Capturing functional and non-functional connector

Marco Autili, Chris Chilton, Felicita Di Giandomenico, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Ilaria Matteucci, Hongyang Qu, Antonino Sabetta, Massimo Tivoli

CONNECT

Emergent Connectors for

Eternal Software Intensive Networked Systems

ICT FET IP Project

Deliverable D2.1

Capturing functional and non-functional connector behaviours

SEVENTH FRAMEWORK PROGRAMME

http://www.connect-forever.eu

INRIA · DOCOMO Euro-Labs · LANCASTER UNIVERSITY · THALES · Universita' dell'Aquila · technische universität dortmund · UNIVERSITY OF OXFORD · UPPSALA UNIVERSITET

Project Number	:	231167
Project Title	:	CONNECT – Emergent Connectors for Eternal Software Intensive Networked Systems
Deliverable Type	:	Report

Deliverable Number	:	D2.1
Title of Deliverable	:	Capturing functional and non-functional connector behaviours
Nature of Deliverable	:	R
Dissemination Level	:	Public
Internal Version Number	:	1.0
Contractual Delivery Date	:	M12
Actual Delivery Date	:	12 February 2010
Contributing WPs	:	WP2
Editor(s)	:	Hongyang Qu
Author(s)	:	Marco Autili, Chris Chilton, Felicita Di Giandomenico, Paola Inverardi, Bengt Jonsson, Marta Kwiatkowska, Ilaria Matteucci, Hongyang Qu, Antonino Sabetta, Massimo Tivoli
Reviewer(s)	:	Valérie Issarny, Paolo Masci, Bernhard Steffen

Abstract

The CONNECT *Integrated Project aims to develop a novel networking infrastructure that will support composition of networked systems with on-the-fly connector synthesis. The role of this work package is to investigate the foundations and verification methods for composable connectors.*

In this deliverable, we set the scene for the formulation of the modelling framework by surveying existing connector modelling formalisms. We covered not only classical connector algebra formalisms, but also, where appropriate, their corresponding quantitative extensions. All formalisms have been evaluated against a set of key dimensions of interest agreed upon in the CONNECT *project. Based on these investigations, we concluded that none of the modelling formalisms available at present satisfy our eight dimensions. We will use the outcome of the survey to guide the formulation of a compositional modelling formalism tailored to the specific requirements of the* CONNECT *project.*

Furthermore, we considered the range of non-functional properties that are of interest to CONNECT*, and reviewed existing specification formalisms for capturing them, together with the corresponding model-checking algorithms and tool support. Consequently, we described the scientific advances concerning model-checking algorithms and tools, which are partial contribution towards future deliverables: an approach for online verification (part of D2.2), automated abstraction-refinement for probabilistic real-time systems (part of D2.2 and D2.4), and compositional probabilistic verification within PRISM, to serve as a foundation of future research on quantitative assume-guarantee compositional reasoning (part of D2.2 and D2.4).*

Keyword List

Software connectors, software components, quantitative verification, functional and non-functional requirements

Document History

Version	Type of Change	Author(s)
0.1	Editing of a preliminary survey	Marco Autili Paola Inverardi Massimo Tivoli
0.7	Complete version for internal review	All
0.8	New version based on Paolo and Bernhard's comments	All
0.9	Revision according to Valérie's comments	All
1.0	Final revision and edit	Chris Chilton

Document Review

Date	Version	Reviewer	Comment
29/01/2010	0.7	Paolo Masci (CNR)	Comments on the linkage of Chapter 4 and 5 and typos
29/01/2010	0.7	Bernhard Steffen (TUDO)	Comments on connector algebras
05/02/2010	0.8	Valérie Issarny (INRIA)	Comments on the structure

Table of Contents

List of Figures

List of Tables

1 Introduction

The CONNECT Integrated Project attempts to develop a novel network infrastructure to allow heterogeneous networked systems to freely communicate with each other. This would be achieved by on-the-fly synthesis of emergent connectors. The role of Work Package 2 (WP2) is to investigate the foundations and verification methods for composable connectors, so that support is provided for composition of networked systems, whilst enabling automated learning, reasoning and synthesis. More specifically, the objectives of WP2 are:

To build a comprehensive theory of composable connectors, by devising a modelling framework for connectors so that complex interaction behaviours can be expressed (with respect to both functional and non-functional properties). The modelling framework will provide support for:

1. *automated reasoning and learning about system interaction behaviour,*
2. *automated connector synthesis, matching, refinement, composition, evolution, and*
3. *(possibly partial) re-use.*

This also concerns finding adequate formalisms to express and quantify, for each connector, the desired Quality of Service (QoS) levels for end-to-end properties among the networked systems, and to formulate algorithms for quantitative and qualitative verification, along with their proof-of-concept implementation.

The work package is structured into the following tasks, which are proceeding in parallel:

- **Task 2.1**. Capturing functional and non-functional connector behaviours. This task aims to guide the project by formalising the notions of connector and component, characterising the types of interaction and identifying a verification approach, capable of capturing non-functional properties.
- **Task 2.2**. Compositional connector operators. The main thrust here is to formulate a compositional modelling and reasoning framework for components and connectors.
- **Task 2.3**. Rephrasing interoperability in terms of connector behaviours. The aim is to formulate techniques for interoperability checking, in the presence of dynamic behaviours and non-functional properties.
- **Task 2.4**. Reasoning toolset. The focus here is on a quantitative verification framework for connectors and components, capable of handling dynamic scenarios and non-functional properties, which includes algorithms and prototype implementations.

In this deliverable, we have set the scene for the formulation of the modelling and verification framework for connector systems through the following:

- In Chapter 2, we survey existing connector modelling formalisms, covering not only classical connector algebra formalisms, for example, WRIGHT, Reo, the Kell calculus, BIP and Bigraphical Reactive Systems, but also, where appropriate, their corresponding quantitative extensions.

  All formalisms are evaluated against eight dimensions of interest for the CONNECT project: compositionality, incrementality, scalability, compositional reasoning, reusability, evolution, non-functional properties and tool support. We also apply the existing formalisms and associated software tools to model a CONNECT scenario in order to demonstrate the capability of these formalisms in the context of CONNECT.
- In Chapter 3, we give an overview of existing classical and new material concerning probabilistic/quantitative verification. Note that automated verification is only one possible method that can be applied in the process of dependability evaluation studied as part of V&V in WP5, and hence there is a partial overlap of material between WP2 and WP5. The work on WP2 is specifically addressing compositional reasoning, which is a key requirement for a reasoning framework suitable for a compositional connector modelling formalism, and focuses on developing algorithmic techniques and implementations.

– In Section 3.1, we review probabilistic labelled transition-system based models for connector systems: continuous and discrete time Markov chains (CTMCs and DTMCs), Markov decision processes (MDPs) and probabilistic timed automata (PTAs). Each model is accompanied with a temporal logic to deal with functional and non-functional property specifications, specifically being CSL for CTMCs, LTL/PCTL for MDPs and PTCTL for PTAs. We also summarise the main verification algorithms and implementation techniques, and briefly discuss extending models with rewards.
– In Section 3.2, we consider the range of non-functional properties that are of interest to CONNECT, such as QoS, and review existing specification formalisms for capturing them, together with the underlying transition-system models, model-checking algorithms and tool support. It is worthwhile noticing that in this first deliverable of WP2, we consider a subset of the QoS properties defined in WP5 [3] that can be expressed in the temporal logics mentioned above.
– We also report in Chapter 3 our scientific contributions towards an automated verification framework for CONNECT, including *online* verification[1] (Section 3.4), automated abstraction-refinement (Section 3.1.3) and probabilistic *compositional reasoning* (Section 3.3), which are partial contributions to future deliverables (D2.2 and D2.4).

[1] This work is partially developed within CONNECT.

2 Survey of existing connector modelling formalisms

2.1 Overview

Broadly speaking, a networked system can be seen as a (possibly complex) software component. Hereafter, we will interchangeably use the terms "networked system" and "component". Externalising the interactions among components and, hence, among networked systems, is one of the primary concerns in CONNECT. Since the late 90's, the notion of software connectors for externalising interactions among computational units (i.e., architectural software components) is playing a crucial role within the notion of software architectures [65]. As pointed out in [50, 65], just to give a practical sense, the notion of connector as means for interaction concretises as shared variables, table entries, buffers, instructions to a linker, (remote) procedure calls, networking protocols, pipes, SQL links between a database and an application, and so forth. In large, and especially distributed systems, connectors become key determinants of system properties, such as performance, resource utilisation, global rates of flow, scalability, reliability, security, evolvability, and so forth.

In order to begin investigating the issues and challenges related to how to model and reason about component (and hence, in the context of CONNECT, about networked system) and connector behaviours, in this survey we discuss and evaluate five existing approaches about component and connector modelling and analysis. During the last decade, the significant attention given to the concept of connecting software components has led to the proliferation of multiple definitions for the notion of connector and different categorisations. In this survey, we basically consider two categories in the same way as in [27]: *in the data flow-setting, connectors define the way data is transferred between components; in a control-flow setting, connectors instead define synchronisation constraints, while abstracting the data flow*.

The first approach that we consider is described in the seminal work conducted by Allen and Garlan [7]. In [7], the authors define a control-flow event-based paradigm for both computation and coordination. The WRIGHT architecture description language [6] is used as a specialised notation for architectural specification. As an underlying formalism, the authors embed in WRIGHT an approach based on process algebra. In fact, in [7], CSP [59] (*Communicating Sequential Processes*) is used by the authors in order to provide an operational formalisation of the separation between computation and coordination.

The second approach essentially concerns the work of Arbab described in [10], among other references. In [10], the author emphasises the separation between computation and coordination by defining a data-flow paradigm. Arbab defines the notion of *Abstract Behaviour Type*s (ABTs) as a higher-level alternative to ADT (*Abstract Data Type*) and proposes it as a proper foundation model for both components and their composition. An ABT defines an abstract behaviour as a relation among a set of *timed-data-streams*, without specifying any detail about the operations that may be used to implement such behaviour or the data types it may manipulate for its realisation. ABTs allow for loose coupling and exogenous coordination, which are considered, in [10], as the two essential properties for components and their composition.

The third approach concerns a family of process calculi called the "*Kell calculus*" [24, 60, 25, 67] developed by Stefani, Bidinger, and Schmitt. It has been intended as a basis for studying distributed (and ubiquitous) component-based systems. Essentially, the Kell calculus is a *high-order* extension of the π-calculus. Its aim is to support the modelling of different forms of process *mobility* (e.g., logical and physical mobility). This is done by considering the possibility to directly transmit *processes* as messages and not only channels (used by processes in order to communicate) as it is in the π-calculus.

The fourth approach, by Bliudze and Sifakis, concerns an early work described in [27]. The authors propose an algebraic formalisation of the structure of the interactions enabled by connectors in a component-based system implemented in the Behaviour-Interaction-Priority (BIP) framework [66, 21]. It is a control-flow paradigm based on active/inactive communication ports of components.

The fifth approach is based upon a framework called Bigraphical Reactive Systems developed by Milner [53], which consists of a bigraph together with a collection of bigraphical rewrite rules. We will examine how we can encode connectors in this system, which deals elegantly with locality and mobility issues. Unlike the other formalisms, this creature is purely notational.

The above mentioned formalisms are all *non-quantitative*, in that they are unable to express quantitative characteristics such as the probability of an event occurring, the elapsing of time, performance, QoS, etc. Since non-functional properties are a key requirement for CONNECT, we also include in the survey

existing *quantitative extensions* of connector algebras. More specifically, we consider three quantitative extensions of Reo [10] because other connector algebras do not have quantitative extensions so far. The first of these is, in part, described in [13]. The authors extend the work in [10] in order to take into account QoS attributes of both computation and coordination, e.g., *shortest time for data transmission*, *allocated memory cost for data transmission*, and *reliability* represented by the probability of successful transmission. The work described in [13] cannot be considered as a mere extension of the work described in [10] since it defines a semantic model for connectors different from ABTs, i.e., it is an operational model based on a QoS extension of *constraint automata* [20] called *Quantitative Constraint Automata*. In spirit, this model is a variant of a *labelled transition-system* model. The remaining extensions of Reo are also based on the constraint-automata semantics, and allow two forms of probability distributions, continuous-time (with no nondeterminism) and discrete-time (with nondeterminism). Note that WRIGHT, BIP and the Kell calculus do not have quantitative extensions.

Beyond the above mentioned approaches, there are many other approaches in the literature that should be considered for a comprehensive survey. Thus, this survey should be viewed as preliminary. Despite this, we believe that the investigations we have performed are sufficient in order to facilitate understanding of the following: (i) what are the key aspects of a formalism or notation for connector modelling and analysis, (ii) what we can reuse/combine from the state of the art, and (iii) what we should add as new features.

It is worth noting that all of the above mentioned approaches strive towards a common goal. That is, to create a foundation for the establishment of a well consolidated component-based development approach where software connectors, as means for interaction, are considered first-class entities.

However, while there are a number of similarities, mainly in the nature of the addressed problem, these approaches also exhibit many differences *on how* and *at what level* the problem is solved. We considered the heterogeneous set of existing approaches and selected those five mentioned above, in order to identify as many aspects as possible that are relevant when modelling/analysing components and connectors. The rationale for the selection of the formalisms is as follows. The work in [7] (WRIGHT) is widely considered as seminal work, and hence it is worth discussing it in this survey. Furthermore, the notation/formalism introduced there conforms to a control-flow event-based paradigm. The work in [10] (Reo connectors) deeply differs from the previous one since, among other differences, it conforms to a data-flow paradigm. Thus, it is informative to consider it as well. Its extension in [13] introduces another key difference, since it takes into account QoS attributes that are not considered by the other approaches. The works in [24, 60, 25, 67] (Kell calculus) and [53] (Biographical Reactive Systems) introduce mobility, locality and dynamism, characteristics that are not present in other formalisms, in contrasting ways. Finally, the work described in [27] (BIP connectors), among other differences, adopts a control-flow paradigm to model the structure of interactions between components by means of an algebra of connectors.

For all of these different approaches, it turns out that it is not easy to identify which specific connector models have to be used to fulfill the specific requirements on the components' interaction under which the system being assembled must operate. Moreover, the lack of a comprehensive all-in-one vision and point-to-point comparison of the existing approaches to connector definition and formalisation makes matters worse. This is why we decided, within CONNECT, to embark on the production of this survey.

In this survey, the considered approaches are briefly summarised and evaluated with respect to eight dimensions of interest for connectors in CONNECT. In light of the above discussion, we decided to keep the description of the following dimensions as general as possible, in order to make them assessable for all of the "different in nature" approaches. Then, we apply and assess each dimension depending on the purposes of the specific approach.

- **Compositionality**: this dimension concerns the ability to define a connector in a hierarchical way out of simpler connectors, and it does not matter how we conduct this hierarchical construction, with the result always equivalent. This means that the connector construction process should be done with respect to a composition operator '$*$' that is associative: for all $x, y, z : x * (y * z) \equiv (x * y) * z$. Compositionality is crucial for the purposes of CONNECT and, in particular, for dynamic connector synthesis since it ensures efficiency of the synthesis process, reuse, and allows it to support connector evolution.

- **Incrementality**: incrementality is implied by compositionality but the former does not imply the latter. This dimension concerns hierarchical construction of connectors. However, it differs from

the previous dimension, since it does not have to be guaranteed that this hierarchical construction enjoys the associativity property. Incrementality is another crucial aspect as it promotes connector reuse.

Note that, even though different approaches may be compositional, they still might express different *abilities*. For instance, as it will be clear later, in a control-flow setting both WRIGHT [7] and BIP [27] refer to a different notion of associative composition operator. For the former, it is the CSP parallel composition [59] operator defined by a trace-based semantics. For the latter, it is an algebraic operator that unifies synchronisation constraints imposed by connectors defined by algebraic expressions. However, the latter only partially supports compositionality since the operator is associative only under the assumption that the connectors have the same type.

- **Scalability**: scalability is also implied by compositionality, and it refers to the ability for connector models to scale to systems with an increasing number of components (e.g., systems of systems). In conjunction with the compositionality degree of the connector composition operator, another aspect that influences scalability is the granuality of atomic connectors.

- **Compositional reasoning**: this dimension is related to compositionality but not necessarily. It refers to the ability to infer properties held by the *whole* system by locally checking properties held by its *parts*. This dimension promotes the efficiency of the analysis that can be carried out by performing local checks instead of a global one, hence avoiding state-space explosion in some cases.

  Analogously to what has been discussed above for compositionality, compositional reasoning might have different meanings in different settings. For instance, WRIGHT [7] and BIP [27] allow for a different compositional reasoning. The former allows for predicting global behavioural properties of the connected system by looking at properties that hold locally for the primary constituents of the system. Among other differences, the latter allows for incrementally checking if two or more connectors represent the same set of interactions (i.e., if they impose the same synchronisation constraints) by looking at the interactions allowed by its constituent parts.

- **Reusability**: this dimension concerns the degree of reuse of connectors. A connector can be: (i) reusable in any context (i.e., it is always reusable), (ii) parameterised with respect to an abstract characterisation of a set of contexts and, hence, reusable only in some contexts (i.e., it is partially reusable), or (iii) not reusable at all (i.e., it is not reusable) since it is tailored to a specific context.

- **Evolution**: this dimension refers to the ability to express into the connector model evolution in terms of modular *dynamic* behaviour and *reconfiguration*. For instance, the connector model should allow for specifying how to deal with the departure or the arrival of a component/connector (leaving or joining the system, respectively) without interrupting the execution of the system. Furthermore, this dimension also refers to the ability to express component/connector migration, hence supporting the analysis of connector/component physical and logical mobility. For instance, considering physical mobility, it should be possible to specify how a component/connector execution can be migrated from a hardware platform (e.g., a mobile phone) to another one, without interrupting the operativity of the system. That is, for both dynamism and mobility it might be useful to be able to specify how the component/connector current state can be stored and restored upon reconfiguration and/or migration.

- **Non-functional properties**: this dimension refers to the possibility to express and reason about *non-functional* characteristics of the component/connector interaction. It promotes the analysis of properties such as *performance*, *reliability*, *security*, and *timeliness*.

- **Tool support**: this dimension concerns the existence of a specialised notation supported by automated tools for architectural analysis. For instance, WRIGHT enables the use of CSP for behaviour conformance checking. The added value of this dimension is more methodological/practical than theoretical/foundational.

Each of the considered approaches are applied to the modelling of a CONNECT scenario, the Mary Scenario introduced in the DoW [1], which involves basic digital systems like phones, to highlight how much universal interoperability is already a central issue.

The survey is organised as follows. Section 2.2 describes the simple application scenario that we consider, in order to present the specific characteristics of each approach in the field. Sections 2.3 and 2.4 give a brief summary for each considered formalism, pointing out its motivations, goals, and main aspects. Advantages and disadvantages are analysed for each approach with respect to the eight dimensions of interest to CONNECT, and a comparison between the current approach and all of the ones considered previously is given.

2.2 The Mary Scenario

In this section we present a scenario of CONNECTed systems, borrowed from the DoW [1], which is used in Sections 2.3 and 2.4 in order to illustrate the application of the approaches analysed and evaluated by this survey.

This scenario, hereafter called the *Mary Scenario*, although simple, stresses how much universal interoperability is already a central issue, by sketching an everyday situation involving basic digital systems like phones.

