{r}) G'_{\mathsf{p}} = G_{\mathsf{p}} \qquad (\forall c \neq \mathsf{sr}) w'_c = w_c\end{array}}{\langle (\mathbf{G}_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{q}_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{w}_{\mathbf{c}})_{\mathbf{c}\in\mathbf{C}} \ ; \ I\rangle \xrightarrow{\mathsf{sr?a}(\mathbf{x})} \langle (\mathbf{G}'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{q}'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \ ; \ (\mathbf{w}'_{\mathbf{c}})_{\mathbf{c}\in\mathbf{C}} \ ; \ I'\rangle}\ \textsc{Rcv}$$

where

$$\begin{aligned}\iota_\alpha &= inst(\alpha, \mathcal{I}') \\ \iota_\beta &= inst(\beta, \mathcal{I}') \\ \gamma &= \bigwedge_{v\in \mathrm{fv}\left(\iota_\beta\right)} G'^{*}_{\mathsf{r}}(v)\end{aligned}$$

The inference rules just defined are analogous to the ones in Definition 71 only that now we gather the conditions during execution (3rd premise of the

rule SEND and 4th premise of the rule RCV). The main difference between the rules SEND and RCV resides in the way in which message "parameters" are handled. In both cases variables are re-incarnated hiding their restrictions that were previously gathered (which is particularly useful to handle machines with cycles). The way in which conditions are used is what distinguishes both rules. In the case of sending transitions the condition is only checked to guarantee that knowledge will not become contradictory trivializing all subsequent checks. No further requirements are imposed over sends. On the other hand, in the case of receptions, the machines joins together the information in the buffer with its own context to verify that the condition of the reception is deductible from that information. Only if this is the case a reception is enabled.

Same as before we introduce the notion of ideal synchronous behavior. We adopt the following notation: $\{G\}$ is the set of all contexts and $\{\mathcal{I}\}$ is the set of all incarnation functions.

Definition 82 (Synchronous semantics)**.** *Let* $S = (M_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}$ *be a* communicating system *for GCFSMs. The synchronous semantics of* S *is given by the LTS* $\mathcal{S}(S) = ((\{G\} \times \{\mathcal{I}\}) \times (Q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}, Int_{\Sigma}, \delta)$ *with* $\delta \subseteq (\{G\} \times \{\mathcal{I}\}) \times (Q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}) \times Int_{\Sigma} \times (\{G\} \times \{\mathcal{I}\}) \times (Q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}$ *defined by the following rule:*

$$\frac{\mathsf{s},\mathsf{r}\in\mathsf{P} \qquad q_{\mathsf{s}} \xrightarrow[\alpha]{\mathsf{sr!a}\langle\vec{\mathsf{x}}\rangle} q'_{\mathsf{s}} \qquad q_{\mathsf{r}} \xrightarrow[\beta]{\mathsf{sr?a}(\vec{\mathsf{y}})} q'_{\mathsf{r}} \qquad (\forall \mathsf{p}\notin\{\mathsf{s},\mathsf{r}\})\ q'_{\mathsf{p}} = q_{\mathsf{p}} \qquad \mathcal{I}' = \mathcal{I}[x \mapsto \max(\mathcal{I})+1]_{x\in\vec{x}} \qquad G' = G \wedge \iota_{\alpha} \qquad \gamma \not\vdash \bot \qquad \gamma \vdash \iota_{\beta}}{\langle [(G,\mathcal{I})]\ ;\ (q_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}\rangle \xrightarrow{\mathsf{s}\to\mathsf{r}:\mathsf{a}} \langle [(G',\mathcal{I}')]\ ;\ (q'_{\mathsf{p}})_{\mathsf{p}\in\mathsf{P}}\rangle}\ \textsc{Int}$$

where

$$\iota_{\alpha} = inst(\alpha,\mathcal{I}')$$
$$\iota_{\beta} = inst(\beta,\mathcal{I}')$$
$$\gamma = \bigwedge_{v\in \mathrm{fv}(\iota_{\beta})} G'^{*}(v)$$

and where $[(G',\mathcal{I}')]$ is the equivalence class given by the equivalence relation in Definition 83.

Note that now the construction of the synchronous system depends on the ability to compute the equivalence class of a state. We need this equivalence criterion to be correct with respect to the semantics given to GCFSMs

in this section, meaning that if two states are regarded as equivalent, then the enabled paths from them are the same. Even more we need that the for any given system, the amount of equivalence classes is finite in order to guarantee that the synchronous system has a finite amount of state and therefore can be actually constructed.

Definition 83. *Given* $G : \mathcal{V} \to \mathbb{G}$, $G' : \mathcal{V} \to \mathbb{G}$, $\mathcal{I} : \mathcal{V} \to \mathbb{N}$ *and* $\mathcal{I}' : \mathcal{V} \to \mathbb{N}$ *we say that*

$$(G, \mathcal{I}) \equiv (G', \mathcal{I}')$$

if and only if

$$\left(\bigwedge \bigcup_{v \in \mathcal{V}} \left\{ \text{“}new\_\text{”}u_i = \text{“}old\_\text{”}u_j \mid u_i \in \mathrm{fv}\big(G'^*(v_{\mathcal{I}'(v)})\big) \text{ and } u_j \in \mathrm{fv}\big(G^*(v_{\mathcal{I}(v)})\big) \right\} \right) \Longrightarrow \left(\bigwedge G'^*(v_{\mathcal{I}'(v)})[u \mapsto \text{“}new\_\text{”}u] \Leftrightarrow G^*(v_{\mathcal{I}(v)})[u \mapsto \text{“}old\_\text{”}u] \right)$$

Essentially the equivalence criterion between contexts establishes that the knowledge on the last incarnation of each variable accumulated in each context must be equivalent. The antecedent of the implication, that from now on we call *HEq*, equalizes the last incarnation a variable in one context with the last incarnation of the same variable in the other context. It also equalizes the incarnation of a variable u in a context with the incarnation of that same variable in the other context. As a result of the restrictions imposed to conditions (no transitive dependencies, etc.) at most one incarnation of each variable may appear in the knowledge associated to a variable in any given context. Besides the evaluation of each formula in a context is carried out by replacing each variable with the last incarnation of it. This is why we equalize them in the equivalence criterion. To simplify the notation we have assumed that every variable $v_i \in \mathrm{fv}\big(G'^*(v_i)\big)$, meaning that every v_i appears free in the knowledge associated to itself.

Proposition 13. *Criterion given in Definition 83 is correct.*

Proof. The criterion is correct if $(G, \mathcal{I}) \equiv (G', \mathcal{I}')$ implies that every sentence $\alpha, \beta \in \mathbb{G}$, $(\gamma \implies \iota_\beta) \Leftrightarrow (\gamma' \implies \iota'_\beta)$ where

$$\begin{aligned} \iota_\alpha &= \mathit{inst}(\alpha, \mathcal{I}) \\ \iota_\beta &= \mathit{inst}(\beta, \mathcal{I}) \\ \gamma &= \bigwedge_{v \in \mathrm{fv}\left(\iota_\beta\right)} (G \wedge \iota_\alpha)^*(v) \end{aligned}$$

y

$$\begin{aligned}\iota'_\alpha &= inst(\alpha, \mathcal{I}')\\ \iota'_\beta &= inst(\beta, \mathcal{I}')\\ \gamma' &= \bigwedge_{v \in \mathrm{fv}(\iota'_\beta)} (G' \wedge \iota'_\alpha)^*(v)\end{aligned}$$

To prove this we consider models $\mathcal{M}$ over the set of variables {"new_"v | $v \in \mathcal{V}$} $\cup$ {"old_"v | $v \in \mathcal{V}$} and we rename the variables of the sentences appropriately. This is, the variables of γ' and β' are prefixed with "new_" and the variables of γ and β are prefixed with "old_". Moreover, for all model $\mathcal{M}$ of $(\gamma \implies \iota_\beta)$ there exists an equivalent model $\mathcal{M}'$ that satisfies *HEq* therefore in $\mathcal{M}'$, γ and γ' are equivalent as a result of the contexts been equivalent. ι_β and ι'_β are syntactically equivalent modulo variable naming. But we said that these variables were equal and thus these formulae are equivalent too. Then it is true that $(\gamma \implies \iota_\beta) \Leftrightarrow (\gamma' \implies \iota'_\beta)$. □

Conjecture 2. *The transition system in Definition 82 considered from the configuration* $\langle (\lambda v.\top, \lambda v.0)\ ;\ (q_{0\mathsf{p}})_{\mathsf{p}\in\mathsf{P}} \rangle$ *is finite.*

Proof of Conjecture 2 reduces to show that traversing two times the same path leads to equivalent configurations in the sense of Definition 83. Then, if we call simple path to a sequence of transitions that passes at most once over each cycle, since the amount of direct paths (paths without cycles) from the initial state is finite, the amount of different contexts with one can arrive at a particular state is also finite. We argue that each time a simple path is traversed to a state, since there are no transitive dependencies, the knowledge accumulated on each of the "live" variables (the last incarnation of each variable) is the same. Then, given that any sentence is evaluated exclusively over the last incarnation of the variables then the context obtained must be equivalent to some previously generated context.

The journey from here to actually extend the results of GMC to this new setting is still long. Contrary to the procedure given in Section 4.6.2 giving a condition similar to GMC is not direct here. In [32] an implementation of the algorithm to build the synchronous system is provided. The results, although encouraging show that there is still much work to do. Unlike in pure CFSMs, in this case the state explosion becomes a problem quite fast. Moreover even proving Conjecture 2 itself may require quite some work. Here we gave only the first steps towards this goal and we leave this as an open path.