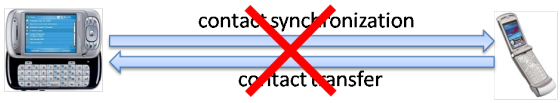

Figure 2.1: The Mary Scenario (a): pre-CONNECTed world

Mary uses a smart phone as her primary mobile phone (right-hand side of Figure 2.1). Among other data, it contains also a complete and up-to-date directory of all her contacts. The pressure of evolving technologies and a change of her daytime requirements lead Mary to buy a new smart phone (left-hand side of Figure 2.1). At this point Mary would like to get rid of her old smart phone and begin to use her brand-new gadget as her sole mobile phone. She needs, though, to synchronise the two phones in order to have all her precious contact data on the new one. In the current pre-CONNECTed world, Mary attempts to transfer her contacts by using the SIM card on her phone, but realises she would experience a possible data-loss since the SIM storage format does not allow keeping all the information associated with a contact (e.g., email address). Moreover, her previous smart phone can not export any data to the SIM card (Figure 2.1). Thus, Mary decides to exploit BlueTooth communication facilities, which are available on both of her phones, in order to exchange complete contact data between the two. The two smart phones are indeed able to connect and Mary may send contact information from one to another, but she is able to transfer only one contact at a time through a rather complicated procedure.

Figure 2.2: The Mary Scenario (b): pre-CONNECTed world

Since it would have taken her far too long to transfer all her contacts, Mary comes up with another solution: using an intermediate networked component that bridges the communication and performs the overall transfer for her. Specifically, Mary switches on her laptop that integrates a BlueTooth subsystem and by using embedded synchronisation software, she is able to connect to her old smart phone and transfer all her contacts onto the laptop, thanks to the fact that the phone has the support to fully synchronise with the application running on the laptop (see Figure 2.2). Then she attempts to synchronise the data on the laptop into her new smart phone. Unfortunately, the synchronisation program lacks the support for her brand-new gadget. So she has to look for a plugin for extending the synchronisation software so that it can communicate with her new smart phone. After configuring the plugin, she is finally able to store all her contacts and all their associated data (see Figure 2.3).

In the CONNECTed world we envision, Mary would have simply avoided all the inconvenience of going through the try-again procedure that finally led her to accomplishing her goal. She would have simply

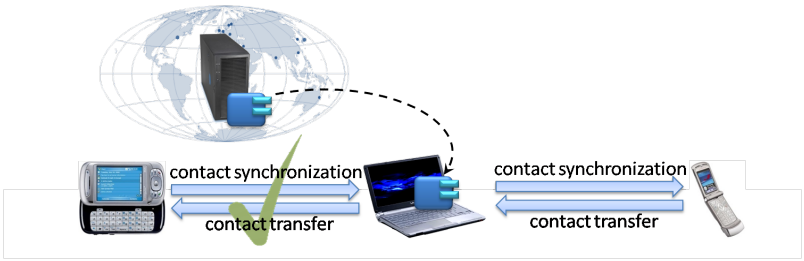

Figure 2.3: The Mary Scenario (c): pre-CONNECTed world

asked her new smart phone to synchronise with the old one and the networked CONNECT enablers would have unobtrusively figured out a way for executing the task. In Figure 2.4, we graphically show the situation Mary wishes to experience. The arrows represent synchronisation and transfer flows of contact data.

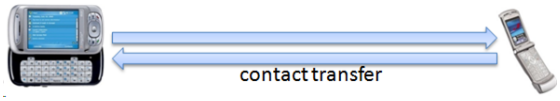

Figure 2.4: The Mary Scenario: foreground

Indeed, behind the scene, the CONNECT enablers would generate an advanced CONNECTor that integrates an intermediate system to bridge the communication. In Figure 2.5, we graphically show the CONNECT solution behind the scene. A software CONNECTor is synthesised in order to achieve interoperability between `CM1` (the contact management software deployed on the old device) and `CM2` (the contact management software deployed on the new device).

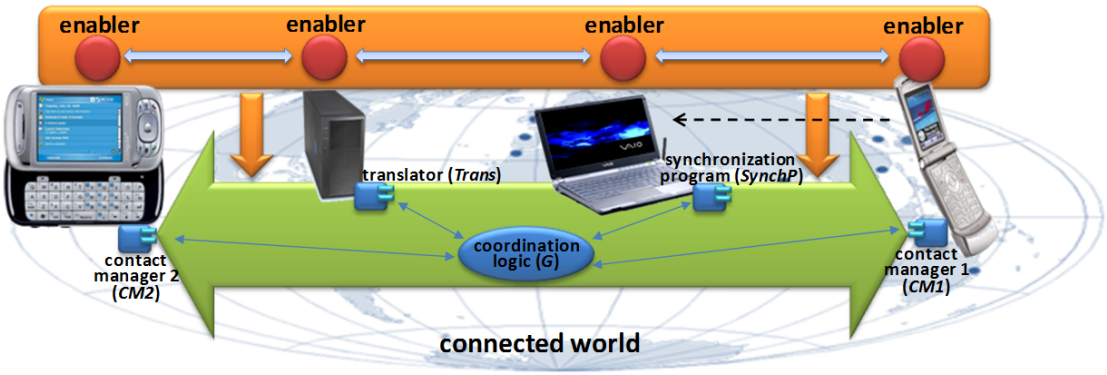

Figure 2.5: The Mary Scenario: the CONNECT solution behind the scene

In order to achieve interoperability, the CONNECT enablers [2] exploit the CONNECTor algebra in order to solve the following central issues:

- *Learning the interaction behaviour of networked systems*: in the Mary Scenario, this is identifying that both phones support the sending/receipt of a contact, that the old one is also able to fully synchronise its contact list with another system (i.e., the laptop), and that in the network there exists another system (i.e., the contact translation software) able to translate the format used to store contacts in the old device into the format used to store contacts in the new device.

- *Automatic synthesis of new CONNECTors that implement new interaction behaviours*: in the Mary Scenario this is equivalent to the use of both the laptop and the contact translation system within the emergent connector for exchanging the contacts between the two smart phones.

- *Validation of the synthesised behaviour with respect to different dimensions*: In the Mary Scenario this could correspond to answering some questions like: “Have all the contacts been correctly transferred?”

- *Assessing the trustworthiness of* CONNECT*ed networks*: Mary was confident of the way she found for transferring her contacts because she trusted all the networked systems (her smart phones and her laptop) and the resulting synthesised behaviour (using the laptop as an intermediate system). In a CONNECTed world, she would have to trust the CONNECT enablers and also part of the surrounding network (e.g., the contact translation system).

2.3 Non-quantitative connector algebras

This chapter provides a concise yet complete description of all the non-quantitative approaches we have surveyed. The *Mary Scenario*, described in Section 2.2, is modelled by using the formalisms offered by different approaches, hence allowing for a point-to-point comparison of them. For each approach we also specify *WHAT* it provides, *WHY* it was conceived and *HOW* it provides what it was conceived for.

2.3.1 Role-glue based architectural connection (WRIGHT)

In the late 90’s, the authors of the work in [7] identified the need of having a formal basis for *box-and-line based* architectural models, only informally specified at that time. Following the basic idea of message communication over the network, the approach uses protocols to describe interaction and, hence, to abstract complex patterns of communication such as pipes, event broadcast, client-server protocols, etc.

The component-connector model in [7] was conceived by having in mind that *expressiveness* and *support for analysis* are winning characteristics for a connector model. To reach a good level of expressiveness, a formal theory should consider a “library” of basic interactions as common cases of basic architectural interaction (e.g., procedure call, pipes, event broadcast, and shared variables). Composition operators have to be defined for combining basic interactions to form more complex ones and it has to be possible to describe fine-grained connector “variants”. To support the analysis, the formalism should permit the analysis of architectural descriptions in an automated way, together with the ability to understand the context-free behaviour of connectors. In other words, the behaviour of the connector has to be clear in a way that is independent of the specific components to which it will connect. Architectural mismatches and well-formedness has to be detectable in the composition of components and connectors, in order to check whether a connector definition is compatible with its use or not. Thus, similarly to abstract behaviour types in programming languages, connector behavioural types should allow for behavioural type checking in architectural analysis. The flexibility for reuse should be further supported by connector subtyping. By considering the above premises, we give the *WHAT-WHY-HOW* summarisation of the work described in [7] as follows:

- **WHAT -** To provide a formal notation plus an underlying theory to model architectural connectors as *explicit semantic entities*.

- **WHY -** To externalise the interaction among components. This is motivated by the fact that, while module interconnection languages and interface definition languages are good for describing *implementation* relationships among dependent parts of the system, they are not suited for explicitly describing *interaction* relationships among loosely coupled components. In fact, differently from implementation-level models, where the focus is on how a component achieves its computation, architectural models specify how that computation is combined with others in the system. In other words, architectural models define the overall structure of the system by also defining abstractions associated to the external interaction among components. These considerations lead to the following *VS*s between the main characteristics of implementation-level models and architectural models:

  - implementation description *VS* architectural description;

  - computation (implementation) *VS* coordination (interaction);

  - “dependent” modules *VS* loosely coupled components;

- definition/use dependency relations among implementation modules *VS* abstract interactions resulting from composition of independent components.

A further motivation of this work concerns the problem of enabling automated reasoning about architectural mismatches.

- **HOW -** To abstractly define computational components together with a collection of connectors. Components are modelled as a collection of *port* protocols and a *specification* of functional properties. Connectors are modelled as a collection of *role* protocols each of them characterising each of the participants in the interaction, and how these roles interact by means of a *glue* protocol.

In [7], the underlining theory used to model and give the semantics of interaction protocols is based on an event-based formalism that is a subset of CSP [59] (*Communicating Sequential Processes*). In order to allow practitioners (e.g., software architects) to easily define an architectural description of the system and automatically reason on its semantics in terms of the semantics of the specified interaction protocols, a formal notation is built on top of this theory. This notation is the WRIGHT architectural description language [6]. In WRIGHT, the architecture of a system is described in three parts: *component and connector types*, *component and connector instances*, and *bindings* between component and connector instances.

A *component type* is described in two parts: *ports* and *specification*. A port models a logical point of interaction with the environment and a specification models an abstract characterisation of the component's functional properties. *Connector type* is also described in two parts: *roles* and *glue*. A role models the expected local behaviour of an interacting party, i.e., obligations of a component participating in the interaction, and glue models a description of how the activities of the roles are coordinated.

Ports, roles and glue are described as interacting protocols in a subset of CSP by considering primitive events or events with I/O data. The formal semantics of the ports, roles and glue composition is also specified in terms of CSP processes by using the parallel composition operator.

Component and connector *instances* specify the actual entities that will appear in the configuration, and *bindings* combine component and connector instances by prescribing which component ports are attached as (or instantiate) which connector roles.

In Section 2.3.1.1 we report an overview of how component and connector types can be modelled and their semantics by means of the subset of CSP chosen in [7]. In this section we discuss also the kinds of analyses and checkings that are made possible by the chosen connector notation and formalism. In Section 2.3.1.2 we apply the summarised CSP-based theory to model the explanatory scenario used in this survey, i.e., the *Mary Scenario* described in Section 2.2. In Section 2.3.1.3 we outline advantages and disadvantages of the reported approach.

2.3.1.1 Overview

As already said, the roles of a connector specify the possible behaviours of each participant in an interaction and the glue describes how to constrain the roles' interaction to behave as specified by the glue itself. In [7], a subset of CSP is used to model these behaviours. This subset has been established with the aim of defining only finite-state (possibly recursive) CSP processes. It includes the constructs listed below[1].

- *Processes and events:* a process describes an entity that can engage in either no events (`STOP` process) or some communication events. Events may be primitive, or they can have associated data (e.g., `e?x` and `e!x` representing input and output of `x` data on an event `e`, respectively).
- *Prefixing:* `e`$\rightarrow$`P` denotes the process that can engage in `e`, hence becoming `P`.
- *Alternative ("external choice"):* `P` $+E$ `Q` denotes a process that can behave like `P` or `Q`, where the choice is made from outside (i.e., made by the other processes that interact with the process).
- *Decision ("internal choice"):* `P` $+I$ `Q` denotes a process that can behave like `P` or `Q`, where the choice is made non-deterministically by the process itself.

[1] For the sake of simplicity, for some of them, we changed notation although their semantics is kept as given in [7].

- *Parallel composition:* in order to model the *combination* of ports, roles and glue processes, CSP processes can be combined using the parallel composition operator $*$. This operator has an *interleaving semantics*. That is, if e is an event in the alphabet of a process P, then e synchronises with the same event in the alphabet of a process Q producing an event e in the alphabet of the parallel composition. Synchronisation of events is thus determined by the alphabets of the CSP processes. An event e in the alphabet of a process P for which no corresponding event exists in the alphabet of any other process Q, is engaged only by P, hence, producing the same event in the parallel composition. Essentially, these "non-shared" events are engaged by exactly one process at a time.

- *Labelling:* labelling can be applied to both events and processes. l.e denotes the event e labelled with l. l:P denotes the process P with each of its events labelled with l. For the purposes of the work described in [7], a success event, suc, exists and the labelling operator does not apply to it. In other words, l:P labels all events of P except for suc.

For the sake of simplicity, with SUC, we denote the process that engages the suc event and then terminates, i.e., SUC denotes suc→STOP.

Connector type: to describe a connector type, process descriptions for each of its roles and glue have to be provided by suitably combining the WRIGHT notation with the CSP formalism. For instance, the following listing is a client-server connector description:

```
connector C_S_connector =
    role Client = (request!x -> result?y -> Client) +I SUC
    role Server = (invoke?x -> ret!y -> Server) +E SUC
    glue = (Client.request?x -> Server.invoke!x -> Server.ret?y -> Client.result!y -> glue) +E SUC
```

The Server role describes the interaction behaviour of the server. It is defined as a process that repeatedly accepts an invocation and then returns; or it can terminate with success instead of being invoked. Because the external choice operator is used, the choice of invoke or suc is determined by the environment. In other words, the behaviour described by Server models a true server as a passive process whose interaction is initiated by the environment.

Client describes the interaction behaviour of the user of the server. Analogously to Server, it is a process that can call the server and then receive the result repeatedly; or terminate. Since we use the internal choice operator, the choice of whether to invoke the server or terminate is determined by the Client process. In other words, Client is the initiator of the interaction.

Comparing the two roles, note that the different choice operators allow one to distinguish between situations in which a role is *obliged* to provide some service (as for Server) and the situation in which a role may *choose* to take advantage of some services, but is not required to do so (as for Client). This is an important distinction for characterising architectural connection, since to understand an interaction it is critical to know what aspects of the behaviour are *required* for a participant and which are simply *available*.

The **glue** process coordinates the behaviour of the two roles by indicating how the events of the roles work together. Thus, connector semantics considers roles as independent processes constrained only by the glue that coordinates their events. More precisely[2], the semantics of a connector description is the CSP parallel composition of the CSP processes specified for the glue and the roles. By means of the labelling operator, the alphabets of the roles and glue are arranged so that the desired coordination occurs. That is, the alphabet of the glue is the union of all possible events labelled by the respective role names, together with suc. This allows the glue to interact with each role. Contrariwise, (except for suc) the role alphabets are disjoint (by virtue of the labelling), so each role can directly interact only with the glue. Because the labelling does not apply to suc, all of the roles and glue can agree on suc for it to occur. This ensures the joint successful termination of all the roles and glue.

Component type: to describe a component type, process descriptions for each of its ports and specification have to be provided by suitably combining the WRIGHT notation with the CSP formalism. For instance, the following listing is a fragment of data-user component description:

```
component DataUser =
    port DataRead = (get -> DataRead) +E SUC
    port ...
    ...
```

[2] A rigorous formalisation of the CSP-based connector semantics is described in [7].

```
port ...
spec ...
```

As already said, bindings attach component ports to connector roles by replacing roles with ports. Since the port protocols define the *actual* behaviour of the components when those ports are associated with the roles, the port protocol takes the place of the role protocol in the resulting system. That is, the roles act as a specification for the ports: provided that the ports are *compatible* to the roles (as explained later), the ports will stand in for the roles in the running system. Thus, the semantics of an attached connector is the CSP process that results from the replacement of the role processes with the associated port processes (supposed that they are compatible).

Port-role compatibility and deadlock-freedom: an important goal of architectural description is to answer the question of when two components can safely communicate using a particular form of interaction. In terms of the notation and underlining formalism discussed so far, this question can be rephrased as follows: "can a given port be used in a given role?". The answer to this question concerns port-role compatibility.

The port-role compatibility check can be seen as a *behavioural type check*. That is, a port is compatible with a role when the set of traces exhibited by the role process is "included" in the set of traces exhibited by the port process. Thus a port should be a behavioural sub-type of a role in the sense that the port should exhibit the same interaction behaviour as the role plus, possibly, other interactions. This makes sense since the glue that is defined only w.r.t. the roles makes the instantiated connector, where ports replace roles, "ignoring" the interactions of the ports that are not interactions of some role. Summing up the compatibility check can be seen as a *trace containment* check [51] between the protocol of the port and the protocol of the role. That is the port is a *refinement* of a role (that, in turn, is an abstraction of the port). Thus roles act as specifications for the ports and compatibility checks verify the behaviour of a port over the traces described by the role. The main motivation for this is to promote correct composition and as maximum as possible reuse of connector and component types.

Like type correctness for programming languages, compatibility for architectural description is intended to provide certain guarantees that the system is well formed. Compatibility, in fact, guarantees that important properties hold in a "compatible" system hence supporting the development of practical tools for compatibility checking. In particular, in [7], an important result concerning compatibility and the properties that it allows a compatible system to preserve is discussed in detail. This result can be phrased as follows: *compatibility ensures the deadlock-freedom of any instantiated deadlock-free and conservative connector*; where a connector is *conservative* if the glue traces are a sub-set of the possible interleavings of role traces[3], and *deadlock-free* if whenever it is in a situation where it cannot make progress, then the last event to have taken place must have been the success event (`suc`). The importance of the above mentioned result is that local compatibility checking is sufficient to maintain deadlock-freedom for any instantiation, i.e., from local checks, it is possible to infer a global check. Note that this is a particular case of compositional verification, particular since it is related to one specific property, i.e., deadlock-freedom.

2.3.1.2 Scenario modelling

In this section we model the components and the connector of the *Mary Scenario* by means of the WRIGHT notation that, in turn, embeds the CSP formalisation for the component and connector type protocols. The following listing represents a fragment of the *Mary Scenario* architectural description:

```
System MaryScenario
    component SynchronizationProgram =
        port SynchP = (synch1 -> contact?c -> SynchP) +E (synch2 -> contact!c -> SynchP)
        spec ...
    component Translator =
        port Trans = translate?c -> res!a -> Trans
        spec ...
    component ContactManager1 =
        port CM1 = (synch1 -> contact!c -> CM1) +E (synch2 -> contact?c -> CM1)
        spec ...
    component ContactManager2=
        port CM2 = (synchOut -> out!a -> CM2) +I (synchIn -> in?a -> CM2)
        spec ...
    connector MaryConnector =
```

[3] That is, the glue prevents behaviours that are not covered by the role specifications. Otherwise a contradiction might occur, i.e., compatible ports could lead the system to a failure, e.g., a deadlock.

```
        role SynchP = (synch1 -> SynchP) +E (synch2 -> contact!c -> SynchP)
        role Trans = translate?c -> res!a -> Trans
        role CM1 = synch1 -> CM1
        role CM2 = synchIn -> in?a -> CM2
        glue = CM2.synchIn -> CM1.synch1 -> SynchP.synch1 -> SynchP.synch2 -> SynchP.contact?c ->
               Trans.translate!c -> Trans.res?a -> CM2.in!a

    Instances
        sp: SynchronizationProgram
        t: Translator
        cm1: ContactManager1
        cm2: ContactManager2
        k: MaryConnector

    Attachments
        sp.SynchP as k.SynchP
        t.Trans as k.Trans
        cm1.CM1 as k.CM1
        cm2.CM2 as k.CM2
end MaryScenario
```

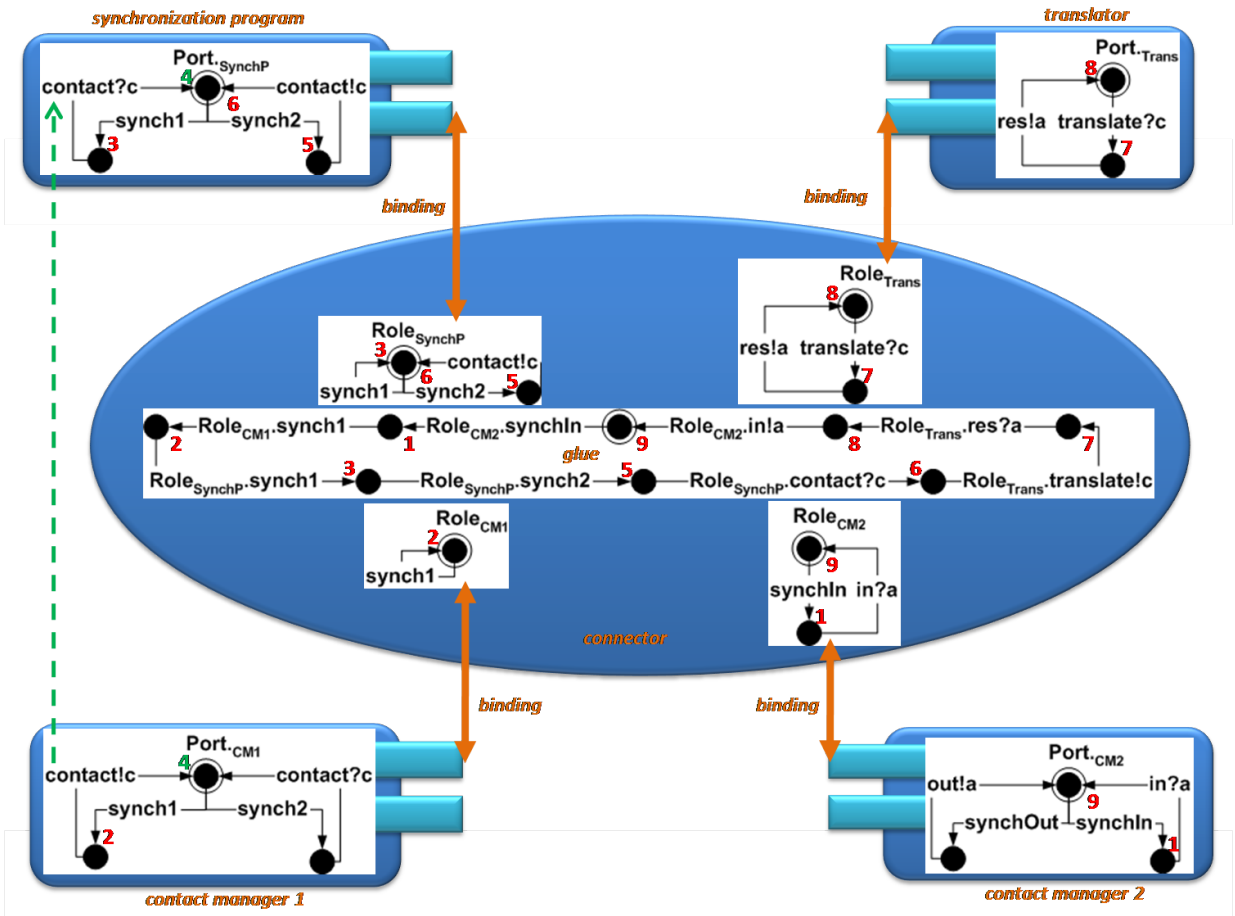

Figure 2.6: Architectural connections of the Mary Scenario for a role- and glue-based connector

`ContactManager1` is the component type abstracting the contact management software deployed on the old device. It can engage in two *contact synchronisation mode* events, i.e., `synch1` and `synch2`. The former is used to inform the contact manager that the environment is going to download contacts, the latter is used to inform the manager that the environment is going to upload contacts. The `contact` event is engaged in order to send or receive contacts. In the *Mary Scenario*, the manager for the old device engages only in `synch1`, and hence also in `contact!c`. Since the contact manager software deployed on the old device has been built ad-hoc for the synchronisation program, `ContactManager1` and `SynchronizationProgram` can directly synchronise on the `contact` event. Thus, the role `CM1` engages only in `synch1`. Note that `ContactManager1` is compatible with `CM1`.

ContactManager2 is the component type abstracting the contact management software deployed on the new device. It can engage in two *contact synchronisation mode* events, i.e., synchOut and synchIn. The former is used to inform the environment that the contact manager is going to upload contacts, the latter is used to inform the environment that the manager is going to download contacts. The event out (resp., in) is engaged to upload (resp., download) contacts. Note that ContactManager2, unlike ContactManager1, is the initiator of the interaction with its environment (whereas, for ContactManager1, the initiator is assumed to be the environment). This is why the architectural descriptions of ContactManager1 and ContactManager2 differ (including the different uses of choice operators). The out (resp., in) event is engaged in order to send (resp., receive) contacts. In the *Mary Scenario*, the manager for the new device engages only in synchIn and, hence, in in?a. Thus, the role CM2 engages only in synchIn and in?a. Note that ContactManager2 is compatible with CM2.

The Translator component type models the contact translator software used to translate contacts from the format used by SynchronizationProgram (deployed on Mary's laptop), to the format used by ContactManager2. Translator engages in translate and res in order to acquire a contact and return its translation. Note that Translator and Trans are identical.

The SynchronizationProgram component type models the software deployed on Mary's laptop used to synchronise contacts with the old device (CM1). SynchronizationProgram and the role SynchP are defined analogously to the other component types and roles, thus no further explanation is required.

The glue for the connector type MaryConnector coordinates the interaction of the components attached to the roles in order to achieve the aim of the *Mary Scenario*. That is, glue coordinates the I/O interaction initiated by the component attached to CM2 in order to force the component attached to CM1 to exchange contacts with the component attached to SynchP. The I/O interaction of the component attached to SynchP is, in turn, coordinated to send the received contacts to the component attached to Trans that translates and sends them to the component attached to CM2, hence achieving Mary's goal.

In Figure 2.6, by using labelled transition systems, we graphically represent the composition of the components and the connector that realise the *Mary Scenario*'s architectural model. The numbers on the transitions show how the interaction progresses.

2.3.1.3 Advantages and disadvantages

In this section we summarise the advantages and disadvantages of using role-glue connectors modelled in WRIGHT with CSP as an underlying formalism. The main benefits can be summarised as follows:

- **Compositionality**: role-glue connectors expressed as CSP processes can be compositional since the CSP parallel composition operator is an associative (and also commutative) operator. Further, compositionality implies incrementality. Since the work described in [7] does not address issues related to the encapsulation of subsystems as components (or connectors) in other systems (i.e., incrementality), role-glue connectors modelled by means of the WRIGHT architectural description language are not compositional although the protocol composition operator of the underlining formalism (i.e., the CSP parallel composition operator) enjoys compositionality.
- **Compositional reasoning**: CSP's parallel composition operator works particularly well in this regard. In particular, it has the desirable compositional property that the traces of a (parallel) composition must satisfy the specifications of each of its parts. This means that one can reason about the behaviour of a system's parts separately, confident that the resulting system will continue to respect the properties established about the parts.
- **Reusability**: role-glue connectors in WRIGHT are *not always reusable* in any context. In fact, a role-glue connector can be reused under port-role compatibility since the roles, by defining the obligations of the components participating in the interaction, give an abstract characterisation of the possible contexts in which the connector can be reused. Thus role-glue connectors are partially reusable w.r.t. the concept of behavioural subtyping polymorphism. With respect to complete reusability, this is not always a complete limitation. In fact, correctness of the reuse is straightforwardly achieved through behavioural subtyping.
- WRIGHT provides automated **tool support** for architectural analysis with respect to qualitative temporal properties, such as deadlock freedom and behaviour conformance check.

Despite the above promising properties of role-glue connectors there are also a few points against to take into account:

- **No incrementality**: in [7], the authors have not addressed issues of hierarchical description. Clearly, any scalable architectural specification language will need to allow encapsulation of subsystems as components (or connectors) in other systems. The key issue to resolve is what should be the "correctness" relationship between a subsystem and the architectural element that it represents. Note that within the context of WRIGHT there is an obvious answer to the question: the subsystem must be a refinement of the element it represents, once events internal to the subsystem have been hidden. That is, a subsystem should be substitutable for the component or connector that it represents. For finite WRIGHT specifications this property is checkable using automated tools.
- **No scalability**: role-glue connectors are centralised (monolithic) connectors and, hence, they do not always scale. This is a direct consequence of the fact that incrementality has not been addressed as discussed in the previous point.
- **No evolution**: for this seminal work, modular dynamicity and reconfiguration cannot be expressed. This is due to the fact that CSP is inherently limited to systems with a static process structure. That is, the set of possible processes must be known at system definition time: new processes cannot be created or passed as parameters in a running system. WRIGHT inherits this limitation. As a direct consequence of the fact that modular dynamicity and reconfiguration cannot be modelled, no form of component/connector migration can be expressed and, hence, neither mobility can be modelled.
- **No support for non-functional property specification**: WRIGHT does not handle properties such as timing behaviour of interactions (or fairness because CSP's semantic model is not rich enough). To address such properties, one can imagine retaining the general descriptive framework (of ports, roles, glue, and glue specification) for connectors, but replacing CSP with an alternative formalism.

2.3.2 Reo connectors as abstract behaviour types

In [10], the starting point of the discussion concerns the well-known notion of *Abstract Data Types* (ADTs). The authors argue that, although ADTs had a tremendous success as a base for object orientation, they subvert some desirable properties of component-based systems. For instance, ADTs prevent the definition of *loosely coupled* components and of their *exogenous coordination*. In [10], loose coupling and exogenous coordination are considered as the two essential properties for components and their composition.

Loose coupling refers to the ability for the components to be semantically independent of one another and internally impose no restrictions on the other components they compose with. As it is explained in [10], ADTs and their underlying method invocation semantics imply a rather tight semantic coupling between the caller and callee pairs of objects.

Exogenous coordination [8] means coordination from outside and refers to the ability, in a model or language, to coordinate the behaviour of black-box entities, without their knowledge, from outside of those entities. This is an essential property for a component composition model to have because it allows building systems with very different emergent behaviour out of the exact same components, simply by composing them differently. In [10], a vivid example of the significance of exogenous coordination is given, with two instances of the classical dining philosophers problem. Different connectors can exogenously impose different coordination protocols on the same components (e.g., philosophers and chopsticks) to yield different composed systems that exhibit different emergent system behaviour. In the case of the dining philosophers, for instance, the possibility of deadlock as an emergent behaviour can be eliminated simply by composing the same components differently. As discussed in [10], ADTs do not allow the definition of glue code that is void of any application-domain specific functionality, hence preventing exogenous coordination in which the role of the glue code is merely to connect components, facilitating their communication and coordinating their interactions, without performing any application-specific computation.

The above discussed motivations have lead the authors to define, in [10], the notion of *Abstract Behaviour Type* (ABT) as a higher-level alternative to ADT, and propose it as a proper foundation model for both components and their composition. An ABT defines an abstract behaviour as a relation among a set

of *timed-data-streams*, without specifying any detail about the operations that may be used to implement such behaviour or the data types it may manipulate for its realisation. As shown in [10], the ABT model supports a much looser coupling than is possible with ADT and is inherently amenable to exogenous coordination. By considering the above premises, the following is the *WHAT-WHY-HOW* summarisation of the work described in [10]:

- **WHAT -** To define a foundation model for both components and their composition enabling two key properties: (i) *loose coupling* and (ii) *exogenous coordination*.
- **WHY -** Since ADTs prevent loosely coupled components and their exogenous coordination, a higher-level alternative to them is needed, i.e., ABTs.
- **HOW -** Formalising the notion of ABT for abstracting behaviour as a relation among *timed-data-streams*, without specifying any detail about the operations implementing such behaviour or the data manipulated to realise it.

In [10], a component-based system consists of component instances and their connectors (i.e., the "glue code"), both of which are uniformly modelled as ABTs. Indeed, the only distinction between a component and a connector is just that a component is an atomic ABT whose internal structure is unknown, whereas a connector is known to be an ABT that is itself composed out of other ABTs. In other words, although components and connectors are indistinguishable when used as primitives to compose more complex constructs, they are still different in that components are black-box primitives whose internal structures are invisible, whereas the internal structure of a connector shows that it, in turn, is constructed out of other (connector and/or component) primitives according to the same rules of composition.

As a concrete instance of the application of the ABT model, the authors describe *Reo*: an exogenous coordination model wherein complex coordinators, called "connectors" are compositionally built out of simpler ones [9].

In Section 2.3.2.1 we report an overview of how the behaviour semantics of loosely coupled components and exogenous connectors, i.e., Reo connectors, can be defined through the ABT model. In Section 2.3.2.2 we apply the reported notions in order to model the explanatory scenario used in this survey, i.e., the *Mary Scenario* described in Section 2.2. In Section 2.3.2.3 we outline advantages and disadvantages of the reported approach by also relating it to the work described in Section 2.3.1.

2.3.2.1 Overview

In [10], the notion of components [11, 14] uses channels as the basic inter-component communication mechanism. A channel is a point-to-point medium of communication with its own unique identity and two distinct ends. A channel supports transfer of *passive data* only; no transfer of control (e.g., procedure calls, exchange of pointers to internal objects/entities) can take place through a channel. As highlighted in [10], using channels as the only means of inter-component communication allows a clean, flexible, and expressive model for construction of the glue code (i.e., the connector) for component composition which also supports exogenous coordination.

A channel that supports only the transfer of passive data implies a *bland* notion of component. That is, the component model considered in [10] allows a component instance to exchange only *untargeted* and *passive* messages with its environment (contrariwise to the kind of target and active messages that characterise communication in object-oriented systems).

Untargeted messages allow a component to not be aware of who the receiver of the message is or how it should be identified. The receiver, on the other hand, is not required to know anything about the sender: it is prepared to receive messages "from its environment" and not from any specific sender.

Passive messages contain only data and carry no control information (e.g., imply no method invocation).

Summing up, untargeted and passive messages are essential to enable loose coupling and exogenous coordination. Furthermore, in contrast to the sophisticated mechanisms required to exchange targeted and active messages (e.g., remote procedure call), the mechanism necessary for exchanging untargeted and passive messages supports only the mundane I/O primitives: receiving (resp., sending) a datum is just a read (resp., write) operation performed by the component instance on a "contact point" that is

recognised by the component environment for the purpose of information exchange. In [10], these contact points are called *ports* of a component instance. Each port is unidirectional. Reading and writing from and to ports are blocking operations, i.e., a component instance synchronises with its environment by classical *rendezvous*.

Because this input/output interaction takes place through the ports of the component instance, sequences of data items that pass through a port emerge as the key building blocks for describing behaviour. In order to relate sequences of data items that pass through different ports (i.e., to relate otherwise independent events), in [10], a notion of relative temporal order is defined. The authors use positive numbers to represent *relative* moments in time. Actual numeric values are not relevant, only their relative ordering is significant and sufficient to support: *ordering* of events (e.g., the occurrence of a certain event precedes or succeeds that of another), *atomicity* of a set of events (e.g., a given set of events occur only atomically), and *temporal progression* (e.g., only a finite set of events can occur within any bounded temporal interval).

All these considerations lead to define the notion of ABT as a relation on the observable input/output of an entity, without saying anything about how it can be realised.

More precisely, to model the sequences of observable input/output data that pass through a port over time, the notion of *timed data stream* is defined in [10]. It is a pair $\langle \alpha, a \rangle$ of a data stream α and a time stream a. α is defined as a sequence of *uninterpreted* data items $\alpha(0)$,$\alpha(1)$,$\alpha(2)$,.... a is defined as a sequence of positive real numbers $a(0)$,$a(1)$,$a(2)$,... such that a is both *strictly increasing* (i.e., for each i, $a(i)$ precedes $a(i+1)$ over the time) and *progressive* (i.e., for every $N > 0$ there exists an index $n \geq 0$ such that $a(n) > N$). For instance, the timed data stream $\langle \alpha, a \rangle$ with $\alpha(0) = 1$,$\alpha(1) = 2$,$\alpha(2) = 3$,... and $a(0) = 1$,$a(1) = 3, a(2) = 5$,... can model a counter starting from 1 and increasing by 1 its value each 2 units of (relative) time. Two timed data streams are equal if their respective (data and time) elements are equal. In general, two streams (data, time, or timed data streams) are related by a relation Op if their respective elements are related by Op as well. Let a be a stream (data, time, or timed data stream), then a' denotes the "tail" of a after $a(0)$ (i.e., the sub-stream $a(1), a(2), \ldots$). In general, $a^{(k)}$ (with $k > 0$) denotes the tail of a after $a(k-1)$. Let s and a be an item and a stream respectively, then $s.a'$ denotes the stream obtained by replacing the first item of a with s.

$$\text{DataCompressor}(<\alpha_{in}, a_{in}>;<\beta_{out}, b_{out}>) \equiv$$
$$\forall i : (\beta_{out}(i) \neq \alpha_{in}(i)) \wedge (a_t(i) < b_r(i) < a_t(i+1))$$

Figure 2.7: ABT model of a data compression component

An ABT is a (maximal) relation R, i.e., R is a set of constraints, over timed data streams denoted as $R(I_1,I_2,\ldots,I_m;O_1,O_2,\ldots,O_n)$ where $I_1,I_2,\ldots,I_m$ (resp., $O_1,O_2,\ldots,O_n$) are the input (resp., output) data streams of R. For instance, in Figure 2.7, we show the ABT model of a *data compression* component. Note that, accordingly to the ABT philosophy, we abstract by the specific data compression algorithm that this component performs by just specifying that the produced data are different from the received data. For the sake of clarity, on the top side of the figure we show a graphical representation of this component, while on the bottom side we report its ABT model by using the syntax proposed in [10]. The component `DataCompressor` has one input port (`in`) and one output port (`out`). Thus it is defined as an ABT over two timed data streams: $\langle \alpha_{in}, a_{in} \rangle$ and $\langle \beta_{out}, b_{out} \rangle$ modelling the sequences of data that pass over the time through the input and output ports, respectively. The relation defined by the ABT model of `DataCompressor` is a set of constraints on the data and time elements of the involved input/output timed data streams. For instance, the i^{th} output data item is different from the i^{th} input data item, i.e., $\beta_{out}(i) \neq \alpha_{in}(i)$. This is due to the effect of the underlining compression algorithm. The compression calculus does not happen atomically in the sense that it takes time, the instant of time at which the i^{th} input datum is received precedes the moment at which the i^{th} output datum is produced, i.e., $a_{in}(i) < b_{out}(i)$. However, `DataCompressor` behaves correctly since the $(i+1)^{th}$ input data item is received when the i^{th} output data

item has been already produced, i.e., $b_{out}(i) < a_{in}(i+1)$, hence maintaining the order of the compressed data items consistent with respect to the order of the uncompressed data items.