4.9 Coherence and incremental binding

We pointed out two drawbacks in Section 4.4 on the use of CFSMs as labels for provides points and global graphs as labels for requires points. First, the nature of binding in SOC is incremental. This means that services are procured following an on-demand pattern. On the other hand, there is an efficient procedure to determine whether a set of CFSM will be able to interact without errors. Unfortunately this procedure does not rely on a compositional property. This means that in order to be able to apply this procedure one needs to have the CFSMs for every one of the participants in a multiparty communication. This goes directly against the on-demand nature of procuring services at runtime. Second, there is a need for some kind of conformance or coherence check between the declared behaviors in the form of CFSMs and the actual behavior that results from the automata labeling processes in ARNs. The difficulty of achieving a sound and efficient procedure may vary depending on the gap between the automata used to label ARNs' processes and CFSMs.

In this section we aim at contributing a solution to these two classes of problems at once by introducing a new kind of automata that internalizes the communication between participants by means of special actions simulating the sending and receiving of messages using internal buffers. These new automata, named asynchronous communication finite state automata, present three types of transitions: a) internal transitions that intend to model internal computation (non communication actions), b) buffer transitions that internalize the asynchronous behavior providing a way to express the communication between two parties after composition and lastly c) a third type of transitions that model the external communication actions as in CFSMs.

Composition of these automata brought the need to extend CFSMs model by allowing multichannel CFSMs, communicating machines that may have more than one channel in each direction with each participant. This extension is needed in order to express the communication interface of these automata. If one considers CFSMs, when composing two of such machines each of them may have a channel with a third party. These channels are independent, in the sense that they are different FIFO buffers. Thus to preserve semantics becomes necessary for the composition to have two channels with this third party.

In [49] Montesi and Yoshida presents a class of choreographies admitting a composition operation. This class of choreographies has two types actions

involving messages, on the one hand there are complete actions describing message exchanges such as those present in classical choreographies and partial actions which refer to messages with the outside, being the main reason why these choreographies are called partial by the authors. This notion of partiality, while close to the aim of our approach, does not succeed in characterizing the expected behavior. From the point of view of [49] partial composition of choreographies pursue the possibility of incremental composition of protocols which, when considered as a bottom-up approach, allows for a synthesis of choreographies from end-points (name given to partial choreographies only consisting of partial actions). From our point of view partial composition refer to an incremental composition of end-points (in our case described as Communicating Finite State Machines [9]) with the aim of complying with a previously fixed total choreography (in our case described with a Global Graph [19]) declaring the protocol ruling the communication over a given communication channel.

Another related work is [14] where Caires and Torres Vieira formalize the notion of conversation as a way to organize exchanges of messages in a service-oriented computing systems. A conversation is a distributed, possibly concurrent, set of interactions between several participants. Conversations can be seen as message exchanges between participants in a virtual chat room usually called conversation context. Participants in a conversation can dynamically join conversations, even if they are ongoing, by referring to conversation context identifiers that can even be passed around as messages. Conversation types serve the purpose of formalizing the discipline under which processes participate in a conversation. This perspective to formalizing the behavior of communication channels results from the composition of processes identifying participants through their role in the communication. The result is a potentially very rich orchestrated behavior while from our perspective, we aim at a combination of orchestration and choreography enabling a conformance check to be used to determine interoperability of participants through a communication channel. Moreover, conversation types do not provide any support for partial composition as judgements are decided over a fixed composition of processes.

In [41] Input/Output automata are presented. These automata resembles the model we introduce in this section. IO Automata have both internal actions and communication (input and output) actions. Yet they have some characteristics that we do not find suitable for our case. First, IO automata are input enabled, meaning that an IO automata is always allowed to receive a message. In CFSMs this is not the case. On the other hand, when

composing IO automata synchronization of shared actions is forced, as a way to express communication. This abstracts away many of the complexities introduced by concurrency and therefore we believe it is not a good model for our use case.

Another related work is [40]. There Service Automata are defined. As we do here, service automata pursue the goal of providing an integrated framework for describing services' behavior and to reason about their composition. Some differences between that model and ours should be noted. First service automata present a single buffer between two services and buffers are modeled as multisets meaning that there is no message ordering. That implies that messages can always be consumed as long as they are present in a buffer. This is in contrast to CFSMs where buffers present a FIFO ordering and only the oldest message in the buffer can be consumed. Second, service automata allow for synchronous interactions (i.e. they have a primitive to synchronously exchange messages). In [40] composability and correctness of service automata is studied and an algorithm to synthesize at most one missing (compatible) participant from a choreography is given.

The present section is organized as follows. In Section 4.9.1 we introduce multichannel CFSMs together with their relation with CFSMs and in Section 4.9.2 we present asynchronous communicating finite state automata.

4.9.1 Multichannel CFSMs

In this section we introduce multichannel communicating finite state machines (mCFSM), an extension to CFSMS that allows multiple channels between each pair of participants.

Definition 84 (mCFSM)**.** *A* multichannel CFSM *on* Msg *(mCFSMs, for short) is a finite transition system* $(\mathsf{Q}, q_0, \mathsf{C}, \mathsf{Msg}, \delta)$ *where*

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

A communicating system *is a map S assigning a mCFSM $S(\mathsf{p})$ 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 $\tau \in S(\mathsf{p})$ when τ is a transition of $S(\mathsf{p})$.*

In the same way as before the semantics of a communicating system is obtained by considering configurations. Here configurations are exactly the same as in pure CFSMs, only that now the channels are not restricted to a single pair between each pair of participants.

Definition 85 (Semantics of mCFSM)**.** *The* configuration *of a multichannel 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 $w_{\mathsf{pq}} \in \mathsf{Msg}^*$. A configuration $s' = (\vec{q'}, \vec{w'})$ is* reachable *from another configuration $s = (\vec{q}, \vec{w})$ by the* firing of the transition τ *(written $s \xrightarrow{\tau} s'$) if there exists $\mathsf{m} \in \mathsf{Msg}$ such that either:*

Snd $\tau = (q_{\mathsf{p}}, \mathsf{pq}_n!\mathsf{m}, q'_{\mathsf{p}}) \in \delta_{\mathsf{p}}$ *and*

(a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ *for all* $\mathsf{p}' \neq \mathsf{p}$*; and*

(b) $w'_{\mathsf{pq}_n} = w_{\mathsf{pq}_n} \cdot \mathsf{m}$ *and* $w'_{\mathsf{p'q'}_m} = w_{\mathsf{p'q'}_m}$ *for all* $\mathsf{p'q'}_m \neq \mathsf{pq}_n$*; or*

Rcv $\tau = (q_{\mathsf{q}}, \mathsf{pq}?\mathsf{m}, q'_{\mathsf{q}}) \in \delta_{\mathsf{q}}$ *and*

(a) $q'_{\mathsf{p}'} = q_{\mathsf{p}'}$ *for all* $\mathsf{p}' \neq \mathsf{q}$*; and*

(b) $\mathsf{m} \cdot w'_{\mathsf{pq}_n} = w_{\mathsf{pq}_n}$ *and* $w'_{\mathsf{p'q'}_m} = w_{\mathsf{p'q'}_m}$ *for all* $\mathsf{p'q'}_m \neq \mathsf{pq}_n$

Fact 3. *Multichannel CFSMs with just one channel for each ordered pair of participants are equivalent to pure CFSMs.*

Multichannel CFSMs can be emulated by pure CFSMs in the sense that a multichannel communicating system is free from communication errors if and only if the emulated system also is. The procedure consist in spawning a new participant for each channel between two other participants. This new participant is a simple forwarder of messages from one participant to another. In this way a multichannel communication between a pair of participants is replaced by multiple one channel communications with a forwarder in the middle. An application of this procedure is depicted in Figure 4.16. The key aspect of this emulation is that it preserves message ordering.

Definition 86 (Emulated system)**.** *Given a multichannel communicating system $(M_{\mathsf{p}})_{\mathsf{p} \in \mathsf{P}}$ we enlarge the set P by adding one participant for each channel in the original system $\mathsf{P}' = \mathsf{P} \cup \bigcup_{\mathsf{p} \in \mathsf{P}} \{\mathsf{p}^{\mathsf{pq}_n} \mid \mathsf{pq}_n \in \mathsf{C}_{\mathsf{p}}\} \cup \bigcup_{\mathsf{p} \in \mathsf{P}} \{\mathsf{p}^{\mathsf{qp}_n} \mid \mathsf{qp}_n \in \mathsf{C}_{\mathsf{p}}\}$. Each new participant $\mathsf{p}^{\mathsf{sr}_n} \in \bigcup_{\mathsf{p} \in \mathsf{P}} \{\mathsf{p}^{\mathsf{pq}_n} \mid \mathsf{pq}_n \in \mathsf{C}_{\mathsf{p}}\} \cup \bigcup_{\mathsf{p} \in \mathsf{P}} \{\mathsf{p}^{\mathsf{qp}_n} \mid \mathsf{qp}_n \in \mathsf{C}_{\mathsf{p}}\}$ is defined by the following mCFSM:*