ABTs can be composed to yield other ABTs through a composition similar to the relational join operator in relational databases. That is, two ABTs can be composed over a common timed data stream if one is the producer and the other is the consumer of that timed data stream. Analogously, two ABTs can be composed over zero or more common timed data streams, each ABT playing the role of the producer or the consumer of one of the timed data streams, independent of its role regarding the others. This is accomplished by "fusing" the producer and consumer portals together such that the produced timed data stream is identical to the consumed one. A formal definition of ABT composition can be found in [10]. For instance, an ABT $A3x2(I_1,I_2,I_3;O_1,O_2)$ can be obtained by the composition of two ABTs: $A3x1(I_1,I_2,I_3;\langle\alpha,a\rangle^1)$ and $A1x2(\langle\beta,b\rangle^1;O_1,O_2)$ by fusing $\langle\alpha,a\rangle$ and $\langle\beta,b\rangle$ together (i.e., $\langle\alpha,a\rangle = \langle\beta,b\rangle$). A possible graphical representation of $A3x2$ is shown in Figure 2.8.(e). In a composition of ABTs, different timed data streams with the same upper index are fused together, e.g., $B(\langle\alpha,a\rangle;\langle\nu,n\rangle)$ can be obtained by the following composition (denoted with $\circ$): $B_1(\langle\alpha,a\rangle,\langle\beta,b\rangle^1;\langle\gamma,c\rangle^2) \circ B_2(\langle\delta,d\rangle^2;\langle\mu,m\rangle^1,\langle\nu,n\rangle)$ (where $\langle\beta,b\rangle = \langle\mu,m\rangle \wedge \langle\gamma,c\rangle = \langle\delta,d\rangle$).

Connectors, still modelled as (a composition of) ABTs, can be either basic communication channels (i.e., a point-to-point medium of communication with its own unique identity and two distinct ends) that transfer passive-data only, or a composition of other connectors (and, possibly, components). In *Reo* [9], which is the underlying exogenous coordination model assumed in [10], a connector is graphically represented as a set of *channel ends* and their *connecting channels*, all organised in a graph of nodes and edges as shown in Figure 2.8.

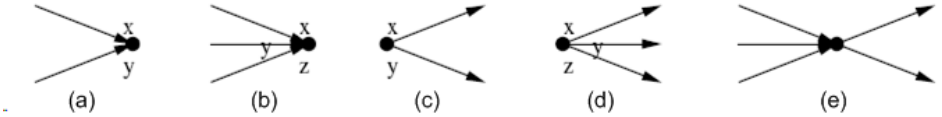

Figure 2.8: Graphical representation of Reo connectors, some examples

Nodes correspond to sets of channel ends, i.e., a node envelopes one or more channel ends, and it is represented as a "bullet" (see Figure 2.8). They serve as an attaching point to either build complex connectors out of simpler ones, or to attach a connector to the ports of component instances. A channel has at least two ends: *source* and *sink* ends[4]. Channel ends have no particular graphical representation, they are simply the "extreme points" of the edge that represents the channel. A datum enters through a source channel end into its respective channel, and it leaves through a sink channel end from its respective channel. As shown in Figure 2.8, a channel is represented as a directed edge (i.e., an arrow, in order to allow one to distinguish source and sink ends). A node that envelopes only source (resp., sink) ends is called a "source node" (resp., "sink node"). Furthermore, in the ABT model of a connector, another kind of (internal/hidden) node can be present. It is called a "mixed node" and it concerns the *fusion* of sink and source nodes into one node. Fusing two nodes destroys both nodes and produces a new node that envelopes all the channel ends of the two destroyed nodes.

I/O operations are allowed on source and sink nodes only; components cannot read from or write to mixed nodes. A source node replicates every data item written to it as soon as all of its enveloped source channel ends can consume that data item. Reading from a sink node non-deterministically selects one of the data items available through its enveloped sink channel ends. A mixed node combines the behaviour of a sink node and a source node in an atomic iteration of an infinite loop: in each atomic iteration it nondeterministically selects an appropriate data item available through its enveloped sink channel ends and replicates that data item into all of its enveloped source channel ends. A data item is appropriate for selection in an iteration only if it can be consumed by all source channel ends that are enveloped by that node. In other words, source and sink nodes are open towards the external environment, while mixed nodes are closed and do not permit I/O interaction with the external environment.

As shown in [10], each of the connectors shown in Figure 2.8 can be modelled by means of ABTs (and their composition). The ABT model of a connector is defined as a composition of the ABT models of its

[4]We impose no constraint on the number of channel end types, therefore it is perfectly valid to have a channel with two source ends.

constituent channels and nodes. For a single channel (i.e., a directed edge in Figure 2.8), source and sink ends are modelled by timed data streams and the channel by the ABT relating the input/output timed data streams modelling its ends. For instance, by considering the connector shown in Figure 2.8.(a), and by assuming that each of its channels are synchronous communication channel, then each of them could be modelled as the following ABT: $Sync(\langle\alpha, a\rangle;\langle\beta, b\rangle) \equiv \langle\alpha, a\rangle = \langle\beta, b\rangle$. The sink node shown in Figure 2.8.(a) as a bullet, can be modelled as follows:

$Mrg(\langle\alpha, a\rangle,\langle\beta, b\rangle;\langle\gamma, c\rangle) \equiv$
if $a(0) < b(0)$, $\alpha(0) = \gamma(0) \wedge a(0) = c(0) \wedge Mrg(\langle\alpha', a'\rangle,\langle\beta, b\rangle;\langle\gamma', c'\rangle)$;
if $a(0) > b(0)$, $\beta(0) = \gamma(0) \wedge b(0) = c(0) \wedge Mrg(\langle\alpha, a\rangle,\langle\beta', b'\rangle;\langle\gamma', c'\rangle)$;
if $a(0) = b(0)$, $\exists t : a(0) < t\ min(a(1), b(1)) \wedge \exists r, s \in \{a(0)\ , t\} \wedge r \neq s \wedge$
$Mrg(\langle\alpha, r.a'\rangle,\langle\beta, s.b'\rangle;\langle\gamma, c\rangle)$.

Intuitively, the Mrg ABT produces an output that is a merge of its two input streams. If $\alpha(0)$ arrives before $\beta(0)$, i.e., $a(0) < b(0)$, then the ABT produces $\gamma(0) = \alpha(0)$ as its output at $c(0) = a(0)$ and proceeds with the tails of the streams in its first input timed data stream. If $\alpha(0)$ arrives after $\beta(0)$, i.e., $a(0) > b(0)$, then the ABT produces $\gamma(0) = \beta(0)$ as its output at $c(0) = b(0)$ and proceeds with the tails of the streams in its second input timed data stream. If $\alpha(0)$ and $\beta(0)$ arrive at the same time (i.e., $a(0) = b(0)$), then in this formulation Mrg picks an arbitrary number t in the open time interval $(a(0), min(a(1), b(1)))$ and uses it to nondeterministically break the tie.

Thus, the connector shown in Figure 2.8.(a), can be modelled as the following ABT composition: $Sync_1(\langle\alpha_1, a_1\rangle;\langle\beta_1, b_1\rangle^1) \circ Sync_2(\langle\alpha_2, a_2\rangle;\langle\beta_2, b_2\rangle^2) \circ Mrg(\langle\alpha, a\rangle^1,\langle\beta, b\rangle^2;\langle\gamma, c\rangle)$, where $Sync_1$ and $Sync_2$ are the ABT models for the two channels (and their ends), and Mrg is the ABT model of the sink node. Note that the hidden node enveloping the ends $\langle\beta_1, b_1\rangle^1$ and $\langle\alpha, a\rangle^1$, and the hidden node enveloping the ends $\langle\beta_2, b_2\rangle^2$ and $\langle\beta, b\rangle^2$, are mixed nodes. The other connectors in Figure 2.8 can be modelled analogously. Among them there are quite interesting cases that highlight the compositional nature of Reo connectors modelled through ABTs. For example, as shown in [10], the ABT model of the sink node shown in Figure 2.8.(b) results from the composition of two Mrg ABTs (through the production of a mixed node).

Summing up, a key property of Reo connectors is that they impose specific coordination patterns on the entities (e.g., component instances) that perform I/O operations through these connectors, without the knowledge of those entities.

2.3.2.2 Scenario modelling

Now, let us model the components and Reo connectors of the *Mary Scenario* by means of ABTs. In Figure 2.9, we show the component-connector software architecture of the *Mary Scenario*.

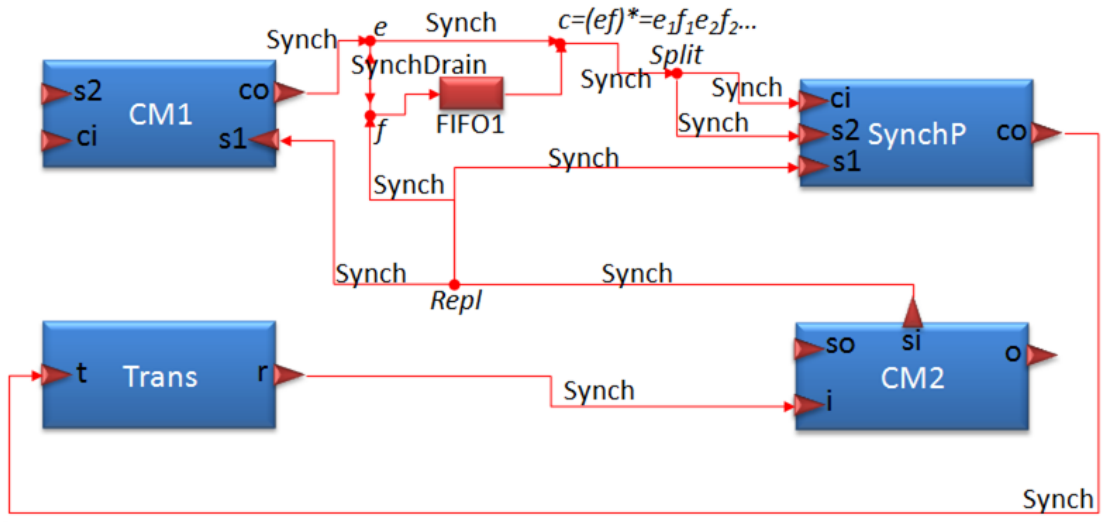

Figure 2.9: The Mary Scenario modelled by means of Reo connectors

CM1 is the component implementing Contact Manager 1, i.e., the contact management software deployed on the old device. It has three input ports, s1, s2, and ci, and one output port, co. s1 and s2 serve to select the *contact synchronisation mode*. s1 is used to inform CM1 that the environment wishes to download a contact item from CM1, s2 is used to inform CM1 that the environment wishes to upload a contact item to CM1. ci is used for receiving a contact item from the environment, co is for sending a contact item to the environment. Thus, in the *Mary Scenario*, only s1 and co are used. The ABT modelling CM1 is defined as follows:

$CM1(\langle\alpha_{CM1.s1}, a_{CM1.s1}\rangle,\langle\alpha_{CM1.s2}, a_{CM1.s2}\rangle,\langle\alpha_{CM1.ci}, a_{CM1.ci}\rangle;\langle\beta_{CM1.co}, b_{CM1.co}\rangle) \equiv$
$(a_{CM1.s1} < b_{CM1.co} < a'_{CM1.s1}) \wedge (a_{CM1.s2} < a_{CM1.ci} < a'_{CM1.s2})$.

This ABT definition specifies that the environment of CM1 can non-deterministically choose to send a datum to either s1 or s2. Once one of the two contact synchronisation input ports is chosen, a contact item is either received or sent accordingly to the chosen synchronisation port. The receiving or sending of a contact item terminates before a synchronisation port is chosen again.

CM2 is the component implementing Contact Manager 2, i.e., the contact management software deployed on the new device. It has two input ports, so and i, and two output ports, si and o. so and si serve to select the *contact synchronisation mode*. so is used to inform CM2 that the environment wishes to download a contact item from CM2, si is used by CM2 to inform the environment that it wishes to download a contact item from the environment. i is for receiving a contact item from the environment, o is for sending a contact item to the environment. Thus, in the *Mary Scenario*, only si and i are used. The ABT modelling CM2 is defined as follows:

$CM2(\langle\alpha_{CM2.so}, a_{CM2.so}\rangle,\langle\alpha_{CM2.i}, a_{CM2.i}\rangle;\langle\beta_{CM2.si}, b_{CM2.si}\rangle,\langle\beta_{CM2.o}, b_{CM2.o}\rangle) \equiv$
$(a_{CM2.so} < b_{CM2.o} < a'_{CM2.so}) \wedge (b_{CM2.si} < a_{CM2.i} < b'_{CM2.si})$.

It is analogous to the one for CM1, hence no further explanation is needed.

Trans is the Contact Translator component able to translate contacts from the format used by Synchronisation Program (deployed on Mary's laptop), to the format used by CM2. Trans has one input port, t, and one output port r. t is used to get as input a contact item, and r is used to return as output the translated contact item. The following is the ABT modelling Trans:

$Trans(\langle\alpha_{Trans.t}, a_{Trans.t}\rangle;\langle\beta_{Trans.r}, b_{Trans.r}\rangle) \equiv$
$(\beta_{Trans.r} \neq \alpha_{Trans.t}) \wedge (a_{Trans.t} < b_{Trans.r} < a'_{Trans.t})$.

Note that its I/O interaction behaviour is the same as the one of DataCompressor described above.

SynchP is the Synchronisation Program component deployed on Mary's laptop able to synchronise contacts with the old device, i.e., CM1. SynchP has three input ports, s1, s2, and ci, and one output port, co. s2 is used to inform SynchP that the environment wishes to download a contact item from SynchP, s1 is used to inform SynchP that the environment wishes to upload a contact item to SynchP. ci is for receiving a contact item from the environment, co is for sending a contact item to the environment. The ABT modelling SynchP is defined as follows:

$SynchP(\langle\alpha_{SynchP.s1}, a_{SynchP.s1}\rangle,\langle\alpha_{SynchP.s2}, a_{SynchP.s2}\rangle,\langle\alpha_{SynchP.ci}, a_{SynchP.ci}\rangle;$
$\langle\beta_{SynchP.co}, b_{SynchP.co}\rangle) \equiv$
$(a_{SynchP.s1} < a_{SynchP.ci} < a'_{SynchP.s1}) \wedge (a_{SynchP.s2} < b_{SynchP.co} < a'_{SynchP.s2})$.

In Figure 2.9, we graphically represent the composition of channels and nodes (i.e., the composition of basic connectors) that realises the complex connector that coordinates the interaction of CM1, CM2, Trans, and SynchP in order to achieve the aim of the *Mary Scenario*, i.e., correctly downloading the contact list from the old device to the new device. In Figure 2.9, channels are represented as arrows, nodes as bullet. The name of the nodes is shown by using an "italic" font, while the names of the basic connectors resulting from the composition of some channels and nodes are shown by using a "plain" font. In particular, we defined 5 nodes and 15 channels. The composition of these channels and nodes produces the complex connector for the *Mary Scenario*.

This scenario, although simple, allows us to highlight some appealing properties of Reo connectors, e.g., *compositionality*. On one hand, we could define the complex connector in a "*monolithic*" fashion, i.e., by directly composing the ABT models of those 5 nodes and 15 channels. On the other hand, we could choose to follow a compositional approach, i.e., by building the ABT model of the complex connector out of the ABT models of simpler connectors, each of them, in turn, built out of the ABT models of some of those 5 nodes and 15 channels. The result would be perfectly equivalent, that is *Reo connectors enjoy the compositionality property*. Note that this property is an essential property for effective connector reuse, thus implying also another property: *scalability*. Furthermore, compositional connectors enable

compositional reasoning, i.e., a behavioural property held by the *whole* is the result of the composition of the properties held by its *parts*, allowing one to establish properties of the whole by simply looking at the properties of its parts. Finally, compositionality directly implies *incremental* construction of connectors. It is worthwhile noticing that the converse does not always hold, i.e., two models built incrementally are not necessarily equivalent if the ordering of the constructions differ. Thus compositionality is a more desirable property than incrementality.

Coming back to the *Mary Scenario*, we choose to build the complex connector in a compositional way. In particular, by referring to Figure 2.9, $Repl$ is the source node for 3 $Synch$ channels (i.e., basic connectors). The composition of this 3 $Synch$ and $Repl$ defines a first (simple) connector:

$Repl1x3(\langle\alpha,a\rangle;\langle\beta_{CM1.s1},b_{CM1.s1}\rangle,\langle\beta_f,b_f\rangle,\langle\beta_{SynchP.s1},b_{SynchP.s1}\rangle) \equiv$
$\quad Repl(\langle\alpha,a\rangle;\langle\gamma_1,g_1\rangle^1,\langle\gamma_2,g_2\rangle^2,\langle\gamma_3,g_3\rangle^3) \circ Synch(\langle\gamma_1,g_1\rangle^1;\langle\beta_{CM1.s1},b_{CM1.s1}\rangle) \circ$
$\quad Synch(\langle\gamma_2,g_2\rangle^2;\langle\beta_f,b_f\rangle) \circ Synch(\langle\gamma_3,g_3\rangle^3;\langle\beta_{SynchP.s1},b_{SynchP.s1}\rangle)$

where

$Repl(\langle\alpha,a\rangle;\langle\gamma_1,g_1\rangle,\langle\gamma_2,g_2\rangle,\langle\gamma_3,g_3\rangle) \equiv$
$\quad \gamma_1=\alpha \wedge \gamma_2=\alpha \wedge \gamma_3=\alpha \wedge g_1=a \wedge g_2=a \wedge g_3=a.$

It is worthwhile noticing that the ABT $Repl1x3$ captures the behaviour of any connector that synchronously replicates its input stream into its three identical output streams. Thus, it could be reused also in scenarios different from the *Mary Scenario*, whenever there is the need to replicate input streams into three or more identical output streams. The case of more than three output streams, e.g., N output streams, can be compositionally realised by composing $Repl1x3$ with a $Repl1xN-2$ (that, in turn, could be built in a compositional way as well). Besides compositionality, as already mentioned in Section 2.3.2.1, this is due to another key property of Reo connectors. We call this property *context-freedom*, that is a Reo connector is *unaware* of the entities that has to coordinate. This property comes essentially from the fact that Reo connectors are modelled as ABTs and it enables the higher degree of reusability for connectors, i.e., "*reusable in any context*". Other degrees of connector reusability that we consider for the purposes of this survey are "*reusable in a parameterised context*", as in the case of the role-glue connectors described in Section 2.3.1, or "*non-reusable*".

Now, we compose $Repl1x3$ with another $Synch$ in order to build the connector $Repl1x3'$ that connects (and coordinates) the port `si` of `CM2` with the port `s1` of `CM1`, the source node f, and the port `s1` of `SynchP`:

$Repl1x3'(\langle\alpha_{CM2.si},a_{CM2.si}\rangle;\langle\beta_{CM1.s1},b_{CM1.s1}\rangle,\langle\beta_f,b_f\rangle,\langle\beta_{SynchP.s1},b_{SynchP.s1}\rangle) \equiv$
$\quad Repl1x3(\langle\alpha,a\rangle^1;\langle\beta_{CM1.s1},b_{CM1.s1}\rangle,\langle\beta,b_f\rangle,\langle\beta_{SynchP.s1},b_{SynchP.s1}\rangle) \circ$
$\quad Synch(\langle\alpha_{CM2.si},a_{CM2.si}\rangle;\langle\beta,b\rangle^1).$

Another connector that is needed for our scenario is $Split1x2$. It coordinates the sink node c with the ports `ci` and `s2` of `SynchP`, passing through the source node $Split$. The following is the ABT model of $Split1x2$:

$Split1x2(\langle\alpha_c,a_c\rangle;\langle\beta_{SynchP.ci},b_{SynchP.ci}\rangle,\langle\beta_{SynchP.s2},b_{SynchP.s2}\rangle) \equiv$
$\quad Synch(\langle\alpha_c,a_c\rangle;\langle\mu,m\rangle^1) \circ Split(\langle\alpha,a\rangle^1;\langle\beta_1,b_1\rangle^2;\langle\beta_2,b_2\rangle^3) \circ$
$\quad Synch(\langle\gamma_2,g_2\rangle^2;\langle\beta_{SynchP.ci},b_{SynchP.ci}\rangle) \circ Synch(\langle\gamma_3,g_3\rangle^3;\langle\beta_{SynchP.s2},b_{SynchP.s2}\rangle)$

where

$Split(\langle\alpha,a\rangle;\langle\beta_{SynchP.ci},b_{SynchP.ci}\rangle;\langle\beta_{SynchP.s2},b_{SynchP.s2}\rangle) \equiv$
$\quad \forall h{:}(i{=}2h \wedge \alpha(i){=}\beta_{SynchP.ci}(i) \wedge a(i){=}b_{SynchP.ci}(i)) \wedge$
$\qquad (\alpha(j){=}\beta_{SynchP.s2}(j) \wedge a(j){=}b_{SynchP.s2}(j) \wedge j{=}2h+1).$

$Split1x2$ captures the behaviour of any connector that synchronously splits its input stream into two different output streams. One output stream is made by the even data items of the input stream, the other by the odd data items.

The last connector that we need for our scenario is $Ordering$. It coordinates the source nodes e and f with the sink node c and it results from the composition of these nodes with the $Synch$, $SyncDrain$, and $FIFO_1$ channels. Note that, for this composition, no stream fusion is needed.

$Ordering(\langle\alpha_e,a_e\rangle,\langle\alpha_f,a_f\rangle;\langle\beta_c,b_c\rangle) \equiv$
$\quad SyncDrain(\langle\alpha_e,a_e\rangle,\langle\alpha_f,a_f\rangle;) \circ FIFO_1(\langle\alpha_f,a_f\rangle;\langle\beta_c,b_c\rangle) \circ Synch(\langle\alpha_e,a_e\rangle;\langle\beta_c,b_c\rangle)$

where

$SyncDrain(\langle\alpha_e,a_e\rangle,\langle\alpha_f,a_f\rangle;) \equiv a_e{=}a_f$

and

$FIFO_1(\langle\alpha_f,a_f\rangle;\langle\beta_c,b_c\rangle) \equiv \alpha_f{=}\beta_c \wedge a_f{<}b_c{<}a'_f.$

The behaviour of $Ordering$ can be seen as imposing an order on the flow of the data items written to e and f, through to c: the data items obtained by successive read operations on c consist of the first data item written to e, followed by the first data item written to f, the second data item written to e, followed by the second data item written to f, etc. That is, the coordination pattern imposed by $Ordering$ can be summarised as $c = (ef)*$, meaning that the sequence of values that appear through c consists of zero or more repetitions of the pairs of values written to e and f, in that order. $FIFO_1$ is a first-in-first-out queue with one-place buffer. $SyncDrain$ is a channel with two source ends. Since it has no sink end, it is not for producing data items. Instead, it is for synchronising the write operations on e and f, i.e., to force these operations to wait for each other until they are both ready to write.

Now, we can compose $Ordering$, $Split1x2$, $Repl1x3'$, and 3 $Synch$ channels together to build the complex connector, $MaryConnector$, that coordinates the I/O interaction between the ports `co` and `s1` of `CM1`, `s1`, `s2`, `ci` and `co` of `SynchP`, `si` and `i` of `CM2`, and, `t` and `r` of `Trans`:

$$
\begin{aligned}
&MaryConnector(\langle\alpha_{CM1.co}, a_{CM1.co}\rangle,\langle\alpha_{SynchP.co}, a_{SynchP.co}\rangle,\langle\alpha_{CM2.si}, a_{CM2.si}\rangle,\langle\alpha_{Trans.r}, a_{Trans.r}\rangle;\\
&\quad\langle\beta_{CM1.s1}, b_{CM1.s1}\rangle,\langle\beta_{SynchP.ci}, b_{SynchP.ci}\rangle,\langle\beta_{SynchP.s2}, b_{SynchP.s2}\rangle,\langle\beta_{SynchP.s1}, b_{SynchP.s1}\rangle,\\
&\quad\langle\beta_{Trans.t}, b_{trans.t}\rangle,\langle\beta_{CM2.i}, b_{CM2.i}\rangle) \equiv\\
&\qquad Synch(\langle\alpha_{CM1.co}, a_{CM1.co}\rangle;\langle\beta_e, b_e\rangle^1) \circ Ordering(\langle\alpha_e, a_e\rangle^1,\langle\alpha_f, a_f\rangle^2;\langle\beta_c, b_c\rangle^3) \circ\\
&\qquad Repl1x3'(\langle\alpha_{CM2.si}, a_{CM2.si}\rangle;\langle\beta_{CM1.s1}, b_{CM1.s1}\rangle,\langle\beta_f, b_f\rangle^2,\langle\beta_{SynchP.s1}, b_{SynchP.s1}\rangle) \circ\\
&\qquad Split1x2(\langle\alpha_c, a_c\rangle^3;\langle\beta_{SynchP.ci}, b_{SynchP.ci}\rangle,\langle\beta_{SynchP.s2}, b_{SynchP.s2}\rangle) \circ\\
&\qquad Synch(\langle\alpha_{SynchP.co}, a_{SynchP.co}\rangle;\langle\beta_{Trans.t}, b_{Trans.t}\rangle) \circ\\
&\qquad Synch(\langle\alpha_{Trans.r}, a_{Trans.r}\rangle;\langle\beta_{CM2.i}, b_{CM2.i}\rangle).
\end{aligned}
$$

Note that, in defining $MaryConnector$, stream fusion is used only for the composition of $Ordering$, $Repl1x3'$, $Split1x2$, and one $Synch$. According to Figure 2.9, this is due to the fact that $MaryConnector$ is essentially composed by three independent (sub-)connectors, i.e., a complex connector and 2 $Synch$ channels. We recall that Mary's aim is to correctly transfer her contact list from the old device to the new one. $MaryConnector$ is for this purpose. Note that in order to achieve it, by means of Reo connectors modelled through ABTs, we reason in a compositional way. First of all, we exploit the assumption that `CM1` is able to synchronise its contacts with `SynchP`. We recall that this assumption holds since `SynchP` is the contact synchronisation program provided with the old device `CM1` and deployed on Mary's laptop. Thus, under this assumption, we temporarily move the original problem to the problem of transferring the contacts stored into `SynchP` to the new device `CM2`. This problem can be easily solved by exploiting the contact translator component `Trans`. On one hand, it is enough to make `SynchP` able to interact with `Trans` in order to send the contacts from `SynchP` to `Trans`. This requires only a $Synch$ channel between port `co` of `SynchP` and port `t` of `Trans`. On the other hand, `Trans` has to interact with `CM2` in order to send the translated contacts to the latter. Again, this requires only a $Synch$ channel between port `r` of `Trans` and port `i` of `CM2`. Now, a part of the problem is solved and what still remains to be solved is how to make `CM2` able to synchronise contacts with `CM1`, through `SynchP`, although `CM2` and `CM1` are not natively built for this purpose. The solution is represented by the complex connector resulting from the composition of $Ordering$, $Repl1x3'$, $Split1x2$, and one $Synch$. In our scenario, `CM2` is the initiator of the interaction. It cannot directly synchronise contacts with `CM1` that, in turn, can only directly synchronise contacts with `SynchP`. Thus the built complex connector, through its sub-connector $Repl1x3'$, allows:

(i) `CM2` to inform its environment, i.e., `CM1` and `SynchP`, that it wants to receive contacts;

(ii) `CM1` to be informed that its environment, i.e., `SynchP`, wants to receive contacts; and

(iii) `SynchP` to be informed that its environment, i.e., `CM1`, wants to upload contacts.

Meanwhile, a *synchronisation signal* is sent to the node f of $Ordering$ hence storing it in the $FIFO_1$ buffer. In other words, $Repl1x3$ allows `CM2` to initiate and suitably coordinate an I/O interaction between `CM1` and `SynchP`. This is the only way for `CM2` to (indirectly) receive contacts from `CM1`.

Thus, `CM1` starts to send contacts to `SynchP`, which is waiting for these contacts. This happens through the (sub-)connector resulting from the composition of $Ordering$ and $Split1x2$. $Ordering$ allows the production of a stream of *contact-from-CM1*, *synchronisation-signal-for-SynchP*, ... This stream is then split by $Split1x2$ in order to allow `SynchP`:

(i) to receive the sent contacts; and

(ii) at the same time, to be informed that its environment, i.e., `Trans`, wants to receive those contacts.

Note that, by referring to the download of contacts from `SynchP`, `Trans` is, now, the environment for `SynchP` thanks to the $Synch$ channel among them.

Summing up, $MaryConnector$, once attached to the components of the *Mary Scenario*, coordinates

the I/O interaction initiated by CM2 in order to force CM1 to exchange contacts with SynchP. The I/O interaction of SynchP is, in turn, coordinated to send the received contacts to Trans that translates and sends them to CM2, hence achieving Mary's goal.

Consequently, the definition of the ABT model of the system depicted in Figure 2.9 is straightforward, so we choose to omit it for brevity.

2.3.2.3 Advantages and disadvantages

In this section, we summarise the advantages and disadvantages, in part already exhibited by the *Mary Scenario* described in Section 2.3.2.2, of using Reo connectors modelled as ABTs. The main benefits can be summarised as follows:

- **Compositionality**: Reo connectors are compositional. This means that they can be incrementally built out of simpler connectors and it does not matter in what way we conduct this incremental construction, the result is always the same. More formally, the ABT composition operator $\circ$ is associative and commutative. Indeed associativity of $\circ$ is the necessary property for enabling the compositional construction of connectors [44].
- **Incrementality**: directly implied by compositionality.
- **Scalability**: it is implied by compositionality and by the fact that any Reo connector (simple or complex) is essentially a composition of very basic synchronous channels, each of them with only two ends, i.e., a source and a sink.
- **Compositional reasoning**: a behavioural property held by the *whole* is the result of the composition of the properties held by its *parts*, hence allowing one to establish properties of the whole by simply looking at the properties of its parts.
- **Reusability**: a Reo connector is *completely unaware* of the entities that it has to coordinate. This property comes essentially from the fact that Reo connectors are modelled as ABTs. Thus, a Reo connector is *always reusable*.
- **Evolution**: Reo connectors have an inherently dynamic topology in the sense that they support the construction of *open systems*. That is, a system configuration can dynamically change due to connection/disconnection of connectors and component instances, hence supporting run-time reconfiguration of connectors.
- **Reo provides tool support**: there are few existing tools that provide a graphical notation for architectural analysis whose semantics are defined by the ABT model. For instance, the Eclipse Coordination Tools (ECT) [4], just to mention one of them, provides a GUI to facilitate the development of Reo connector systems. The GUI provides the user with a repository of relevant basic Reo connectors (e.g., synchronous, synchronous drain, FIFO and lossy channels) that can be used similarly to a system of primitive types in any programming language. Users can add components and links to the connector system to have a complete view of the overall system. Furthermore, Flash animation facilities are provided in order to experiment with the behaviour of the specified model. It is also possible to generate connectors from other specification language models like UML or BPMN automatically.

Despite the above appealing properties of Reo connectors there are also few points against to take into account:

- **No support for non-functional property specification**: at least for the component/connector model proposed in [10], in defining the component/connector interaction, no non-functional property is taken into account. This prevents the analysis of properties such as *performance* and *reliability*. Indeed, in Section 2.4.1, we briefly describe an extension of the work described in [10]. The aim of this extension, described in detail in [13], is to model also QoS attributes of both components and Reo connectors. The work described in [13] cannot be considered as a mere extension of the work described in [10] since it defines a semantic model for Reo connectors different from ABTs, i.e., it

is an operational model based on a QoS extension of *constraint automata* [20] called *Quantitative Constraint Automata*. For this reason, in this survey, we discuss the two works separately.

We conclude this "*pros. vs cons.*" analysis of Reo connectors by relating them with the role-glue connectors described in Section 2.3.1. By referring to Section 2.3.1, we recall that a connector is characterised by its *glue* that dictates how the activities of the *roles* have to be coordinated. Although this vision eases the automatic analysis of connectors, i.e., compatibility check, deadlock-freedom analysis, etc., note that the *glue* corresponds to a centralised coordination entity, while a Reo connector corresponds to possibly many distributed coordination entities. This is why Reo connectors are compositional and role-glue connectors are not, although, as discussed in Section 2.3.1, role-glue connectors support a kind of compositional reasoning under certain conditions. Furthermore, in contrast to Reo connectors, role-glue connectors have a constrained reusability. That is, as discussed in Section 2.3.1, a role-glue connector can be reused under *port-role compatibility*. This is due to the fact that the *roles*, by defining the obligations of the components participating in the interaction, give an abstract characterisation of the possible contexts in which the connector can be reused. This is why role-glue connectors are *partially reusable*, while Reo connectors are always reusable in any context. This means that Reo connectors allow for reusing the structure of the interaction since they model the *means of interaction* rather than *the interaction* itself, as it is for role-glue connectors. This allows Reo connectors to be *completely unaware* of the entities that they have to coordinate, contrariwise to role-glue connectors that, due to the *roles* specification, are abstractly aware of the entities that they have to coordinate. However, the above considerations imply that the semantic correctness of the reuse may not be assessed by construction.

2.3.3 Kell calculus

In this section we discuss a process algebra called the "*Kell calculus*" [24, 60, 25, 67]. Indeed, the Kell calculus is a family of process calculi rather than a single calculus of processes. It has been intended as a basis for studying distributed (and ubiquitous) component-based programming. As it is done for the other component/connector modelling approaches presented in this survey, the discussion will be conducted by first reporting and summarising the basic notions and definitions concerning the Kell calculus (Section 2.3.3.1). This summary is based on the information collected from the most significant literature about the Kell calculus [24, 60, 25, 67], thus many definitions and concepts are borrowed from these works. Then, in Section 2.3.3.2, we model the *Mary Scenario* by means of the formal tools provided by the Kell calculus. Finally, in Section 2.3.3.3, we discuss advantages and disadvantages of the calculus with respect to component/connector modelling and analysis.

Before embarking on a discussion specific to the Kell calculus, let us briefly discuss its origins. The core of the Kell calculus is the original π-calculus [52, 54, 55]. Essentially, the Kell calculus is an extension of the π-calculus. The work on the π-calculus began with the need of enhancing a previous process calculi called *Calculus of Communicating Systems* [51] (CCS) in order to achieve an algebraic formulation of the different forms of process *mobility* (e.g., logical and physical mobility) in distributed systems. The main idea consisted of adding a new syntactical construct, the *channel*, and new semantic reduction rules for the handling of channels. This led to a first version of the π-calculus. Despite the idea of allowing only channels to be the *content of the communication* has been demonstrated, by this first version, sufficient to achieve mobility, later, this initial version has been extended by following an *high-order approach*. That is, mobility can also be achieved by the powerful means of transmitting *processes* (and not only channels) as messages. Summing up, the current π-calculus [52, 54, 55] (by not considering all its variants existing in the literature) allows an algebraic formulation of (i) distributed systems, where the primary building blocks are independent threads of control, called processes, that can interact through *named* communication channels by performing atomic I/O actions on these channels (i.e., primitive *send* and *receive*); and of (ii) their mobility aspects by allowing both processes and channels to be the content of a sent or received message.

The Kell calculus, whose core is π-calculus, is a family of higher-order process calculi with hierarchical *localities* and *locality passivation*, which is indexed by the pattern language used in input constructs.

The word "kell" is a variation on the word "cell", in a loose analogy with biological cells, and denotes a locality or locus of computation, e.g., a network, a computer, a component, etc.

The Kell calculus has been introduced, as an extension of the π-calculus, to study programming models for wide-area distributed systems and component-based systems. In particular, the Kell calculus originates from the need to formulate *modular dynamicity*, i.e., the ability to modify a running system by replacing some of its components, or by introducing new components. As it is stated in [24], "*The Kell calculus can be seen as an attempt to understand the operational basis of modular dynamicity: localities in the Kell calculus model named components, and locality passivation provides the basis for dynamic reconfiguration operations*". A main design principle of the calculus is to keep all actions "local" in order to facilitate its distributed implementation. That is, an implementation of the calculus should not need to consider atomic actions occurring across wide-area networks, i.e., distributed synchronisation between two sites, which is notoriously costly. This is in contrast to what is done for *Mobile Ambients* [30] that is a variant of π-calculus. Thus, beyond the need for modular dynamicity, the Kell calculus originates also from the need to overcome issues concerning distributed synchronisation among different localities that are typical of other calculi such as Mobile Ambients.

This premise allows us to summarise the motivations and contributions of the Kell calculus by means of the following *WHAT-WHY-HOW* characterisation of the work described in [24, 60, 25, 67].

- **WHAT -** To develop a formal calculi intended as a basis for studying distributed component-based systems with respect to behavioural aspects and issues concerning mobility and modular dynamicity.
- **WHY -** π-calculus, as it originally is, does not allow the handling of modular dynamicity and all the variants of the π-calculus conforming to Mobile Ambients are based on distributed synchronisation between different sites that is costly to implement.
- **HOW -** By extending π-calculus with the concepts of *locality* and *locality passivation* and by enforcing the *locality principle*, i.e., keeping all actions local.

2.3.3.1 Overview

As already mentioned, the Kell calculus is a family of calculi that vary as the chosen language of input patterns varies. Thus, once one has chosen a pattern language has been decided upon, an instance of the Kell calculus is obtained. In this section we report and discuss the syntax and the operational semantics of the instance of the Kell calculus presented in [24].

Five kinds of input patterns are allowed: *kell patterns*, that match a subkell containing a process; *local patterns*, that match a local message received from a channel; *up patterns*, that match a message to a kell from its parent kell; and two kinds of *down patterns*, that match a message to a kell from either any of its subkell or a specific one through a channel. The syntax of patterns is given below, accordingly to the order that we used above to list them:

$\xi ::= a[x] \mid c\langle\tilde{u}\rangle \mid c\langle\tilde{u}\rangle^{\uparrow} \mid c\langle\tilde{u}\rangle^{\downarrow} \mid c\langle\tilde{u}\rangle^{\downarrow^a}$

$a[x]$ denotes a kell named with a and containing a named term x. $c\langle\tilde{u}\rangle$ denotes a received message, whose content is $\tilde{u}$, exchanged locally to a kell through the channel c. $\tilde{u}$ is a vector of named terms u. We recall that, in Kell calculus, a process can be a named term. $c\langle\tilde{u}\rangle^{\uparrow}$ denotes a message, whose content is $\tilde{u}$, received from the parent kell through the channel c. $c\langle\tilde{u}\rangle^{\downarrow}$ denotes a message, whose content is $\tilde{u}$, received from any subkell through the channel c. $c\langle\tilde{u}\rangle^{\downarrow^a}$ denotes a message, whose content is $\tilde{u}$, received from the subkell a through the channel c.

The syntax of Kell calculus processes (P) is given below, note that it uses input patterns:

$P ::= \mathbf{0} \mid \xi \triangleright P \mid \nu x.P \mid P|P \mid a[P] \mid c\langle\tilde{P}\rangle$

$P_* ::= \mathbf{0} \mid \xi \triangleright P \mid P_*|P_* \mid a[P_*] \mid c\langle\tilde{P}\rangle$

$\xi ::= a[x] \mid c\langle\tilde{u}\rangle \mid c\langle\tilde{u}\rangle^{\uparrow} \mid c\langle\tilde{u}\rangle^{\downarrow} \mid c\langle\tilde{u}\rangle^{\downarrow^a}$

$u ::= x \mid (x)$

0 denotes the *null* process, i.e., the process doing nothing.

$\xi \triangleright P$ denotes a *trigger*. It is the process that expects an input conforming to the input pattern ξ and, once received it, behaves as P.

A pattern ξ acts as a *binder* in the calculus. Named terms, or simply *names*, x, i.e., *name constants*, *name variables* or *process variables*, that do not occur within parentheses $()$ in a pattern are bound by the pattern. Instead names occurring in a pattern within parentheses, e.g., (x), are *not* bound in the pattern.