- $\mathsf{Q}_{\mathsf{p}^{\mathsf{sr}_n}} = \{q_0\} \cup \bigcup_{\mathsf{m} \in \mathsf{Msg}} \{q_{\mathsf{m}}\}$
- $\mathsf{C}_{\mathsf{p}^{\mathsf{sr}_n}} = \{\mathsf{sp}_n^{\mathsf{sr}_n}, \mathsf{p}^{\mathsf{sr}_n}\mathsf{r}_n, \mathsf{p}^{\mathsf{sr}_n}\mathsf{s}_n, \mathsf{rp}_n^{\mathsf{sr}_n}\}$
- $q_{0_{\mathsf{p}^{\mathsf{sr}_n}}} = q_0$
- $\delta_{\mathsf{p}^{\mathsf{sr}_n}} = \bigcup_{\mathsf{m} \in \mathsf{Msg}} \{(q_0, \mathsf{sp}_n^{\mathsf{sr}_n}?\mathsf{m}, q_{\mathsf{m}}), (q_{\mathsf{m}}, \mathsf{p}^{\mathsf{sr}_n}\mathsf{r}_n!\mathsf{m}, q_0)\}$

Each old participant $\mathsf{q} \in \mathsf{P}$ *is replaced by a new participant* q' *where:*

- $\mathsf{Cq}' = \{\mathsf{qp}_n^{\mathsf{qr}_n} \mid \mathsf{qr}_n \in \mathsf{C}_\mathsf{q}\} \cup \{\mathsf{p}^{\mathsf{sq}_n}\mathsf{q}_n \mid \mathsf{sq}_n \in \mathsf{C}_\mathsf{q}\}$
- $\delta_{\mathsf{q}'} = \bigcup_{\mathsf{m} \in \mathsf{Msg}} \{(q, \mathsf{qp}_n^{\mathsf{qr}_n}!\mathsf{m}, q') \mid (q, \mathsf{qr}_n!\mathsf{m}, q') \in \delta_\mathsf{q}\} \cup \{(q, \mathsf{p}^{\mathsf{sq}_n}\mathsf{q}_n?\mathsf{m}, q') \mid (q, \mathsf{sq}_n?\mathsf{m}, q') \in \delta_\mathsf{q}\}$

It should be clear that even though we transform each channel (buffer) into two new channels[6] the message ordering is guaranteed as a result of the way in which δ for the forwarders is defined. Note that the forwarders honor that order since upon consuming one message from the "input" channel (the channel that serves to receive messages from the original sender) the sole possible action of the forwarder is to send that very same message throughout the "output" channel (the channel that serves to send messages to the original receiver). Since every channel is FIFO and the forwarders behave in a FIFO way, the intrachannel order is guaranteed. On the other hand, there is no guarantee with respect to interchannel message ordering in the original model. Therefore we argue that the forwarders do not introduce more concurrency than what was originally in the model and because of that the two systems, although they are not bisimilar and neither have the same traces, they are equivalent with respect to deadlock freeness, absence of unspecified receptions and orphan messages. This observation is important because it provides a way of checking multichannel communicating systems by resorting once again to GMC checking of the emulated system.

Proposition 14. *The emulated system preserves communication errors.*

Proof. First note that the forwarders cannot inccur in communication errors as in the initial state they are able to receive the full range of messages and upon consumption of a message their only possibility is to forward that message. Second note that message ordering is preserved. That means that if in the original system messages m and m' were sent in that order over

[6] As channels always come in pairs, one in each direction, the forwarders always have four channels, two with each of the participants involved. Only two of these channels are actually used as forwarders are unidirectional.

a channel sr in the emulated system they are sent in the same order over the channel $\mathsf{sp}^{\mathsf{sr}}$ and thus they are sent in that very same order through the channel $\mathsf{p}^{\mathsf{sr}}r$. Then one can show that for every error configuration reachable in the original system there is a configuration that presents the same error that is reachable in the emulated system and viceversa.

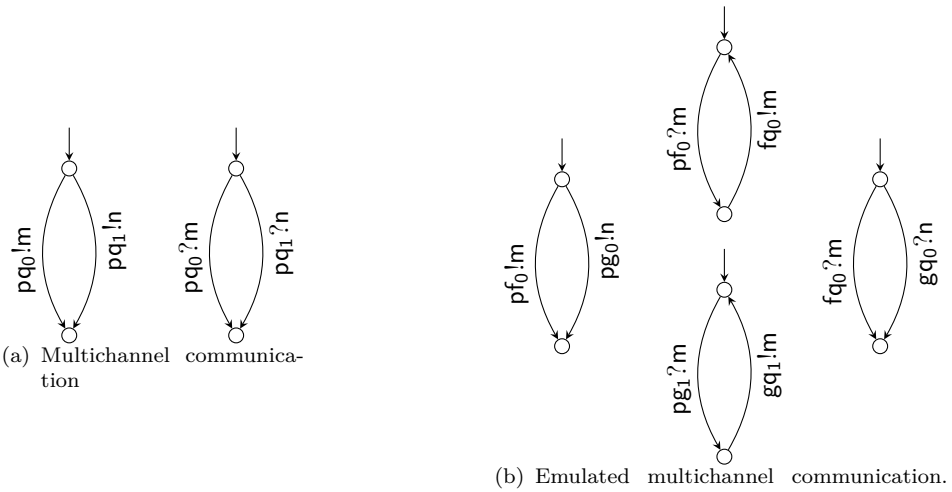
- $\Rightarrow$ Consider any error configuration e reachable in the original system and consider any path that reaches e. Then replace every action $\mathsf{sr!m}$ with the sequence of actions $\mathsf{sp}^{\mathsf{sr}}\mathsf{!m}, \mathsf{sp}^{\mathsf{sr}}\mathsf{?m}, \mathsf{p}^{\mathsf{sr}}\mathsf{r!m}$ and every action $\mathsf{sr?m}$ with $\mathsf{p}^{\mathsf{sr}}\mathsf{r?m}$, this new path is present in the emulated system and reaches a state where buffer configurations and enabled arrows are the same for the set of shared machines (i.e. all the machines in the emulated system minus the forwarders) and the forwarders are in their initial state with empty buffers.

- $\Leftarrow$ Consider any error configuration e reachable in the emulated system and consider any path that reaches e. We already established that the error cannot be forwarders' fault as they can always progress. Therefore no matter what the state of the buffers is in e there is a configuration e' that presents the same communication error and where the state of the buffers of the forwarders is empty and they are in their initial state. Then consider any path that reaches e' it is formed by sequences of the form $\mathsf{sp}^{\mathsf{sr}}\mathsf{!m}, \ldots, \mathsf{sp}^{\mathsf{sr}}\mathsf{?m}, \ldots, \mathsf{p}^{\mathsf{sr}}\mathsf{r!m}$ and $\mathsf{p}^{\mathsf{sr}}\mathsf{r?m}$ (where the $\ldots$ denote the possible interleaving of other actions). Then it suffices to replace every sequence $\mathsf{sp}^{\mathsf{sr}}\mathsf{!m}, \ldots_0, \mathsf{sp}^{\mathsf{sr}}\mathsf{?m}, \ldots_1, \mathsf{p}^{\mathsf{sr}}\mathsf{r!m}$ with the sequence $\ldots_0, \ldots_1, \mathsf{sr!m}$ and every $\mathsf{p}^{\mathsf{sr}}\mathsf{r?m}$ with $\mathsf{sr?m}$ to obtain a path that exists in the original system and reaches a configuration that has the buffers in the same state with the same enabled arrows.

□

4.9.2 Asynchronous communication finite state automata

In this section we present Asynchronous Communication Finite State Automata (ACFSA for short). These automata are defined with a set of channels and internal buffers and have three different kind of transitions: a) internal transitions that model internal process computation, b) buffer transitions that model asynchronous communication between the same automaton. Buffer transitions may also be thought of a model of message passing communication in a multithreaded process. And lastly, c) external



(a) Multichannel communication

(b) Emulated multichannel communication. We omit unused arrows for clarity

Figure 4.16: Example of encoding multichannel communication into multiple one channel communications.

communication transitions model sending and receiving of external messages (i.e. with other ACFSAs).

Definition 87 (Asynchronous communication finite state automaton)**.** *Let* P *be a set of participants and* Msg *a set of messages. An* asynchronous communication finite state automaton *(ACFSA for short) is a structure* $A_{\mathsf{P}} = \langle \mathsf{Q}, \mathsf{B}, \mathsf{C}, \Sigma, \delta, q_0, \mathsf{F} \rangle$ *such that:*

- Q *is a finite set of states,*
- $\mathsf{B} \subseteq \{\mathsf{pq}_n \mid \mathsf{pq} \in \mathsf{P}^2, n \in \mathbb{N}, \mathsf{p} \neq \mathsf{q}\}$ *is a finite set of buffers,*
- $\mathsf{C} \subseteq \{\mathsf{pq}_n \mid \mathsf{pq} \in \mathsf{P}^2, n \in \mathbb{N}, \mathsf{p} \neq \mathsf{q}\}$ *a set of channels such that* $\mathsf{B} \cap \mathsf{C} = \varnothing$*,*
- Σ *a set of labels that we assume disjoint with respect to* Msg *(i.e.* $\Sigma \cap \mathsf{Msg} = \varnothing$*)*
- $\delta = \delta_{Int} \cup \delta_{Ex} \cup \delta_{Buff}$ *where:*
  - $\delta_{Int} \subseteq \mathsf{Q} \times \Sigma \times \mathsf{Q}$
  - $\delta_{Ex} \subseteq \mathsf{Q} \times (\mathsf{C} \times \{!, ?\} \times \mathsf{Msg}) \times \mathsf{Q}$
  - $\delta_{Buff} \subseteq \mathsf{Q} \times (\mathsf{B} \times \{<<, >>\} \times \mathsf{Msg}) \times \mathsf{Q}$

- $q_0 \in \mathsf{Q}$ *is the initial state, and*
- $\mathsf{F} \subseteq \mathsf{Q}$ *is the set of final states.*

The semantics of ACFSAs is defined with respect to a configuration. A configuration consists of a state of the automaton together with the state of each internal buffer.