$\nu x.P$, , "new x in P", denotes the *restriction* operator. It restricts the use of the name x only to P. Another way of describing it is that it declares a new unique name x, distinct from all external names, for use in P. In other words, suppose that x is *free*, i.e., it is not bound, in P, then $\nu x.P$ makes x bound in P.

$P|P$ denotes the parallel composition of Kell calculus processes.

P_* denotes all the processes in *normal form*. A Kell calculus process is in normal form when it does not contain any name restriction operator.

A Kell calculus term is evaluated with respect to an *evaluation context*. Thus to finalise the presentation of the syntax of the Kell calculus, we report below the syntax of evaluation contexts (ranged over **E**,...):

$\mathbf{E} ::= \cdot \mid \nu x.\mathbf{E} \mid a[\mathbf{E}] \mid P|\mathbf{E}$

Filling the hole $\cdot$ in, an evaluation context **E** with a Kell calculus term R results in a Kell calculus term denoted by $\mathbf{E}\{R\}$.

The operational semantics of the Kell calculus are defined in the CHAM style [22], via a structural equivalence relation and a reduction relation. The structural equivalence $\equiv$ is the smallest equivalence relation that verifies the rules in Figure 2.10 and that makes the parallel operator $|$ associative and commutative, with **0** as a neutral element. Note that this implies that the Kell calculus is *compositional*. The reduction relation $\longrightarrow$ is the smallest binary relation on the calculus that satisfies the rules given in Figure 2.11.

$$\nu a.\mathbf{0} \equiv \mathbf{0} \ [\text{S.NU.NIL}] \qquad \nu a.\nu b.P \equiv \nu b.\nu a.P \ [\text{S.NU.COMM}]$$

$$\frac{a \notin \mathtt{fn}(Q)}{(\nu a.P) \mid Q \equiv \nu a.P \mid Q} \ [\text{S.NU.PAR}] \qquad \frac{P =_\alpha Q}{P \equiv Q} \ [\text{S.}\alpha] \qquad \frac{P \equiv Q}{\mathbf{E}\{P\} \equiv \mathbf{E}\{Q\}} \ [\text{S.CONTEXT}]$$

Figure 2.10: Structural equivalence

The rules shown in Figure 2.10 are simple *rewriting rules* and, hence, they do not deserve further discussion. Since, in this section, for the sake of simplicity, we are not reporting all the formal details presented [24, 60, 25, 67], the only two aspects that need at least an informal explanation are $\mathtt{fn}(Q)$ in the rule S.NU.PAR and $=_\alpha$ in the rule S.α. Let Q be a Kell calculus term, $\mathtt{fn}(Q)$ denotes the set of *free names* of Q, i.e., names that are not bound. Thus, the intuitive meaning of rule S.NU.PAR, is that one can force a Kell calculus process P to synchronise with another process Q through the channel a by means of the restriction operator. The intuitive meaning of rule S.α is that a process P can be rewritten as (i.e., is equivalent to) a process Q if there exists a *substitution* α of names and process variables in P (to names) such that the application of α to P results in Q. Furthermore it is worthwhile noticing that there is no structural equivalence rule that deals with *scope extrusion* beyond a kell boundary, as instead it is done in Mobile Ambients with the rule $a[\nu b.P] \equiv \nu b.a[P]$, provided $b \neq a$. This is due to avoid the issue discussed above concerning distributed synchronisation between different localities.

However, such name extrusion is still needed to allow communication across kell boundaries. The solution adopted in [24] is to allow only scope extrusion across kell boundaries and to restrict passivation to processes without name restriction in evaluation context (i.e., processes in normal form). Formally, by referring to Figure 2.11, this is achieved by requiring a process to be in normal form (P_*) in rule R.PASS and by adding a scope extrusion sub-reduction relation $\xrightarrow{\equiv}$. Rules R.IN and R.OUT govern the crossing of kell boundaries. Only messages may cross a kell boundary. In rule R.IN, a trigger receives a message from the outside of the enclosing kell. In rule R.OUT, a trigger receives a message from a subkell. $Q\varphi$ is the process resulting from the application of the substitution φ to Q.

2.3.3.2 Scenario modelling

In this section we model a variant of the *Mary Scenario* by means of the instance of the Kell calculus described in Section 2.3.3.1. We choose a variant of the scenario that allows us to highlight the peculiarities of the Kell calculus as modelling notation, that is the explicit modelling of *mobility* and *modular dynamicity* (i.e., *reconfiguration*), while keeping the scenario's aim and main functionalities unchanged.

Let us suppose that the old device is connected to Mary's laptop via bluetooth. A contact manager application, $CM1$, is deployed on the old device. A synchronisation program, $SynchP$, is deployed on

$$\frac{a \neq b}{a[\nu b.P] \stackrel{\equiv}{\rightarrow} \nu b.a[P]}\ [\text{SR.NEW}] \qquad \frac{P \stackrel{\equiv}{\rightarrow} P'}{E[P] \stackrel{\equiv}{\rightarrow} E[P']}\ [\text{SR.CONTEXT}]$$

$$\frac{P' \equiv P \qquad P \stackrel{\equiv}{\rightarrow} Q \qquad Q \equiv Q'}{P' \stackrel{\equiv}{\rightarrow} Q'}\ [\text{SR.STRUCT}]$$

$$\frac{\widetilde{v} = \widetilde{u}\varphi}{c\langle\widetilde{v}\rangle \mid b[R \mid (c\langle\widetilde{u}\rangle^{\uparrow} \triangleright Q)] \rightarrow b[R \mid Q\varphi]}\ [\text{R.IN}] \qquad \frac{\widetilde{v} = \widetilde{u}\varphi}{c\langle\widetilde{v}\rangle \mid (c\langle\widetilde{u}\rangle \triangleright Q) \rightarrow Q\varphi}\ [\text{R.LOCAL}]$$

$$\frac{\widetilde{v} = \widetilde{u}\varphi \qquad \downarrow^{\bullet} = \downarrow^{b} \wedge \downarrow^{\bullet} = \downarrow}{b[c\langle\widetilde{v}\rangle \mid R] \mid (c\langle\widetilde{u}\rangle^{\downarrow^{\bullet}} \triangleright Q) \rightarrow b[R] \mid Q\varphi}\ [\text{R.OUT}]$$

$$\frac{}{a[P_*] \mid (a[x] \triangleright Q) \rightarrow Q\{P_*/x\}}\ [\text{R.PASS}] \qquad \frac{P \rightarrow Q}{\mathbf{E}\{P\} \rightarrow \mathbf{E}\{Q\}}\ [\text{R.CONTEXT}]$$

$$\frac{P' \equiv P \qquad P \rightarrow Q \qquad Q \equiv Q'}{P' \rightarrow Q'}\ [\text{R.STRUCT}] \qquad \frac{P' \stackrel{\equiv}{\rightarrow}^{*} P \qquad P \rightarrow Q}{P' \rightarrow Q}\ [\text{R.STRUCT.EXTR}]$$

Figure 2.11: Reduction Relation

Mary's laptop. The new device, where another contact manager application, i.e., $CM2$, is deployed, has a bluetooth network interface as well. Thus, the old device, the new one, and Mary's laptop can constitute a very basic *intranet*. Furthermore, let us suppose that Mary's laptop can connect to the *Internet*. We also suppose that a contact translator application ($Trans$), suitable for the aims of the *Mary Scenario*, is available on the Internet. This contact translator has a limited licensing model that prevents a user from using it more than twice. Summing up the connector to be modelled, K, downloads $Trans$ from the Internet to use it locally and when the $Trans$ license expires, K self-reconfigures by downloading and using a new instance of $Trans$.

The following Kell calculus process definitions are the processes modelling the components of the *Mary Scenario*, the connector, and the whole system.

Translator ($Trans$):
$T ::= t\langle x\rangle^{\uparrow} \triangleright r\langle y\rangle \triangleright t\langle x\rangle^{\uparrow} \triangleright r\langle y\rangle \triangleright expire\langle z\rangle \triangleright T$
$Trans ::= tr[\nu t.\nu r.T]$

The contact translator application runs in a process modelled by the kell tr. It interacts with the environment through two channels t and r. The former is for receiving contacts, the latter is for sending translated contacts. $expire$ models the channel through which the translator notifies its licence expiration.

Synchronisation Program ($SynchP$):
$SP ::= sp\_s1\langle s'\rangle^{\uparrow} \triangleright sp\_ci\langle x\rangle^{\uparrow} \triangleright SP \mid sp\_s2\langle s''\rangle^{\uparrow} \triangleright sp\_co\langle x\rangle \triangleright SP$
$SynchP ::= synchp[\nu sp\_s1.\nu sp\_ci.\nu sp\_s2.\nu sp\_co.SP]$

The contact synchronisation application runs in a process modelled by the kell $synchp$. It interacts with the environment through the channels: sp_s1 for setting up a "*receive contacts*" synchronisation mode; sp_ci to receive contacts; sp_s2 to set up a "*send contacts*" mode; and sp_co to send contacts.

Contact Manager 1 ($CM1$), i.e., the old device:
$CM1' ::= cm1\_s1\langle s'\rangle^{\uparrow} \triangleright cm1\_co\langle x\rangle \triangleright CM1' \mid cm1\_s2\langle s''\rangle^{\uparrow} \triangleright cm1\_ci\langle x\rangle^{\uparrow} \triangleright CM1'$
$Link ::= \nu cm1\_co.\nu sp\_ci.cm1\_co\langle x\rangle^{\downarrow_{cmgr1}} \triangleright sp\_ci\langle x\rangle \triangleright Link$
$CM1 ::= cmgr1[\nu cm1\_s1.\nu cm1\_co.\nu cm1\_s2.\nu cm1\_ci.CM1'] \mid Link$

The contact manager deployed on the old device has a behaviour that is symmetric with respect to the

behaviour of $SynchP$. This is due to the fact that $CM1$ is the contact manager of the old device that is equipped with the synchronisation program $SynchP$. Thus, for this reason, $CM1$ can also directly communicate and interact with $SynchP$. This aspect is modelled by the $Link$ sub-process of $CM1$.

Contact Manager 2 ($CM2$), i.e., the new device:
$CM2' ::= cm2\_so\langle s' \rangle^{\uparrow} \rhd cm2\_o\langle y \rangle \rhd CM2' \mid cm2\_si\langle s'' \rangle \rhd cm2\_i\langle y \rangle^{\uparrow} \rhd CM2'$
$CM2 ::= cmgr2[\nu cm2\_so.\nu cm2\_o.\nu cm2\_si.\nu cm2\_i.CM1']$

The contact manager deployed on the new device has a behaviour analogous to $CM1$. One difference is that it cannot directly interact with $SynchP$.

Mary Connector (K):
$K ::= connection^{\uparrow}\langle \mathrm{X} \rangle \rhd (K' \mid \mathrm{X} \mid (expire^{\downarrow}\langle z \rangle \rhd K))$
$K' ::= cm2\_si\langle s'' \rangle^{\downarrow_{cmgr2}} \rhd cm1\_s1\langle s' \rangle \rhd sp\_s1\langle s' \rangle \rhd sp\_s2\langle s'' \rangle \rhd sp\_ci\langle x \rangle^{\downarrow_{synchp}} \rhd t\langle x \rangle \rhd$
$r\langle y \rangle^{\downarrow_{tr}} \rhd cm2\_i\langle y \rangle \rhd K'$

The connector exploits an Internet connection to download a process denoted with the process variable X. For instance, if K synchronises with a $connection\langle Trans \rangle$ from a super-kell, then for the rule R.IN (see Section 2.3.3.1), K is reduced to the process $K' \mid Trans \mid (expire^{\downarrow}\langle z \rangle \rhd K)$. That is, through the Internet connection, K downloads $Trans$ to locally use it for translating contacts. When the license of $Trans$ expires, K self-reconfigures to became the original process, i.e., the one waiting for downloading a new instance of $Trans$. K' is the sub-process coordinating the components of the *Mary scenario* according to the scenario's main goal, i.e., transferring contacts from the old device to the new one by exploiting the contact translator and the synchronisation program.

The system for the *Mary Scenario* (S):
$S ::= Internet[connection\langle Trans \rangle \mid Intranet]$
$Intranet ::= intranet[(\nu cm1\_s1.\nu sp\_s1.\nu sp\_s2.\nu sp\_ci.\nu t.\nu r.\nu cm2\_i.K) \mid SynchP \mid CM1 \mid CM2]$

The whole system, for the variant of the *Mary Scenario* described in this section, is a distributed system where the translator is deployed on the Internet and the other components (plus the connector) are deployed on the devices constituting the very basic intranet at Mary's place.

2.3.3.3 Advantages and disadvantages

In this section we analyse advantages and disadvantages of using the Kell calculus described in [24, 60, 25, 67] as modelling notation for components and connectors. This is done with respect to the dimensions identified in Section 2.1 plus a new aspect of the **evolution** dimension, i.e., **mobility**, that is not taken into account by the approaches described above.

The parallel composition operator "|" of the Kell calculus is defined in a way such that it results in an associative and commutative operator. This implies that Kell calculus connectors enjoy ***Compositionality*** and, hence **incrementality**, **scalability**, and **compositional reasoning** as well. Furthermore, as highlighted by the *Mary Scenario* presented in Section 2.3.3.2, the Kell calculus allows one to explicitly model **mobility** (i.e., process migration) and **dynamism** (i.e., modular dynamicity or reconfiguration) in the coordination logic of a connector. Thus, it allows to completely support **evolution**.

Despite the above mentioned benefits, with respect to the dimensions that we consider in this survey to evaluate connector modelling approaches, Kell calculus connectors are not ***always reusable*** differently from, e.g., Reo connectors. This means that a connector modelled as a Kell calculus process is not always reusable. However, note that it can be parameterised with respect to process and channel variables used in the Kell calculus definition of the connector, as shown in Section 2.3.3.2 for K. Thus, analogously to *role-glue* connectors, a Kell calculus connector can be reused whenever the Kell calculus terms used to instantiate the process and channel variables in the connector definition fit the behaviour protocol intended by the connector for these process and channel variables. Furthermore, as it is, the Kell calculus do not take into account **non-functional properties** of a process's behaviour. Although very powerful, the Kell calculus is *quite theoretical* and, as far as we know, there is **no tool support** for it.

2.3.4 BIP component framework

In [27], the authors identify the lack of a unified paradigm for describing and analysing the coordination between components. The authors motivate their work by claiming that nowadays *"only dispersed low-level coordination mechanisms (e.g., semaphores, monitors, RPC, message passing, etc)*" exist. To tackle this problem in [27] the authors propose an algebra whose terms model connectors as relations among typed ports of components. The following WHAT-WHY-HOW items summarise the work and the motivations for it:

- **WHAT -** To model the structure of the interaction in component based systems by using a control-flow paradigm. By specifically considering component based system implemented in the Behaviour-Interaction-Priority (BIP) [21, 66] framework, the main goal of the approach is to formally describe coordination among components in terms of involved communication ports. The proposed approach formalises notions of BIP and mechanisms that the BIP engine uses for computing interaction and for controlling the execution flow by coordinating (active ports of) components.
- **WHY -** To be able to model and analyse the structure of the external interaction among software components independently from computation by using a simple and powerful algebraic framework. Moreover, this work is motivated by the lack of a unified paradigm to form the bases for a common semantic model (e.g., to be used for comparing different coordination mechanisms) that should allow for a homogenised comparison of (otherwise unrelated) different architectural specifications.
- **HOW -** By defining an algebra of connectors and a graphical (hierarchical) representation for them to formally model stateless connectors and to structure the interaction. A connector models relationships among communication ports with synchronisation types - i.e., *trigger* or *synchron*. Trigger ports can initiate an interaction, whereas synchron ports need to synchronise with other ports to be able to interact. At a given state a component port can be either active or inactive.

As already said, the approach in [27] considers component based systems built on top of the BIP component framework. Within this framework a system is modelled by specifying three layers: (*i*) the *Behaviour* of each *atomic* component in terms of (possible infinite) sequences of active ports, (*ii*) the possible *Interaction* among components in terms of involved active ports, and (*iii*) the *Priorities* for selecting and scheduling interactions.

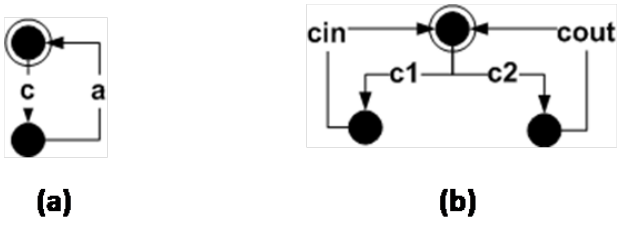

Figure 2.12: BIP behaviour automata

In BIP an atomic component has a set of communication ports that, in a given (local) state, can be active or inactive. The *Behaviour* of each component is specified in terms of an automaton where each transition is labelled with (a subset of) its communication ports (i.e., an interaction - see below). In a given state the labels of the outgoing transitions identify the set of active ports. For instance, a component with two ports c and a might have associated the automaton shown in Figure 2.12(a). This automaton specifies that the ports c and a are alternatively in an active state and, hence, that it is possible to communicate with the component by interacting alternatively with either the port c or the port a. In BIP, a system can then be modelled as (the composition of) a set of atomic components modelled by a set of labelled transition systems.

On top of the behaviour specification, the *Interaction* specification models structured connectors relating communication ports along with their synchronisation types (i.e., trigger or synchron). In other words, this layer specifies the interactions allowed by the connectors. Given a set of component communication ports, an interaction is represented by any non empty subset of these ports. For instance, the set of sets

{{c, c1}, {c, c2}, {a, cin}, {a, cout}} represents four *allowed* interactions through the ports c, c1 and c2, and the ports a, cin and cout of the components whose behaviour is modelled by the automata (a) and (b) in Figure 2.12. In Section 2.3.4.1 we describe how the notion of interaction is formally defined by using the algebra of interaction proposed in [27]. In Section 2.3.4.1 we also describe how the algebra of interaction is used as basis to provide an algebraic formalisation of (hierarchically) structured connectors and, hence, to structure interaction.

On top of the interaction specification, *Priorities* specify a strict partial order relation among interactions. Priorities model simple scheduling policies that allow the BIP engine for selecting and scheduling (possibly multiple) allowed interactions (with respect to active ports). For instance, the relation {{c, c2} $\prec$ {c, c1} $\prec$ {a, cout} $\prec$ {a, cin}} specifies that the interaction {c, c2} has a lower priority than the interaction {c, c1}.