Definition 88 (Instantaneous configuration)**.** *Let* $A_{\mathsf{P}} = \langle \mathsf{Q}, \mathsf{B}, \mathsf{C}, \Sigma, \delta, q_0, \mathsf{F} \rangle$ *be an ACFSA over* P *and* Msg *then an* instantaneous configuration *of* A_{P} *is a pair* $(q, \vec{\beta})$ *where* $q \in \mathsf{Q}$, $\vec{\beta} = (\beta_{\mathsf{b}})_{\mathsf{b} \in B}$ *and* $\beta_{\mathsf{b}} \in \mathsf{Msg}^*$. *We say that an instantaneous configuration is* initial *if it is* $(q_0, \vec{\epsilon})$ *and it is* final *if it is* $(q, \vec{\epsilon})$ *with* $q \in \mathsf{F}$.

Configurations evolve through a transition relation that takes into account the semantics assigned to internal buffer transitions. A sequence of configurations starting from the initial state of an ACFSA that evolves according to rules in Definition 89 is an execution.

Definition 89 (Execution of an ACFSA)**.** *A sequence*

$$r = (q, \vec{\beta}) \xrightarrow{\ell} (q_1, \vec{\beta}_1) \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_{n-1}} (q_n, \vec{\beta}_n)$$

is an execution of the ACFSA $A_{\mathsf{P}} = \langle \mathsf{Q}, \mathsf{B}, \mathsf{C}, \Sigma, \delta, q_0, \mathsf{F} \rangle$ *if and only if* $(q, \vec{\beta})$ *is an initial configuration of* A_{P}, $(q_i, \vec{\beta}_i)$ *is a configuration of* A_{P} *for all* $1 \leq i \leq n$ *and if* $(q, \vec{\beta}) \xrightarrow{\ell} (q', \vec{\beta}') \in r$ *then the following holds:*

Int $\ell \in \Sigma$ *then* $(q, \ell, q') \in \delta$

BSnd $\ell = \mathsf{b}\mathsf{<<}\mathsf{m}$ *then:*

- $(q, \ell, q') \in \delta$ *and*
- $\beta'_{\mathsf{b}} = \mathsf{m} \cdot \beta_{\mathsf{b}}$ *and* $\beta'_{\mathsf{b}'} = \beta_{\mathsf{b}'}$ *for all* $\mathsf{b}' \neq \mathsf{b}$.

BRcv $\ell = \mathsf{b}\mathsf{>>}\mathsf{m}$ *then:*

- $(q, \ell, q') \in \delta$ *and*
- $\beta'_{\mathsf{b}} \cdot \mathsf{m} = \beta_{\mathsf{b}}$ *and* $\beta'_{\mathsf{b}'} = \beta_{\mathsf{b}'}$ *for all* $\mathsf{b}' \neq \mathsf{b}$.

r is said to be successful if and only if $(q_n, \vec{\beta}_n)$ *is a final configuration.*

Proposition 15. *Semantics of buffer transitions is the same as the semantics of mCFSM.*

Proof. It suffices to compare rules BSnd and BRcv in Definition 89 with rules Snd and Rcv in Definition 85 to see they are equal. □

The main operation that we want to do with ACFSAs is composition. The composition of ACFSAs should be closed (i.e. composition of ACFSAs yields an ACFSA). After composing two ACFSAs shared channels should become internal buffers and external communication actions associated to them must become buffer actions accordingly. ACFSAs model concurrent processes therefore we adopt a parallel composition semantics to obtain the transition relation of the new composed automaton. Formal definition of this procedure is given in Definition 90.

Definition 90 (Composition of ACFSAs)**.** *Let* $A_{\mathsf{P}} = \langle \mathsf{Q}, \mathsf{B}, \mathsf{C}, \Sigma, \delta, q_0, \mathsf{F} \rangle$ *and* $A'_{\mathsf{P}} = \langle \mathsf{Q}', \mathsf{B}', \mathsf{C}', \Sigma', \delta', q'_0, \mathsf{F}' \rangle$ *be two ACFSAs over* Msg *such that* $\Sigma \cap \Sigma' = \varnothing$, $\mathsf{B} \cap \mathsf{B}' = \varnothing$, $\mathsf{B} \cap \mathsf{C}' = \varnothing$, $\mathsf{B}' \cap \mathsf{C} = \varnothing$ *and* $\delta_{Buff} \cap \delta'_{Buff} = \varnothing$ *the composition of* A_{P} *with* A'_{P} *(denoted as* $A_{\mathsf{P}} \parallel A'_{\mathsf{P}}$*)) is the ACFSA* $A''_{\mathsf{P}} = \langle \mathsf{Q}'', \mathsf{B}'', \mathsf{C}'', \Sigma'', \delta'', q''_0, \mathsf{F}'' \rangle$ *where:*

- $\mathsf{Q}'' = \mathsf{Q} \times \mathsf{Q}'$
- $\mathsf{B}'' = \mathsf{B} \cup \mathsf{B}' \cup (\mathsf{C} \cap \mathsf{C}')$. *The set of internal buffers of the composition is formed by the the internal buffers of each of the automata plus a new buffer for each of the shared channels (i.e. the channels used to communicate between the two automata being composed).*
- $\mathsf{C}'' = (\mathsf{C} \cup \mathsf{C}') \setminus (\mathsf{C} \cap \mathsf{C}')$. *The set of external channels of the composition is formed by the channels of each of the automata minus the shared channels.*
- $\Sigma'' = \Sigma \cup \Sigma'$. *The internal actions is the union of the internal actions of each of the automata.*
- $\delta'' = \delta''_{Int} \cup \delta''_{Ex} \cup \delta''_{Buff}$ *where:*
  - *a)* $((p,p'),\ell,(q,p')) \in \delta''_{Int}$ *for all* $(p,\ell,q) \in \delta_{Int}$ *and* $((p,p'),\ell,(p,q')) \in \delta''_{Int}$ *for all* $(p',\ell,q') \in \delta'_{Int}$. *Internal actions are preserved.*
  - *b)* $((p,p'),\mathsf{c}[! \mid ?]\mathsf{m},(q,p')) \in \delta''_{Ex}$ *for all* $(p,\mathsf{c}[! \mid ?]\mathsf{m},q) \in \delta_{Ex}$ *such that* $\mathsf{c} \notin \mathsf{C} \cap \mathsf{C}'$ *and* $((p,p'),\mathsf{c}[! \mid ?]\mathsf{m},(p,q')) \in \delta''_{Ex}$ *for all* $(p',\mathsf{c}[! \mid ?]\mathsf{m},q') \in \delta'_{Ex}$ *such that* $\mathsf{c} \notin \mathsf{C} \cap \mathsf{C}'$. *External communication actions carried out through a non-shared channel are preserved.*

c) δ''_{Buff} *is given by the following rules:*

I. $((p,p'),\mathsf{b}[<<\mid>>]\mathsf{m},(q,p')) \in \delta''_{Buff}$ *for all* $(p,\mathsf{b}[<<\mid>>]\mathsf{m},q) \in \delta_{Buff}$ *and* $((p,p'),\mathsf{b}[<<\mid>>]\mathsf{m},(p,q')) \in \delta''_{Buff}$ *for all* $(p',\mathsf{b}[<<\mid>>]\mathsf{m},q') \in \delta'_{Buff}$. *Internal buffer actions are preserved.*

II. $((p,p'),\mathsf{b}[<<\mid>>]\mathsf{m},(q,p')) \in \delta''_{Buff}$ *for all* $(p,\mathsf{b}[!\mid ?]\mathsf{m},q) \in \delta_{Ex}$ *such that* $\mathsf{b} \in \mathsf{C} \cap \mathsf{C}'$ *and* $((p,p'),\mathsf{b}[<<\mid>>]\mathsf{m},(p,q')) \in \delta''_{Buff}$ *for all* $(p',\mathsf{b}[!\mid ?]\mathsf{m},q') \in \delta'_{Ex}$ *such that* $\mathsf{b} \in \mathsf{C} \cap \mathsf{C}'$. *Each external communication action carried out through a shared channel becomes an internal buffer action.*

- $q''_0 = (q_0, q'_0)$
- $\mathsf{F}'' = \mathsf{F} \times \mathsf{F}'$

Proposition 16. *Composition of ACFSAs preserves determinism.*

Proof. It is a direct result from the fact that the two automata do not share actions and how δ of the composition is defined. □

One of the main drawbacks of using CFSM as a contract specification language in SOC is that the nature of SOC binding collides with the way of establishing whether a set of participants can interact free of communication errors in the following sense. In SOC the discovery and binding of services is carried out on-demand, meaning that even though many services may be required to accomplish a task, they are gathered one by one as they are required. On the other hand, CFSMs require the full set of participants taking part in a protocol in order to be able to establish whether the protocol will be carried out without errors. ACFSAs are introduced to try to reduce the gap between these two different natures. Therefore one of the main tasks that we want to accomplish with ACFSAs is to obtain, by means of a projection procedure, the communicating interface in the form of a mCFSM. Note that projecting the communicating interface from an ACFSA forces us to distinguish observable behavior from non observable behavior. In this case the observable behavior is formed by the communication transitions in δ_{Ex} while non observable behavior is formed by internal and buffer transitions in $\delta_{Int} \cup \delta_{Buff}$. In general one would expect that the projected communicating interface is *observationally equivalent* to the original automaton. Observational equivalence, as defined in [46] means that there exists a *weak bisimulation* between the original automaton and the projected communication interface. A weak bisimulation (Definition 92) is

a relation like a bisimulation but allowing the automata to take arbitrarily many ϵ-transitions.