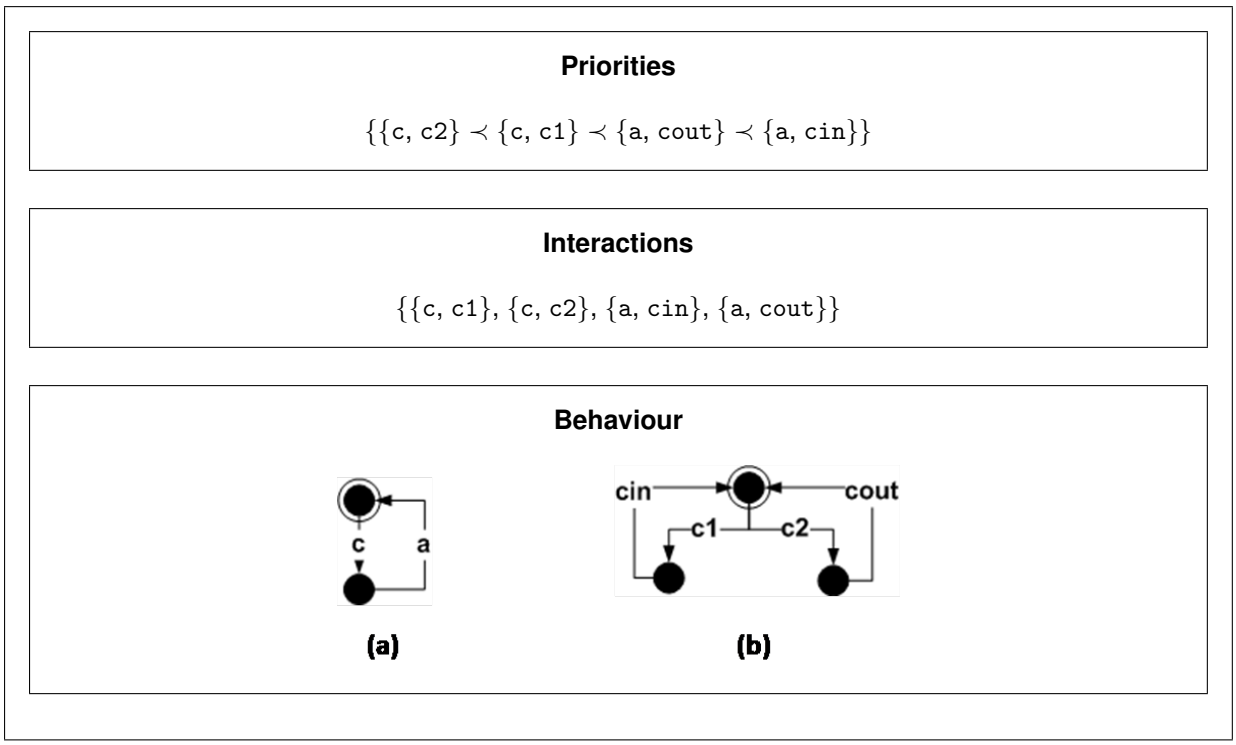

Figure 2.13: BIP: a simple example

BIP uses a powerful composition operator parameterised by the set of allowed interactions and priorities. As formalised in [27], the behaviour of all the atomic components in the system can be composed by using a composition operator similar to the operators used in CSP [59] and CCS [51]. In turn, the BIP composition operator can be used to express both the CSP and CCS composition operators [27].

In BIP, an interaction of the composed system is enabled only if (*i*) it is enabled by the composite automaton resulting from the composition of all the atomic automata, (*ii*) it is in the set of allowed interactions, and (*iii*) it is maximal according to the order imposed by the priority model. Thus, each interaction of the composed system is enabled only if all the ports it involves are active in the current local states of all the corresponding components. Since more than one interaction of the composed system can be enabled at the same time, priorities are used to restrict the deriving non-determinism. For example, the two interactions {c, c1} and {c, c2} are both enabled when the component in Figure 2.12(a) is in the local state where the port c is active, and the component in Figure 2.12(b) is in the local state where both the ports c1 and c2 are active. Considering the priorities above, the BIP engine will select the interaction {c, c1}.

Superposing the three layers described above, Figure 2.13 shows the system composed out of the two components in Figure 2.12 as modelled by the BIP framework.

Within the BIP implementation the automata specifying the components' behaviour are extended with data and C functions (i.e., data transformations), and the BIP engine drives their execution flow in the composed system. Indeed, the BIP compiler can derive the C code that can then be executed on dedicated platform. Each component communicate the set of its active ports to the BIP engine and waits for an interaction. By exploring an enumerative representation of connectors and by considering priorities, the engine selects and computes data transformations associated to maximal interactions (involving active ports), hence notifying the interested components.

2.3.4.1 Overview

In this section, after introducing basic notions and the algebra of interaction, we describe the algebra of connectors proposed in [27]. The algebra of connectors, while formalising the concept of connector in BIP, allows for combining synchronisation by rendezvous and broadcast interactions. Complex coordination schemes can be defined by means of this combination.

As already introduced, in [27] a connector structures interaction by modelling relationships among communication ports of components. A communication port can have either type *trigger*, if it can initiate (asymmetric) interactions, or type *synchron*, if it is passive and can be activated by triggers or involved in (symmetric) interactions synchronising all the ports. Specifically, since the BIP engine allows components to communicate through atomic synchronisation of all the ports involved in a given interaction, an (atomic synchronous) interaction is represented by any non empty subset of these ports. Each subset must contains at least a port that is a trigger or, if all the ports are synchrons, the only possible interaction is the one involving all the ports.

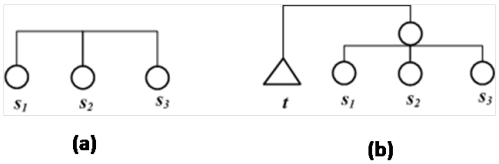

Figure 2.14: A rendezvous and an atomic broadcast connector

As shown in Figure 2.14, connectors can be graphically represented and hierarchically structured. Trigger and synchron ports are represented by empty triangles and circles, respectively, and interactions among them are represented by lines. For instance, Figure 2.14 shows two connectors where s_1, s_2 and s_3 are synchron ports, and t is a trigger port.

The connector (a) graphically models a *rendezvous* interaction that means strong synchronisation among the ports s_1, s_2 and s_3, thus it enables the only synchronous interaction $\{s_1, s_2, s_3\}$. The connector (b) models *atomic broadcast* interactions which means either a rendezvous interaction among all the ports t, s_1, s_2 and s_3 or an interaction involving only the trigger port t. Formally, the connector (b) enables the interactions $\{t\,,\, ts_1s_2s_3\}$, where t and $ts_1s_2s_3$ stand for the sets $\{t\}$ and $\{t, s_1, s_2, s_3\}$, respectively. If we consider a client component C with a trigger port t and three server components S_1, S_2 and S_3 with the synchron ports s_1, s_2 and s_3, respectively, the connector (b) can be used to model interactions where a message sent by C is received either by all the servers component S_1, S_2 and S_3 or by none of them. It is worth noting that the connector (b) has a hierarchical structure since the synchronous interaction $\{s_1, s_2, s_3\}$ has been (hierarchically) typed as synchron (see Figure 2.15). In this way, the three ports $\{s_1, s_2, s_3\}$ can be considered as a monolithic port with respect to C point of view.

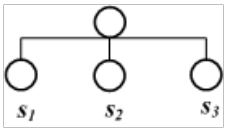

Figure 2.15: Hierarchical typing

The hierarchical structure of the connectors, and hence the hierarchical typing, allows for the *incremental construction* of more complex connectors from simpler ones. Generally speaking, when a

connector is hierarchically typed as a trigger, all the interactions allowed by that connector act as distinct triggers; whereas, when a connector is hierarchically typed as a synchron, all the interactions allowed by that connector can be distinctly synchronised. For instance, by referring to Figure 2.16, the interactions allowed by the connector (a) are $\{t, tr_1\}$; the interactions allowed by the connector (b) are $\{r_2, r_3, r_2r_3\}$. Now, by typing the connector (a) as a trigger and the connector (b) as a synchron, and by composing them together we achieve the composite connector (c) whose allowed interactions are $\{t, tr_1, tr_2, tr_3, tr_2r_3, tr_1r_2, tr_1r_3, tr_1r_2r_3\}$. The latter set of interactions models a broadcast connector that allows all interactions involving the port t and any (possibly empty) subset of the ports $\{r_1, r_2, r_3\}$.

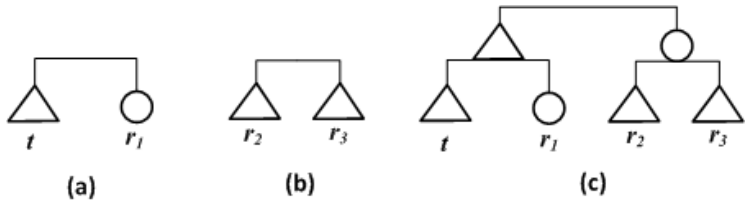

Figure 2.16: Incremental construction of connectors

Note that, the two interactions $\{t, tr_1\}$ allowed by the connector (a) act as triggers and synchronise with all the interactions $\{r_2, r_3, r_2r_3\}$ allowed by the connector (b) (that are then acting as synchrons). In this sense, it can be said that hierarchical typing permits to consider a whole interaction either as a trigger or as a synchron in the same way as a single port. Note also that, after the connector (b) has been hierarchically typed as synchron in the composite connector (c), the interactions allowed by (b) act as synchron, even though they derive from the ports r_2 and r_3 whose type is trigger. In this sense, at least in principle, it can also be said that hierarchical typing "covers" the types of the ports and "assigns" types to whole interactions. These considerations will be clear later, after the connector algebra has been presented.

We now present the algebra of interactions that provides a simple and intuitive syntax to concisely formalise the notion of interaction in BIP. Given a set of ports P, such that $0, 1 \notin P$, the syntax for terms of the *algebra of connectors* $\mathcal{AI}(P)$ is defined as follow:

$$x ::= 0 \,|\, 1 \,|\, p \in P \,|\, x \cdot x \,|\, x + x \,|\, (x)$$

where the *synchronisation* operator "$\cdot$" has a higher precedence than the *union* operator "$+$".

The semantics of $\mathcal{AI}(P)$ is a function $\|\cdot\| : \mathcal{AI}(P) \to 2^{2^P}$ that given a term in $\mathcal{AI}(P)$ returns the set of interactions it represents (i.e., a set of sub-sets of P). $\|\cdot\|$ is defined as follow:

$$\|0\| = \emptyset, \quad \|1\| = \{\emptyset\}, \quad \|p\| = \{\{p\}\},$$
$$\|x_1 + x_2\| = \|x_1\| \cup \|x_2\|,$$
$$\|x_1 \cdot x_2\| = \{a_1 \cup a_2 \mid a_1 \in \|x_1\|, a_2 \in \|x_2\|\},$$
$$(\|x\|) = \|x\|,$$

where $p \in P$ and $x, x_1, x_2 \in \mathcal{AI}(P)$.

The operations of $\mathcal{AI}(P)$ satisfy the following axioms: 0 and 1 are the *identity* elements for the operators "$+$" and "$\cdot$", respectively, and 0 is an *absorbing* element of the operators "$\cdot$". Moreover, both the operators are *idempotent*, *associative* and *commutative*, and synchronisation *distributes* over union. For instance, for the operators to be idempotent means that for any term $t \in \mathcal{AI}(P)$, we have $t * t = t$, with $* \in \{\cdot, +\}$, i.e., nothing changes if an interaction is synchronised or unified with itself. It is easy to prove that the axioms of $\mathcal{AI}(P)$ are *sound* and *complete* with respect to the given semantics, i.e., for $x, y \in \mathcal{AI}(P)$ we have:

$$x = y \Leftrightarrow \|x\| = \|y\|$$

In particular, the completeness proof is achieved by considering terms x, y having the same sets of interactions $\|x\|, \|y\|$, and by showing that, distributing synchronisation over union, x and y are syntactically the same terms.

By using $\mathcal{AI}(P)$ the atomic broadcast interactions $\{t\,,\, ts_1s_2s_3\}$ (allowed by the connector graphically modelled in Figure 2.14(b)) can be succinctly specified by the term:

$$t \cdot (1 + s_1 \cdot s_2 \cdot s_3) \quad or \;\; simply \;\; t(1 + s_1s_2s_3)$$

Even though very concise, this algebraic representation highlights fundamental aspects of the atomic broadcast interactions. We recall the atomic broadcast allows for either a rendezvous interaction among all the ports t, s_1, s_2 and s_3 or an interaction involving only the trigger port t. In fact, the expression $(1+s_1s_2s_3)$ suggests that the strong synchronisation $s_1s_2s_3$ is optional. This implies that the port t can initiate interactions in which the strong synchronisation $s_1s_2s_3$ in triggered or not. The implication becomes explicit if we distribute the synchronisation operator "$\cdot$" over the union operator "$+$", hence obtaining:

$$\begin{array}{l}
\|t \cdot (1 + s_1s_2s_3)\| = \\
\|t + ts_1s_2s_3)\| = \\
\|t\| \cup \|ts_1s_2s_3\| = \\
\{\{t\}\} \cup \{\{ts_1, s_2, s_3\}\} = \\
\{\{t\}\,,\, \{ts_1, s_2, s_3\}\} \quad or \;\; simply \;\; \{t\,,\, ts_1s_2s_3\}
\end{array}$$

As previously said, the algebra of interaction is used as basis to provide an algebraic formalisation of (hierarchical) connectors and, hence, to structure the system interaction. The syntax of the *algebra of connectors* $\mathcal{AC}(P)$ also considers a set of component ports P, such that $0, 1 \notin P$, and it is defined as follows:

$$\begin{array}{rl}
s & ::= [0] \,|\, [1] \,|\, [p] \,|\, [x]\,, \quad (synchrons) \\
t & ::= [0]' \,|\, [1]' \,|\, [p]' \,|\, [x]'\,, \quad (triggers) \\
x & ::= s \,|\, t \,|\, x \cdot x \,|\, x + x \,|\, (x),
\end{array}$$

where the operator "$\cdot$", now called *fusion*, has a higher precedence than the *union* operator "$+$". As it will be clear from the semantics below, the union operation of $\mathcal{AC}(P)$ has the same meaning of the union operation as in $\mathcal{AI}(P)$; whereas, fusion in $\mathcal{AC}(P)$ generalises synchronisation in $\mathcal{AI}(P)$. Square brackets "$[\cdot]$" and "$[\cdot]'$" represent two typing operators for synchrons and triggers, respectively, that allow for defining typed connectors and, hence, for hierarchically structuring connectors.

The semantics of $\mathcal{AC}(P)$ is a function $|\cdot| : \mathcal{AC}(P) \to \mathcal{AI}(P)$ that, given a term $c \in \mathcal{AC}(P)$ (specifying a connector), returns a syntactic term in $\mathcal{AI}(P)$ that corresponds exactly to the interactions allowed by the connector c:

$$\begin{array}{c}
|[p]| = p, \\
|x_1 + x_2| = |x_1| + |x_2|\,, \\
|\prod_{i=1}^{n}[x_i]| = \prod_{i=1}^{n} |x_i|\,, \\
\left|\prod_{i=1}^{n}[x_i]' \cdot \prod_{j=1}^{m}[y_j]\right| = \sum_{i=1}^{n} |x_i| \cdot \left(\prod_{k\neq i} (1 + |x_k|) \cdot \prod_{j=1}^{m} (1 + |y_j|)\right)
\end{array}$$

where $x, x_1, \ldots, x_n, y_1, \ldots, y_m \in \mathcal{AC}(P)$ and $p \in P \cup \{0, 1\}$.

Composing the *interaction semantics* $\|\cdot\|$ of $\mathcal{AI}(P)$ with the *connector semantics* $|\cdot|$ of $\mathcal{AC}(P)$ we obtain the *interaction semantics* of $\mathcal{AC}(P)$. The last rule expresses the fact that each trigger term must participate in all interactions, whereas, synchron terms are optional.

The operations of $\mathcal{AC}(P)$ satisfy, the following axioms: $[0]$ and $[1]$ are the identity elements for the union and fusion operators, respectively. The union operator is *idempotent*, *associative* and *commutative*. The fusion operator is in general *commutative* and *distributive* over union. However, fusion is *associative* only when "$(\cdot)$" are applied, i.e., only when terms are simply grouped; fusion is not in general associative when typing is applied, thus, $[x][y][z]$, $[[x][y]][z]$ and $[x][[y][z]]$ might have different meaning. Moreover, fusion is *idempotent* only on *monomial* connectors, i.e., connectors that involve only the fusion operator "$\cdot$". For the fusion operator to be idempotent only on monomial connectors, means that the allowed interactions might change if a non monomial connector is fused with itself, but nothing changes if we fuse a monomial

connector with itself (i.e., the allowed interactions would be the same). The typing operators "[]" and "[]′" also satisfy other relevant axioms, thus, please refer to [27] for details.

Still referring to Figure 2.14(b), the structure and the interactions allowed by atomic broadcast connector are captured in $\mathcal{AC}(P)$ by the expression $t'[s_1 s_2 s_3]$ (square brackets on 0, 1 and ports $p \in P$, as well as the operator "$\cdot$" are usually omitted). By resolving its semantics we have:

$$
\begin{array}{l}
|\, t'\,[s_1 s_2 s_3]\,| \,= \\
|\, t\,|(1 + |\, s_1 s_2 s_3\,|) \,= \\
|\, t\,|(1 + |\, s_1\,|\,|\, s_2\,|\,|\, s_3\,|) \,= \\
t\,(1 + s_1 s_2 s_3)
\end{array}
$$

The achieved expression in $\mathcal{AI}(P)$ corresponds exactly to set of all possible interactions allowed by atomic broadcast (i.e., $\{t\,,\, ts_1 s_2 s_3\}$ - see above). As already, the typing operator induces a hierarchical structure, and hence (as shown in Figures 2.14(b), 2.15 and 2.16(c)) connectors can be graphically represented as sets of trees. Moreover, each connector can be specified as a union of monomial connectors by distributing the fusion and the typing operator over the union operator. However, as already said, when distributing one has to be careful to distinguish the parentheses "$(\cdot)$" from the typing operators "$[\cdot]$" and "$[\cdot]'$". The parentheses are treated in the same way as for arithmetic basic multiplication and sum operations. The typing operators has to be treated by strictly following the semantic rules of $\mathcal{AC}(P)$. Thus, considering that for a port p, p stands for $[p]$ and p' stands for $[p]'$,

$|a(b' + c)| = |ab' + ac| = |ab'| + |ac| = |b|(1 + |a|) + |ac| = |b| + |a||b| + |a||c| = b + ab + ac$
and
$\|b + ab + ac\| = \|b\| \cup \|ab\| \cup \|ac\| = \{b, ab, ac\}$;
whereas,
$|a[b' + c]| =^{5} |a([b'] + [c])| =^{6} |ab| + |ac| = |a||b| + |a||c| = ab + ac$
and
$\|ab + ac\| = \|ab\| \cup \|ac\| = \{ab, ac\}$.

Now, after the semantics of connectors have been presented, it should be clear why and how connectors can be represented as trees having ports on their leaves. That is, by virtue of the fact that each connector can be modelled as a union of monomial connectors (by distributing fusion and typing over union), each monomial connector can in turn be represented as a tree. This also means that in the presence of distinct connectors, within a composed system, the union operator of $\mathcal{AC}(P)$ is used to unify them, and hence, to unify the interactions allowed by each single connector. In this sense the union operator of $\mathcal{AI}(P)$ is equivalent to the union operator of $\mathcal{AC}(P)$.

In [27] it is proved that the axioms of $\mathcal{AC}(P)$ are *sound* with respect to its semantics, thus for $x, y \in \mathcal{AC}(P)$:

$$x \,=\, y \Rightarrow |x| \,=\, |y|$$

Unfortunately, the axiomatisation of $\mathcal{AC}(P)$ is *not complete* since the following *equivalence relation* is not a congruence. Two connectors c_1 and c_2 are equivalent (denoted $c_1 \simeq c_2$) iff they have the same semantics, i.e., they model the same sets of interactions, thus we have:

$$c_1 \,\simeq\, c_2 \Longleftrightarrow |c_1| \,=\, |c_2|$$

This equivalence is *not a congruence* since it is not preserved by the fusion operator. This fact impacts on the *compositional construction*, i.e., the structural composability and de-composability of interactions by adding and/or removing connectors in such a way that the resulting composed connector is not affected by the order of those operations. That is, there can be two terms x and y having the same sets of interactions $|x| = |y|$ but different syntactic specification, i.e., $x \neq y$. A direct implication is that there can exist contexts where two equivalent terms cannot be in general substituted. For instance, it is enough to consider two

[5] By applying the axiom $[x + y] = [x] + [y]$ on typing operators (the axiom $[x + y]' = [x]' + [y]'$ is also satisfied).
[6] By applying the axiom $[[x]'] = [x]$ on typing operators (the axiom $[[x]]' = [x]'$ is also satisfied).

ports $p, q \in P$ for which we have that $p' \simeq p$ but $p'q$ is not equivalent to pq. This very simple example shows that the equivalence of terms is not preserved by fusion.