Definition 91 (ϵ-closure)**.** *Let* $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *be a communicating machine we define the* $\epsilon-$*closure of* δ *as*

$$\delta^{\frown} = \{(q, \ell, q') \mid q \xrightarrow{\epsilon}_{\delta} \ldots \xrightarrow{\epsilon}_{\delta} q'' \xrightarrow{\ell}_{\delta} q''' \xrightarrow{\epsilon}_{\delta} \ldots \xrightarrow{\epsilon}_{\delta} q'\}$$

(i.e. q' *is reachable from* q *by a string with at most one transition different from* ϵ*).*

Definition 92 (Weak bisimulation)**.** *We say that a machine* $(\mathsf{Q}, q_0, \mathsf{Msg}, \delta)$ *weakly simulates a machine* $(\mathsf{Q}', q_0', \mathsf{Msg}, \delta')$ *if and only if there exists a relation* $S \subseteq \mathsf{Q} \times \mathsf{Q}'$ *such that:*

a) $(q_0, q_0') \in S$ *and*

b) for all $(q, q') \in S$ *if* $(q', \ell, q_\ell') \in \delta'$ *then*

i. $(q, \ell, q_\ell) \in \delta^{\frown}$ *and*

ii. $(q_\ell, q_\ell') \in S$*.*

S *is called a* weak simulation*. If both* S *and* S^{-1} *are weak simulations then* S *is a* weak bisimulation*.*

If there is a weak bisimulation between machines M *and* M' *they are regarded as* weakly bisimilar*.*

Definition 91 and Definition 92 were given in terms of communicating machines for consistency with Definition 64 but they can be straightforwardly extended to ACFSAs. Based on the definition of weak bisimilarity we can define the notion of *weak determinacy* [46] or *weak determinism* [29]. A machine, or in general, an automata is weak deterministic if for every state, every two states that are reachable by a transition ℓ in the ϵ-closure are weak bisimilar. Weak determinism is, in a way, a relaxed version of determinism where ϵ-transitions have a special treatment instead of being treated as regular transitions as is done in *strong* determinism.

Note that in general a procedure like the following will not yield a communication interface observationally equivalent to the original automaton:

1st we transform every internal and buffer action into an ϵ-transition. After this we obtain a non-deterministic automaton with ϵ-transition.

2nd we remove ϵ-transitions by resorting to any known procedure like the one described in [30].

In [29] the authors establish that an *I/O transition system* (an LTS where input, output and internal transitions are distinguished) is weakly deterministic if there is a minimal ϵ-free automaton that is weakly bisimilar (or observationally equivalent). Moreover, such automaton is an ϵ-free weakly deterministic automaton and therefore it is strongly deterministic. We can then refine the previous procedure as follows:

1st we transform (hide) every internal and buffer action into an ϵ-transition. After this we obtain a possibly non-deterministic automaton with ϵ-transition.

2nd we apply a transition minimisation procedure such as the one described in [21]. If the obtained automaton is ϵ-free and for each state there is at most one transition for each label then the original automaton was weakly deterministic and the minimised automaton is observationally equivalent to the original one.

Note that in general the composition of ACFSAs as defined in Definition 90 will not yield a weakly deterministic ACFSA (after hidding internal and buffer actions) even when the composed automata are. For a detailed explanation of why weak determinism is not preserved we refer the reader to [46, Section 11.2].

The previous procedure essentially yields an mCFSM. One would like to have a way of checking this yielded mCFSM against others in order to establish if they can interact without errors. We know that if the automaton was weakly deterministic then the projected communication interface is observationally equivalent, yet there is still a potential threat to the precision of the yielded mCFSM. This is due to the fact that the projection does not take into account the precedence of buffer transitions that is forced as a result of the semantics assigned to them in Definition 89.

For example in Figure 4.17 we show a set of three ACFSA depicted as mCFSM as we are only interested in the communication aspects. As a side note these are perfectly valid ACFSAs. It is clear that this set of participants can interact without errors. Moreover these three participants validate GMC conditions. In Figure 4.17(b) we show the result of composing participants A and B. For readability we only show the sub automaton reachable after receiving the message a through channel ca. The sub

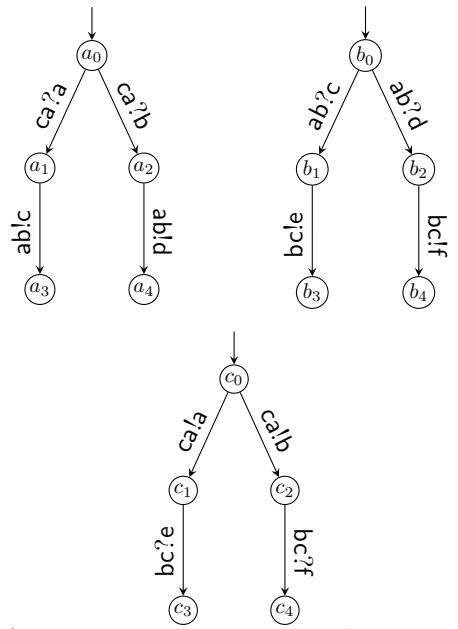
automaton reachable after receiving the message b through channel ca is analogous. One should note that as a result of the composition the communication transitions between the two automata being composed become buffer actions. Every possible interleaving of these actions is generated in the composition. Then some of the paths generated may not be actually executable according to Definition 89. For example it is clear that after receiving message a from an external channel, message d will never be sent and therefore it will never be received. Thus, in our example the paths that include the transition ab>>d are spurious.

Afterwards in Figure 4.18 we show the result of the procedure that projects the communication interface from an ACFSA. First in Figure 4.18(a) we transform every internal and buffer transition into an ϵ-transition obtaining a non-deterministic automaton. Then in Figure 4.18(b) and Figure 4.18(c) we transform that automaton into a deterministic one eliminating ϵ-transitions. The result is the automaton (or mCFSM) shown in Figure 4.18(c).

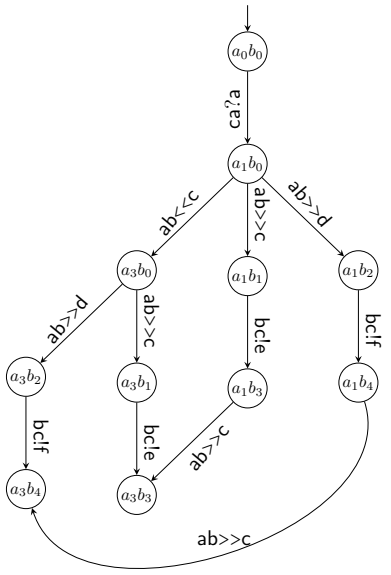
If we now go back to the beginning of this example, we ended up in a situation where we want to check the two machines shown in Figure 4.19. It is clear that these two machines do not validate GMC conditions. More than that, a communicating system formed by these two participants is not free of communication errors. This is in contrast to the original system that was able to execute free of communication errors. Note that the composed automaton of this example is not weakly determinstic, then we already knew that we had no guarantee on the projected communication interface being observationally equivalent. It remains to be studied the relation between weak determinism and the pontential imprecision introduced by the composition as defined in Definition 90.

4.9.3 Concluding remarks

We showed that compatibility of communication interfaces is not preserved by the composition and projection procedure. Note that the machine shown in Figure 4.19(a) is not precise in the sense that it accepts more traces than those that are actually accepted by the ACFSA that originated it. A quick look to the automaton in Figure 4.17(b) reveals that the only actually executable path (of all the paths it has) is the one that after receiving a sends c through the internal buffer, then receives c from the internal buffer and finally it sends e.

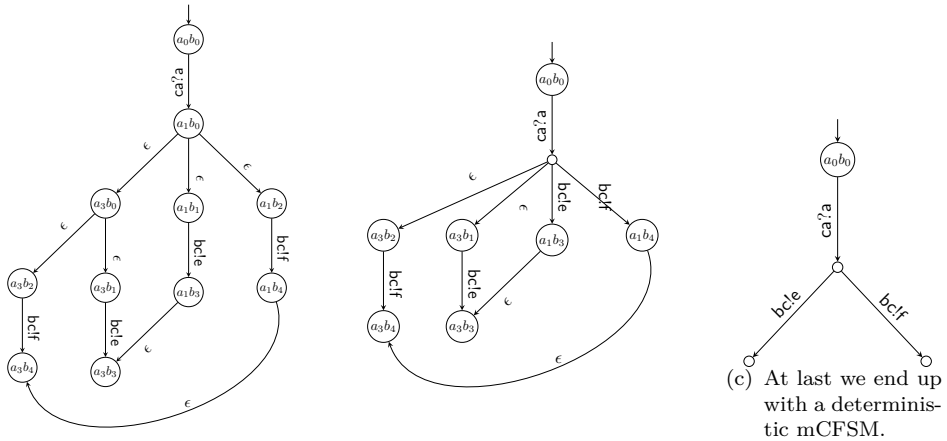


(a) Three asynchronous automata A, B and C depicted as CFSM for simplicity that clearly do not have any communication errors and comply with the GMC conditions



(b) Composition of automata a and b from Figure 4.17(a). Only paths of receiving a are shown for readability. When partially composing two asynchronous automata every communication action between them becomes a buffer action.

Figure 4.17: Example of composition of two participants from a set of three GMC participants.

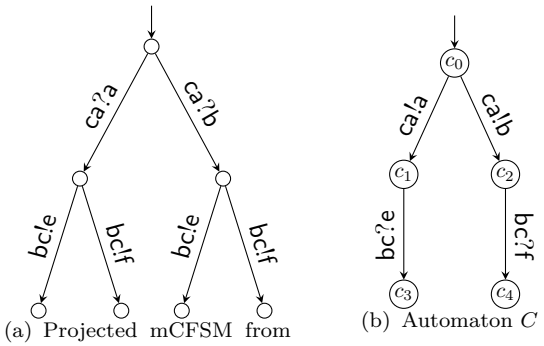


(a) When projecting the communication interface from an asynchronous automaton we transform every internal and buffer transition into ϵ-transitions.

(b) The second step is to transform this non-deterministic, ϵ-transition automaton into a deterministic one getting rid of the ϵ-transitions. We show this intermediate step to ease the following of the example.

(c) At last we end up with a deterministic mCFSM.

Figure 4.18: Projecting mCFSM



(a) Projected mCFSM from the composition of automata A and B

(b) Automaton C

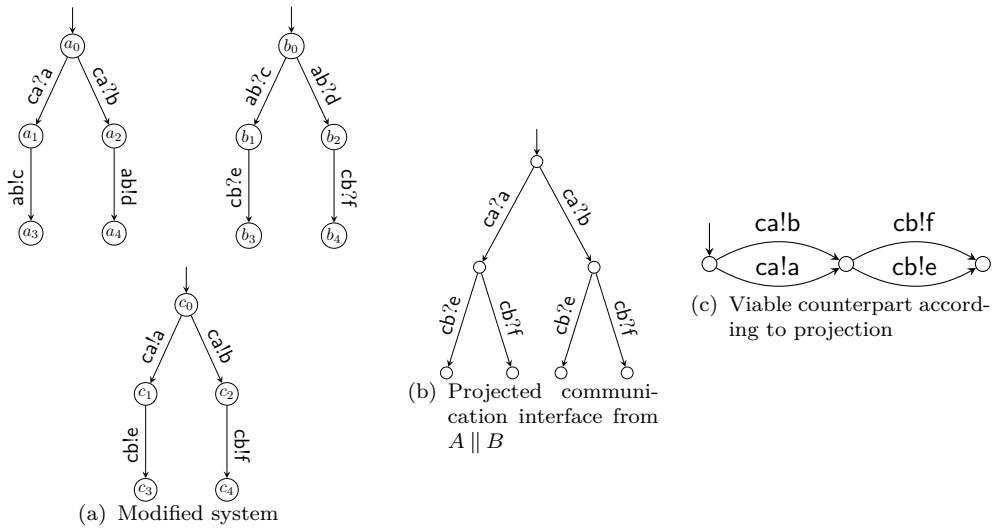
Figure 4.19: Non compatible mCFSMs

One may be tempted to assert that the projected mCFSM serves as a bound of the behavior in the sense that if this projection can communicate without errors with a third participant, then we can compose with this third participant and be sure that the effective behavior will not present communication errors. Regrettably this is not true. To show that it suffices to consider the modified system shown in Figure 4.20(a). The projection of the composition between participants A and B is shown in Figure 4.20(b). In Figure 4.20(a) a machine that is compatible (it validates GMC) with the projection is shown. It is clear that the real automaton may inccur in communication errors if composed with such a counterpart since, for example, after receiving `a` it expects to be receiving `e` but the counterpart may send either `e` or `f`. Again notice that the composed automaton is not weakly deterministic, we stress once again that the relation between weak determinism and the composition of ACFSAs remains to be studied.

Therefore we can conclude that even though the composition is correct in the sense that it respects semantics, the procedure given is not sufficient to obtain a useful communication interface. Such a negative result forces a discussion regarding the precision of the composition in characterizing the exact set of paths that are effectively executable. From our point of view, augmenting precision requires identifying those paths in the composition that are actually executable. The minimum desirable property for the obtained communicating interface is that it is enough to validate error-freeness of the composition. In this sense note that we do not seek for a general result but mainly the preservation of GMC (which is sufficient but not necessary).

4.10 Resumen

Coreografías y orquestación son los principios de diseño fundamentales para el desarrollo de software distribuido [55]. En el caso de orquestación la coordinación es responsabilidad de un orquestador que especifica y, posiblemente, implementa el flujo de trabajo distribuido. Las coreografías proporcionan la noción de una visión global, esto es una especificación holística que describe las interacciones distribuidas. Esta visión global puede ser proyectada hacia las partes que constituyen el software. En el modelo de orquestación, las componentes computacionales distribuidas coordinan entre sí a partir de interactuar con un componente especial, el orquestador, que en tiempo de ejecución decide cómo debe evolucionar el flujo de trabajo.



(a) Modified system

(b) Projected communication interface from $A \parallel B$

(c) Viable counterpart according to projection

Figure 4.20: Modified counter example

En un modelo coreografiado, las componentes distribuidas ejecutan e interactúan autónomamente entre sí sobre la base de que se espera que el flujo de control interno cumpla con su rol especificado en la visión global. La dificultad principal en un modelo coreografiado es chequear si un conjunto de participantes con intención de interactuar entre sí van a tener éxito al ejecutar en paralelo o, en otras palabras, si un conjunto de participantes puede interoperar sin errores de comunicación.

Las *communicating machines* [9] son un formalismo basado en autómatas para describir las vistas parciales de una coreografía. Las mismas describen el rol de un participante en una comunicación de múltiples participantes en términos de mensajes intercambiados. Para el caso de *communicating machines* el chequeo de interoperabilidad fue resuelto en [38] por medio de un algoritmo para sintentizar coreografías a partir de *communicating machines*. Allí se garantiza que si el algoritmo puede sintentizar una coreografía exitosamente, entonces los participantes podrán interactuar sin errores de comunicación.

Representamos coreografías como *global graphs* para representar la visión global y *communicating finite state machines* (CFSMs) [9] para representar procesos individuales. Un *global graph* [19] es básicamente un grafo de flujo de trabajo que representa las relaciones de causalidad así como los puntos de elección distribuida y el *fork/join* de *threads*. Por otro lado una CFSM

es un autómata de estados finitos donde todos los estados son de aceptación y cuyas transiciones son etiquetas en un conjunto de acciones donde una acción puede ser un envío ℓ = pq!a, o sea p escribe el mensaje a en el buffer a q, o una recepción ℓ = pq?a desde el canal pq, o sea q extrae el mensaje a del buffer de p.

Un sistema de *communicating finite state machines* evoluciona a través de interacciones de comunicación sobre canales, representados como pq de p a q ; los mensajes enviados se mantienen en un buffer FIFO no acotado hasta que el receptor lo extrae.

Divisamos dos inconvenientes con el framework basado en orquestación de las ARNs:

1. el mecanismo de *binding* basado en *entailment* de LTL establece una relación asimétrica entre *requires-points* y *provides-points* dado que formaliza la noción de inclusión de trazas; también,

2. incluir explícitamente al orquestador (el autómata que etiqueta un hiperarco de comunicación) aumenta el tamaño del autómata resultante haciendo que el análisis sea más costoso.

Para atacar estas dos debilidades proponemos una variante de ARNs en la que los *provides-points* se etiquetan con *communicating machines* que declaran el comportamiento exportado por un servicio y los hiperarcos de comunicación se etiquetan con *global graphs* que declaran el comportamiento global del canal de comunicación. Este nuevo framework permite dos enfoques diferentes para verificar *binding*. Un enfoque de arriba hacia abajo basado en proyectar el *global graph* que etiqueta una conexión hacia cada uno de los puertos para luego chequear bisimulación entre el comportamiento declarado y el esperado. Otro enfoque de abajo hacia arriba que se basa en el algoritmo para sintetizar coreografías presentado en [38]. Si se puede sintetizar una coreografía a partir de los participantes entonces podemos chequear isomorfismo entre el grafo generado y aquel que define a una conexión.

Extendimos las *communicating machines* para considerar valores asociados a los mensajes. Utilizamos condiciones expresadas en un lenguaje de primer orden para incorporar asunciones y requerimientos sobre los datos intercambiados. En una primera presentación, en la que un mensaje debe estar asociado con la misma condición en cada transición en la que aparezca, damos un procedimiento para chequear si esta clase de máquinas pueden interactuar sin errores de comunicación. Uno de los inconvenientes de utilizar la condición GMC para esto es que la misma es suficiente pero no necesaria.

Por lo tanto la misma deja afuera sistemas que en la práctica interactúan sin errores. Para atacar este problema, enunciamos una propiedad llamada bisimulación *up-to* recepciones que, conjeturamos, bajo ciertas condiciones debería permitirnos relajar la condición GMC. Hasta este punto los datos fueron considerados de manera exclusivamente local. Esto significa que las condiciones sólo podían predicar sobre los valores intercambiados en la transición a la cual referían. Propusimos una forma de establecer algunas relaciones entre valores de distintas transiciones. Para esto impusimos la restricción de prohibir dependencias transitivas entre los datos de modo de evitar la indecidibilidad.

Por último desarrollamos un nuevo tipo de autómatas cuyo objetivo es proporcionar un framework integral para modelar tanto acciones de cómputo internas así como acciones de comunicación. La interfaz de comunicación de estos autómatas se da en términos de *communicating machines*. Para esto extendimos a las *communicating machines* para permitir múltiples canales entre participantes. Estos autómatas poseen una operación de composición y esta composición internaliza la semántica de la comunicación. Mostramos que la composición ingenua en conjunto con la proyección de la interfaz de comunicación no es un mecanismo sufiente para preservar la ausencia de errores de comunicación en un sistema.

Chapter 5

Conclusions and further work

In this chapter we summarize the contributions of this thesis along with conclusions and discussions that are scattered throughout the thesis and point out open lines of research.

5.1 Summary of contributions

In this thesis we aimed to explore and contribute to the build of a formal execution model for services. We worked under the assumption that this is an ineludible prerequisite in order to enable an implementation of the necessary infraestructure to realize SOC to its full extent. To this end we worked on two topics. The first one was providing formal semantics to services and similar systems. To this end we equiped ARNs with an operational semantics that fully captures reconfigurations and provides a unified view for actions and structural changes. This enables reasoning on both aspects in an integrated way. In particular, as demonstration we developed a definition of a satisfaction relation that permits the evaluation of temporal formulae. This semantics allows us to identify and distiguish reconfiguration actions. A salient aspect of this work is that it does not require for an ARN to be grounded. On the contrary semantics for open systems is provided. Therefore we believe it serves better as an execution model for services. Although the semantics is based on an automata model, it is provided in a farely abstract manner by resorting to notions of category theory.

This framework was further extended to cope with non monotonic reconfigurations or remotions of services. We considered remotions as the result of two possible scenarios. First the successful termination of a ser-

vice. We modeled finite or terminating services as finite state automata. We provided a way of enconding these automata into Muller automata so we could incorporate these kind of services into the ARN framework with minimum alterations. Second, we gave a characterization for failures and we fixed a rollback behaviour into our execution model. We discussed that other approaches to failure resolution should be considered but it remains as further work. We also showed that in this model an execution that fails infinitely often may still be considered as successful. We showed how we can use LTL formulae satisfaction to mitigate this problem.

The second aspect of SOC we worked on was automatic discovery and binding. To this end we proposed a way of integrating the framework of ARNs and its orchestration model, with a choreographed model based on communicating finite state machines providing a mechanism for checking binding that solves the asymetric relation resulting from considering trace inclusion as a binding mechanism. This new framework permits two different approaches to binding. A top-down approach based on projecting the global graph attached to connections to each port and checking bisimulation between the declared and the expected behaviour. And a bottom-up approach that relies on the algorithm to synthesize choreographies presented in [38]. If a choreography can be synthesized from the participants then one can check if that choreography is isomorphic to the global graph that defines the connection.

We extended communicating machines to consider values associated to messages. Conditions expressed in a first order language were used to attach assumptions and requirements on exchanged data. In a first presentation, where the same message is forced to be associated with the same condition on every transition, we gave a procedure to check whether this class of machines can interact free of communication errors. One of the drawbacks of using GMC condition is that it is sufficient but not necessary. Therefore it disregards perfectly fine systems. We enunciated a property called bisimulation up-to receptions that we conjecture that under certain conditions should allow us to relax GMC. Up to this point data was considered in a local only perspective meaning that conditions were only allowed to predicate on the values exchanged in the same transition. We also proposed a way of allowing some relations between values exchanged in different transitions. This was done by allowing conditions to predicate over variables in other transitions and imposing restrictions preventing transitive dependencies on data in order to avoid falling into undecideability. In the end we developed a new kind of automata that aims at provinding an integrated framework for

modelling internal computation as well as communication whose communicating interface is given in terms of communicating machines. To this end we further exteneded communicating machines to enable multiple channels between participants. These automata can be composed and the composition internalizes the communication semantics. We showed that naive composition plus projection of communicating interfaces is not enough to preserve communication error-freeness.

5.2 Further work

Here we state some direct possible lines to continue the work presented in this thesis:

We already stated some possible lines to extend our work in Chapter 3. In ARNs we were able to state that a successful execution of an activity yields a successful execution of each of its components. This result was not exteneded to HARNs. Studying the conditions under which this result can be extended should be an immediate step further. Also in order to provide an even more realistic execution environment for ARNs we propose to consider the repository as a dynamic environment which can change along the execution. Another line that can be explored in order to provide a more complete framework is to consider explicit support for compensations and study its impact in the semantics.

A number of topics remain open from Chapter 4. In the first place the proof or disproof of Conjecture 1 and Conjecture 2 is a natural step forward. More generally we propose to study and develop conditions that allow us to relax GMC requirements in order to enable a communicating system not to excercise some paths. We argue that this would constitue an interesting tool to improve CFSMs as mechanism for binding check when using data. Note that his goal can be seen as opposite to one of the benefits that we declared for using CFSMs as a mechanism for binding check. That is the symmetric relation between participants as opposed to the asymmetric relation yielded by trace inclusion. Therefore the implications of allowing prunnings to communicating systems should be studied under the light of this goal.

ACFSAs remain as an open topic also. We propose to pursue conditions under which the preservation of error-freeness is guaranteed after composition. In particular we propose to study the relation between service automata in [40] and ACFSAs as both formalisms share an important

set of ideas and goals. Further reasearch would also involve studying the relation between ACFSAs and the semantics for service execution given in Chapter 3 in order to provide an integrated view on the reconfiguration and communicational aspects of service oriented systems.

5.3 Resumen

En esta tesis exploramos y contribuimos a construir un modelo de ejecución formal para software orientado a servicios. Trabajamos bajo la hipótesis de que ésto es un requisito ineludible para posibilitar la implementación de la infraestructura necesaria para realizar el paradigma de SOC de forma completa. Con este objetivo trabajamos en dos temas. El primero, dotar de semántica formal a los servicios y sistemas de software similares. Para esto equipamos a las ARNs con una semántica operacional que captura completamente las acciones de reconfiguración y provee una visión unificada de las acciones y de los cambios estructurales. Ésto permite razonar sobre ambos aspectos de una manera integrada. En particular, como muestra, desarrollamos una definición de relación de satisfacción que permite la evaluación de fórmulas temporales. Esta semántica nos permite identificar y distinguir las acciones de reconfiguración. Un aspecto saliente de este trabajo es que no se consideran a las ARNs como sistemas cerrados. Por el contrario proveemos una semántica de sistema abierto. Por lo tanto consideramos que esta semántica es más apropiada como modelo de ejecución para servicios. A pesar de que la semántica está basada en autómatas, la misma es provista de manera abstracta utilizando nociones de teoría de categorías.

Luego extendimos este framework para icorporar renconfiguraciones no monótonas (o remoción de servicios). Consideramos remociones como el resultado de dos escenarios posibles. Primero, la terminación exitosa de un servicios. Modelamos servicios finitos (o que terminan) utilizando autómatas de estados finitos. Proveemos una forma de codificar estos autómatas con autómatas de Muller de manera de poder incorporar este tipo de servicios en el framework de ARNs con mínimos cambios. Segundo, damos una caracterización de falla y fijamos un comportamiento de *rollback* en nuestro modelo de ejecución. Discutimos que se deberían considerar otros enfoques a la resolución de fallas. Esto queda como trabajo futuro. También mostramos que en este modelo una ejecución que falla infinitamente podría ser considerada exitosa y mostramos cómo se puede utilizar la satisfacibilidad de fórmulas LTL para mitigar este problema.

El segundo aspecto vinculado a SOC en el que trabajamos fue el de *discovery* y *binding* automático. Con este objetivo propusimos una forma de integrar el framework de las ARNs y su modelo de orquestación, con un modelo de coreografías basado en *communicating finite state machines* de manera de proveer un mecanismo de verificación de *binding* que soluciona la relación asimétrica que resulta al considera inclusión de trazas como mecanismo de *binding*. Este nuevo framework permite dos enfoques diferentes para verificar *binding*. Un enfoque de arriba hacia abajo basado en proyectar el *global graph* que etiqueta una conexión hacia cada uno de los puertos para luego chequear bisimulación entre el comportamiento declarado y el esperado. Otro enfoque de abajo hacia arriba que se basa en el algoritmo para sintetizar coreografías presentado en [38]. Si se puede sintetizar una coreografía a partir de los participantes entonces podemos chequear isomorfismo entre el grafo generado y aquel que define a una conexión.

Extendimos las *communicating machines* para considerar valores asociados a los mensajes. Utilizamos condiciones expresadas en un lenguaje de primer orden para incorporar asunciones y requerimientos sobre los datos intercambiados. En una primera presentación, en la que un mensaje debe estar asociado con la misma condición en cada transición en la que aparezca, damos un procedimiento para chequear si esta clase de máquinas pueden interactuar sin errores de comunicación. Uno de los inconvenientes de utilizar la condición GMC para esto es que la misma es suficiente pero no necesaria. Por lo tanto la misma deja afuera sistemas que en la práctica interactúan sin errores. Para atacar este problema, enunciamos una propiedad llamada bisimulación *up-to* recepciones que, conjeturamos, bajo ciertas condiciones debería permitirnos relajar la condición GMC. Hasta este punto los datos fueron considerados de manera exclusivamente local. Esto significa que las condiciones sólo podían predicar sobre los valores intercambiados en la transición a la cual referían. Propusimos una forma de establecer algunas relaciones entre valores de distintas transiciones. Para esto impusimos la restricción de prohibir dependencias transitivas entre los datos de modo de evitar la indecidibilidad. Por último desarrollamos un nuevo tipo de autómatas cuyo objetivo es proporcionar un framework integral para modelar tanto acciones de cómputo internas así como acciones de comunicación. La interfaz de comunicación de estos autómatas se da en términos de *communicating machines*. Para esto extendimos a las *communicating machines* para permitir múltiples canales entre participantes. Estos autómatas poseen una operación de composición y esta composición internaliza la semántica de la comunicación. Mostramos que la composición ingenua en conjunto con la

proyección de la interfaz de comunicación no es un mecanismo sufiente para preservar la ausencia de errores de comunicación en un sistema.

Bibliography

[1] G. Ausiello, P. G. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In A. Restivo, S. R. D. Rocca, and L. Roversi, editors, *Proceedings of 7th Italian Conference on Theoretical Computer Science*, volume 2202 of *Lecture Notes in Computer Science*, pages 312–327. Springer-Verlag, October 2001.

[2] F. Barbanera, U. D. Liguoro, and R. Hennicker. Global types for open systems. In *Proceedings 11th Interaction and Concurrency Experience, ICE 2018, Madrid, Spain, 20-21 June 2018.*, 2018.

[3] 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.

[4] S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In J. Field and M. Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 191–202. ACM, 2012.

[5] B. Benatallah, F. Casati, and F. Toumani. Representing, analysing and managing Web service protocols. *Data & Knowledge Engineering*, 58(3):327–357, 2006.

[6] C. Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK, 1985.

[7] M. Boreale and M. Bravetti. Advanced mechanisms for service composition, query and discovery. In Wirsing and Hölzl [66], pages 282–301.

[8] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger. A survey of self-management in dynamic software architecture specifications. In *Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems, WOSS 2004*, pages 28–33. ACM, 2004.

[9] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[10] R. Bruni, A. Bucchiarone, S. Gnesi, D. Hirsch, and A. Lluch-Lafuente. Graph-based design and analysis of dynamic software architectures. In *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2008.

[11] R. Bruni, A. Bucchiarone, S. Gnesi, and H. C. Melgratti. Modelling dynamic software architectures using typed graph grammars. *Electr. Notes Theor. Comput. Sci.*, 213(1):39–53, 2008.

[12] R. Bruni, A. Lluch-Lafuente, U. Montanari, and E. Tuosto. Service oriented architectural design. In *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 186–203. Springer, 2007.

[13] M. G. Buscemi and H. C. Melgratti. Contracts for abstract processes in service composition. In A. Legay and B. Caillaud, editors, *Proceedings Foundations for Interface Technologies, FIT 2010, Paris, France, 30th August 2010.*, volume 46 of *EPTCS*, pages 9–27, 2010.

[14] L. Caires and H. T. Vieira. Conversation types. *Theor. Comput. Sci.*, 411(51-52):4399–4440, 2010.

[15] R. Cambini, G. Gallo, and M. G. Scutellà. Flows on hypergraphs. *Mathematical Programming*, 78:195–217, 1997.

[16] G. Cécé and A. Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005.

[17] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[18] G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In T. Walsh, editor, *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*, volume 2239 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2001.

[19] P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In H. Seidl, editor, *Proceedings of 21st European Symposium on Programming, ESOP 2012, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2012*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] F. DeRemer and H. H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Trans. Software Eng.*, 2(2):80–86, 1976.

[21] J. Eloranta. Minimizing the number of transitions with respect to observation equivalence. *BIT*, 31(4):576–590, 1991.

[22] J. L. Fiadeiro. *Categories for software engineering.* Springer-Verlag, 2005.

[23] J. L. Fiadeiro and A. Lopes. Consistency of service composition. In J. de Lara and A. Zisman, editors, *Proceedings of 15th International Conference Fundamental Approaches to Software Engineering (FASE 2012), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2012)*, volume 7212 of *Lecture Notes in Computer Science*, pages 63–77, Tallinn, Estonia, March/April 2012. Springer-Verlag.

[24] J. L. Fiadeiro and A. Lopes. An interface theory for service-oriented design. *Theoretical Computer Science*, (503):1–30, 2013.

[25] J. L. Fiadeiro, A. Lopes, and L. Bocchi. An abstract model of service discovery and binding. *Formal Aspects of Computing*, 23(4):433–463, 2011.

[26] X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.

[27] F. Gadducci. Graph rewriting for the π-calculus. *Mathematical Structures in Computer Science*, 17(3):407–437, 2007.

[28] R. Guanciale and E. Tuosto. An abstract semantics of the global view of choreographies. In M. Bartoletti, L. Henrio, S. Knight, and H. T. Vieira, editors, *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, volume 223 of *EPTCS*, pages 67–82, 2016.

[29] R. Hennicker, S. Janisch, and A. Knapp. On the observable behaviour of composite components. *Electr. Notes Theor. Comput. Sci.*, 260:125–153, 2010.

[30] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

[31] IBM. Websphere enterprise service bus. On-line. `https://www-01.ibm.com/software/integration/wsesb/library/`.

[32] A. Iglesias. Implementación y estudio de un algoritmo para la comprobación de general multiparty compatibility entre communicating finite state machines con datos. Master's thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, August 2016. Advisors: Ignacio Vissani and Carlos G. Lopez Pombo.

[33] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M. A. Gerosa, and A. B. Hamida. Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1):23–45, Jul 2011.

[34] D. Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006.

[35] R. H. JBoss Developer. Jboss savara project. On-line. Available at `http://savara.jboss.org`.

[36] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. World Wide Web Consortium, Candidate Recommendation CR-ws-cdl-10-20051109, November 2005.

[37] Y. Kesten, Z. Manna, and H. M. A. Pnueli. A decision algorithm for full propositional temporal logic. In C. Courcoubetis, editor, *Proceedings of 5th International Conference Computer Aided Verification CAV '93*, volume 697 of *Lecture Notes in Computer Science*, pages 97–109. Springer-Verlag, 1993.

[38] J. Lange, E. Tuosto, and N. Yoshida. From communicating machines to graphical choreographies. In S. K. Rajamani and D. Walker, editors, *Proceedings of 42rd. Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 221–232, New York, NY, USA, 2015. ACM.

[39] A. Lapadula, R. Pugliese, and FrancescoTiezzi. A formal account of ws-bpel. In D. Lea and G. Zavattaro, editors, *Proceedings of 10th International Conference on Coordination Models and Languages*, volume 5052 of *Lecture Notes in Computer Science*, pages 199–215. Springer-Verlag, 2008.

[40] N. Lohmann. *Correctness of services and their composition*. PhD thesis, University of Rostock, 2010.

[41] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In F. B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 137–151. ACM, 1987.

[42] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems*. Springer-Verlag, New York, 1995.

[43] S. McIlvenna, M. Dumas, and M. T. Wynn. Synthesis of orchestrators from service choreographies. In M. Kirchberg and S. Link, editors, *Conceptual Modelling 2009, Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009), Wellington, New Zealand, January 20-23 2009.*, volume 96 of *CRPIT*, pages 129–138. Australian Computer Society, 2009.

[44] S. McLane. *Categories for working mathematician*. Graduate Texts in Mathematics. Springer-Verlag, Berlin, Germany, 1971.

[45] E. Mendelson. *Introduction to Mathematical Logic; (3rd Ed.)*. Wadsworth and Brooks/Cole Advanced Books & Software, Monterey, CA, USA, 1987.

[46] R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[47] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Information and computation*, 100(1):1–40, Sept. 1992.

[48] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Information and computation*, 100(1):41–77, Sept. 1992.

[49] F. Montesi and N. Yoshida. Compositional choreographies. In P. R. D'Argenio and H. Melgratti, editors, *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 425–439. Springer-Verlag, 2013.

[50] H. N. Nguyen, P. Poizat, and F. Zaïdi. Service-oriented computing: 10th international conference, icsoc 2012, shanghai, china, november 12-15, 2012. proceedings. In C. Liu, H. Ludwig, F. Toumani, and Q. Yu, editors, *Proceedings of Service-Oriented Computing: 10th International Conference, ICSOC 2012, Shanghai, China, November 12-15*, pages 525–532, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[51] H. N. Nguyen, P. Poizat, and F. Zaïdi. Automatic skeleton generation for data-aware service choreographies. In A. Nikora and S. Chulani, editors, *Proceedings of 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 320–329, Pasadena, CA, Nov 2013. IEEE Computer Society.

[52] OASIS. Ws-bpel v2.0. On-line. `https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm`.

[53] T. Open Service Oriented Architecture collaboration. Whitepapers and specifications. On-line. Available at `www.osoa.org`, (see also `oasis-opencsa.org/sca`).

[54] Oracle. Oracle service bus. On-line. `http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html`.

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

[56] D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier Science, 2004.

[57] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13(1):45–60, 1981.

[58] M. Simonot and V. Aponte. A declarative formal approach to dynamic reconfiguration. In *Proceedings of the 1st International Workshop on Open Component Ecosystems*, IWOCE '09, pages 1–10, 2009.

[59] T. G. Stavropoulos, D. Vrakas, and I. Vlahavas. A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review*, 40(3):247–270, Oct 2013.

[60] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 133–192. Elsevier, 1990.

[61] E. Tuosto and R. Guanciale. Semantics of global view of choreographies. *J. Log. Algebr. Meth. Program.*, 95:17–40, 2018.

[62] I. Tutu and J. L. Fiadeiro. Service-oriented logic programming. *Logical Methods in Computer Science*, 11(3), 2015.

[63] W3C. Soap version 1.2. On-line. `https://www.w3.org/TR/2007/REC-soap12-part1-20070427/`.

[64] W3C. Web services description language. On-line. `https://www.w3.org/TR/wsdl/`.

[65] W3C. Ws-cdl cr. On-line. `https://www.w3.org/TR/ws-cdl-10/`.

[66] M. Wirsing and M. M. Hölzl, editors. *Rigorous Software Engineering for Service-Oriented Systems - Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing*, volume 6582 of *Lecture Notes in Computer Science*. Springer, 2011.