However, as previously shown (see Figure 2.16), connectors can be *hierarchically structured* and *incrementally constructed*. As further example, the atomic broadcast connector $t'\,[s_1 s_2 s_3]$ in Figure 2.14(b) can be constructed by firstly considering the connector $s_1 s_2 s_3$, then by typing it as a synchron $[s_1 s_2 s_3]$ and, finally, by connecting the trigger t' for obtaining $t'\,[s_1 s_2 s_3]$.

In [27] the authors define two subalgebras: $\mathcal{AS}(P)$ which involves only synchrons and $\mathcal{AT}(P)$ which involves only triggers. Basically, the authors add to these subalgebras the following axioms that allow for achieving *associativity* also on fusion of typed connectors:

$$[\,[x][y]\,][z] = [x][y][z] = [x]\,[\,[y][z]\,] \qquad (for \mathcal{AS}(P))$$
$$[\,[x]'[y]'\,]'\,[z]' = [x]'[y]'[z]' = [x]'\,[\,[y]'[z]'\,]' \qquad (for \mathcal{AT}(P))$$
$$[x]'\,y = [x]'\,y \,+\, [x]' \qquad (for \mathcal{AT}(P))$$

What is important to know for the purposes of these notes is that the semantic equivalence is a congruence when applied to equivalent terms formed by connectors having all the same type (either sychron or trigger). This restriction allows for achieving a *sound* and *complete* axiomatisation of $\mathcal{AC}(P)$ and, hence, compositionality when constructing composite connectors out of similarly typed connectors.

In [27] the authors also formalise a correspondence between the algebra of interaction and a Boolean algebra, as well as, between the algebra of connectors and a Boolean algebra. These Boolean algebras (even though need to be further investigated) allow (for instance) to efficiently check if a given interaction belongs to a connector or if two connectors are equivalent. For a detailed formalisation of the Boolean representations we entirely refer to [27].

2.3.4.2 Scenario modelling

In this section we model the *Mary Scenario*. First, we specify the components' behaviour, their interactions and priorities in the BIP component framework. Then, we graphically model a structured connector and fully formalise it in the algebra of connectors.

Figure 2.17 shows the three layers of BIP. The bottom layer consists of four atomic components whose ports' behaviour are specified by means of the four labelled transition systems. Each component also specifies its communication ports and, for the sake of clarity, its input/output direction as well.

`Contact Manager 1` models the behaviour of the software component that manages the Mary's contacts on the old device. The input synchron port `CM1s1` serves to select the *output synchronisation mode*, enabling the output synchron port `CM1cout` for sending out the contacts. The synchron input port `CM1s2` serves to select the *input synchronisation mode*, enabling the synchron input port `CM1cin` through which the contacts can be received. Thus, similarly to the previous approaches, `CM1s1` is used to inform the `Contact Manager 1` that some other component in the system wishes to download the contacts from the device through its `CM1cout`; whereas `CM1s2` is used to inform the `Contact Manager 1` that some other component in the system wishes to upload contacts to the device through its `CM1cin`. The behaviour automaton specifies that from the initial state both the input ports `CM1s1` and `CM1s2` are active and `Contact Manager 1` keeps standing by, while another component (e.g., the Synchronisation Program typically provided by the device manufacturer) selects the synchronisation mode. Then, depending on the selected synchronisation mode, either the port `CM1cout` or the port `CM1cin` is active[7]. In other words, `Contact Manager 1` passively waits for the synchronisation mode to be selected and, hence, for the contact transfer to be triggered. It cannot initiate from its own neither a contact download nor a contact upload.

`Contact Manager 2` models the behaviour of the software component that manages the contacts on the new device. Similarly to `Contact Manager 1`, it allows for uploading or downloading contacts to and from the device. The behaviour automaton of `Contact Manager 2` is (modulo-renaming) similar to the one of `Contact Manager 1`. Indeed, differently from the port `CM1s2` of the `Contact Manager 1`, the port `CM2sin` is a trigger. By means of this trigger port, `Contact Manager 2` can independently initiate an interaction for setting up a contact download through its `CM2ain` port[8].

`Translator` is the component capable to translate contacts. The data it takes (gives) as input (output) through its synchron port `Ttrans_c` (`Tres_a`) and the data transformations it applies offers the same service

[7] For our scenario the ports `CM1s2` and `CM1cin` are not used.
[8] The ports `CM2sout` and `CM2aout` are not used in our scenario.

as for the previous approaches. Thus, the associated behaviour automaton specifies that the synchron ports `Ttrans_c` and `Tres_a` are alternatively in an active state.

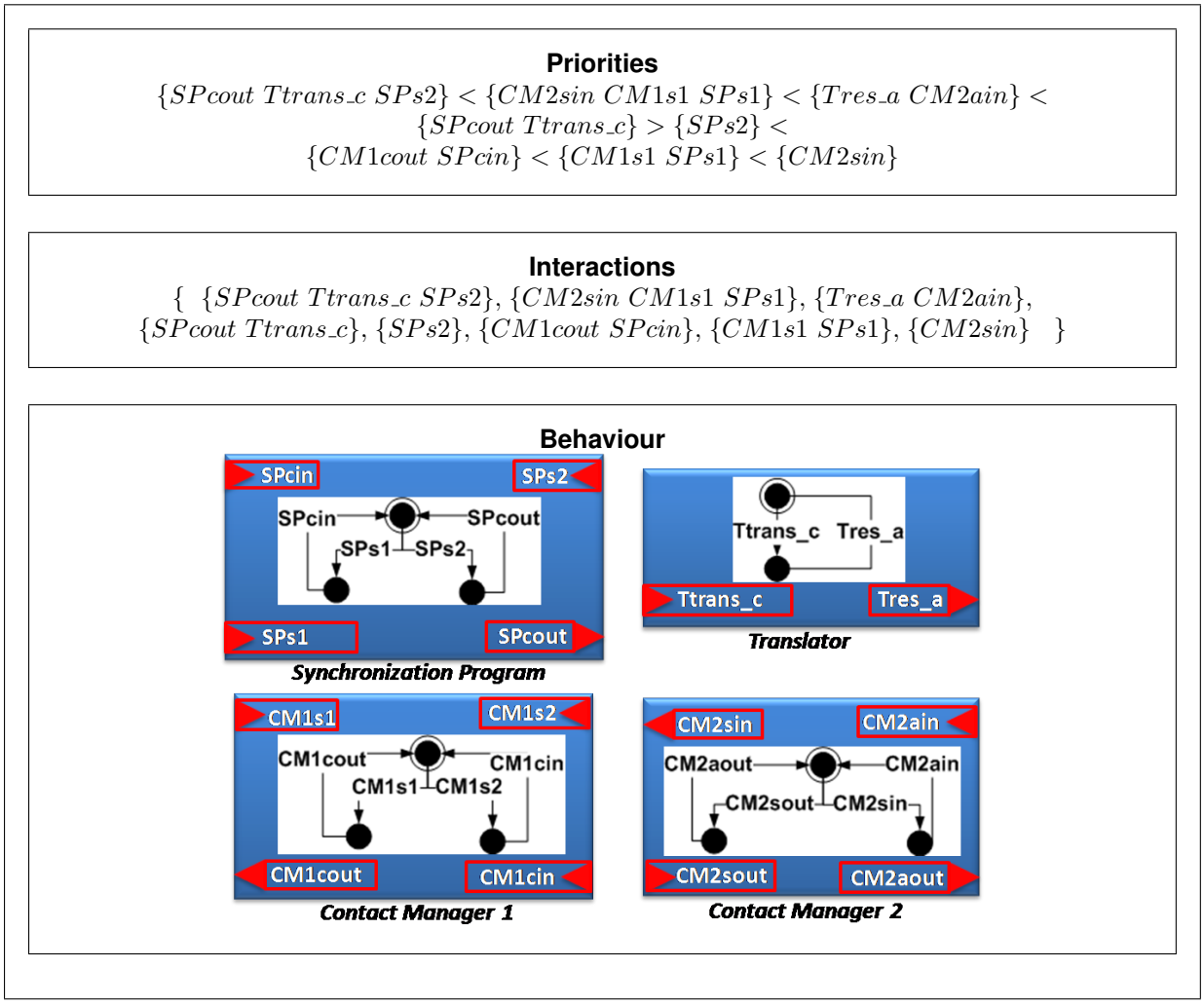

Figure 2.17: The Mary Scenario modelled by the BIP component framework

The `Synchronisation Program` automaton models the behaviour of the component deployed on Mary's laptop able to synchronise contacts with the old device and, hence, with the `Contact Manager 1`. The automaton is (modulo-renaming) analogous to the one of `Contact Manager 1`. In other words, the synchronisation program passively waits for the synchronisation mode to be selected and, hence, for the contact transfer to be triggered. Indeed, the synchronisation program is part of an integrated management suite (for the old device) and, by means of the GUI component, (in usual scenarios) the user chooses to initiate either a contact download through the port `SPs1` or a contact upload through the port `SPs2`. Indeed, in our CONNECT scenario, the contacts' download and upload is not initiated by the user, rather it is initiated by the synthesised connector whose logic has been ad-hoc derived for the *Mary Scenario*.

We recall that in BIP the behaviour automata are extended with data and C functions (i.e., data transformations), and the BIP engine drives their execution flow in the composed system. For our purposes, it is not relevant to specify data and their transformations. For example, the contacts might be all-in-one transferred as a single block (i.e., by a single file) or might be item-by-item streamed. In fact, the graphical representation and the algebra of connectors (that we are going to use to formalise the structure of the interactions of the *Mary scenario*) are concerned with the structure of the connectors (and hence of the interaction) involving communication ports and their types disregarding data and their transformations.

However, within the CONNECT scenario the synthesised connector cannot be a passive means that only transport data between end-points, rather it must be an active party having a specific internal logic. For instance, it should understand that the trigger event from the port `CM2sin` has to be understood as initiator of the synchronous interaction between the `Contact Manager 1` and the `Synchronisation Program` through the ports `CM1s1` and `SPs1`.

In Figure 2.18, we show the graphical representation of the connectors as separated trees.

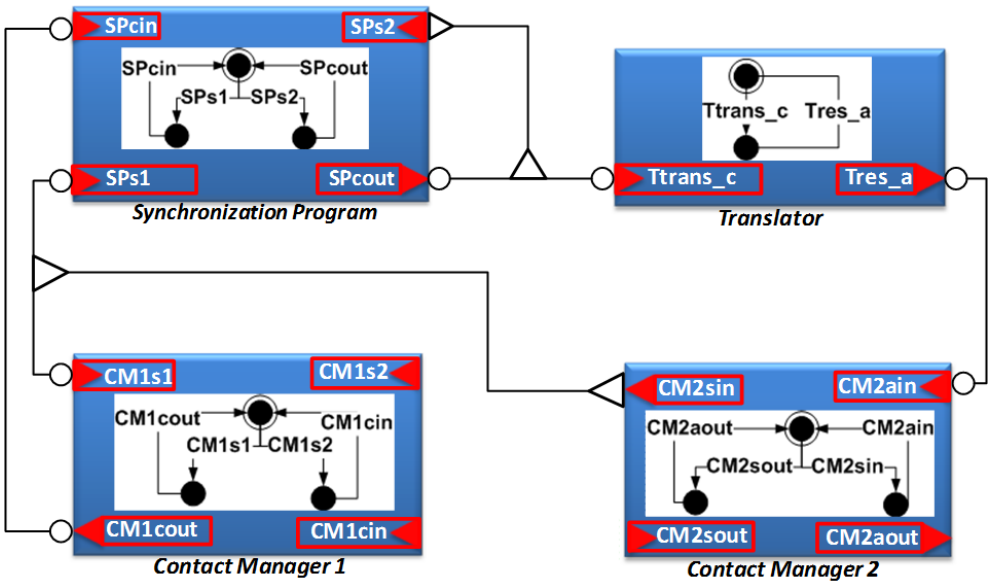

Figure 2.18: Architectural connections of the Mary Scenario in BIP

As an example, let us separately consider (*i*) the connector involving the synchron ports `CM1s1` and `SPs1`, and (*ii*) the single trigger port `CM2sin`. The composite connector involving the three ports `CM1s1`, `SPs1` and `CM2sin` has been incrementally constructed by hierarchically typing the first connector (*i*) as trigger and then by fusing it with the triggers port `CM2sin`. These connectors can be specified by the algebra of connectors as follows:

$$[CM2sin]'[\,[CM1s1][SPs1]\,]' + [CM1cout][SPcin] + [SPs2]'[\,[SPcout][Ttrans\_c]\,]' + [Tres\_a][CM2ain]$$

It is worth noting that we could have specified a single composite connector for the *Mary Scenario* like the one shown in Figure 2.19. For a better understanding of the allowed interactions, the same connector is also shown in Figure 2.20 as a planar tree. Clearly, this composite connector enables much more interactions than the needed ones. However, considering the behaviour automata of the atomic components, most of these interactions are never enabled.

For willing readers in the following we report all the interactions allowed by the composite connectors (the underlined interactions are the only enabled ones):

$\{\underline{Tres\_a\ CM2ain}\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPs2\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1s1\ SPs1\ SPs2\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ SPs2\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1s1\ SPs1\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ SPcout\ Ttrans\_c\}$,

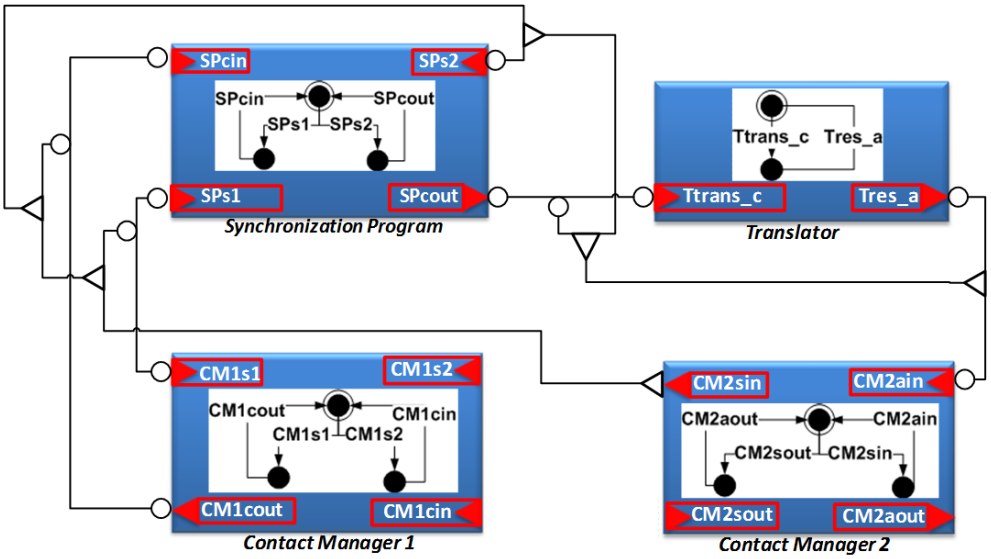

Figure 2.19: Mary Scenario modelled as a single composite BIP-connector

$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPs2\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1s1\ SPs1\ SPs2\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ SPs2\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1cout\ SPcin\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ CM1s1\ SPs1\ Tres\_a\ CM2ain\}$,
$\{CM2sin\ Tres\_a\ CM2ain\}$,
$\{SPs2\ SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{SPcout\ Ttrans\_c\ Tres\_a\ CM2ain\}$,
$\{\underline{SPcout\ Ttrans\_c}\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\}$,
$\{CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\}$,
$\{CM1cout\ SPcin\ SPs2\ SPcout\ Ttrans\_c\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPs2\}$,
$\{CM2sin\ CM1s1\ SPs1\ SPs2\}$,
$\{CM2sin\ SPs2\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPcout\ Ttrans\_c\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPcout\ Ttrans\_c\}$,
$\{CM2sin\ CM1s1\ SPs1\ SPcout\ Ttrans\_c\}$,
$\{CM2sin\ SPcout\ Ttrans\_c\}$,
$\{SPs2\ SPcout\ Ttrans\_c\}$
$\{\underline{SPs2}\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\}$,
$\{CM1s1\ SPs1\ CM1cout\ SPcin\ SPs2\}$,
$\{CM1cout\ SPcin\ SPs2\}$,
$\{CM2sin\ CM1cout\ SPcin\ SPs2\}$, $\{CM2sin\ CM1s1\ SPs1\ SPs2\}$, $\{CM2sin\ SPs2\}$
$\{\underline{CM1cout\ SPcin}\}$,
$\{CM2sin\ CM1s1\ SPs1\ CM1cout\ SPcin\}$,
$\{CM1s1\ SPs1\ CM1cout\ SPcin\}$,
$\{CM2sin\ CM1cout\ SPcin\}$
$\{\underline{CM1s1\ SPs1}\}$,
$\{\underline{CM2sin}\}$
$\{CM2sin\ CM1s1\ SPs1\}$

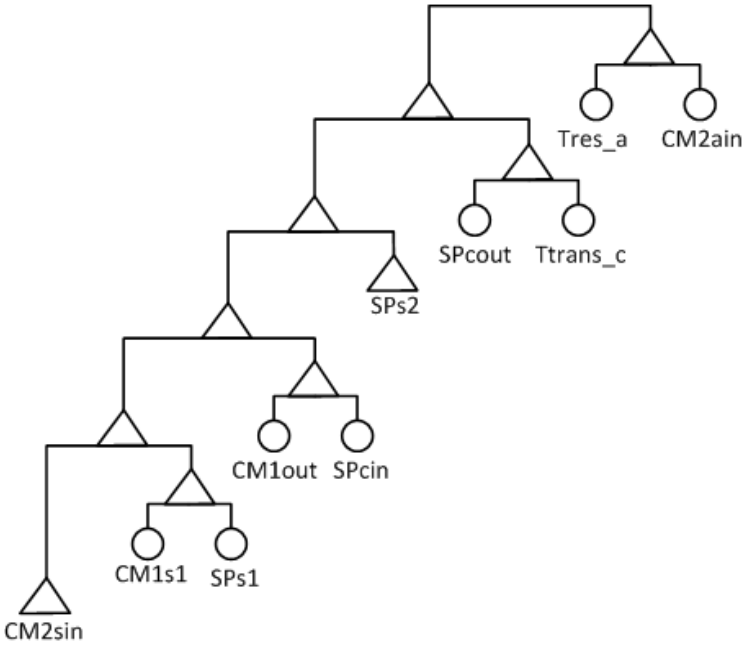

Figure 2.20: A planar view of the single composite BIP-connector

2.3.4.3 Advantages and disadvantages

A number of pros and cons for the approach in [27] can be summarised as follows.

From the theoretical/foundational side the approach is sound and complete. It well formalises concepts and control-flow mechanisms that are implemented by the BIP framework but which, to some extent, can be shared by other approaches. Connectors can express complex coordination schemes combining synchronisation by rendezvous and broadcast. Connectors can be hierarchically structured and represented by trees. The Boolean algebra representation allows for using efficient decision techniques to decide whether or not two connectors are equivalent, and to decide which connector(s) an interaction belongs to.

By formalising mechanisms and concepts of BIP, the algebra is suitable for improving the performance of the BIP execution engine when exploring the set of interactions. In fact, as discussed in [27], the symbolic models and the "mathematical reasoning" allowed by the algebra of connectors can be used to (possibly) reduce the overhead of the BIP engine while exploring the enumerative representations of connectors. In this sense, the approach is **scalable** when considering an increasing number of components and connectors.

However, even though the approach allows for hierarchically structuring and **incrementally** constructing connectors, **composability** is in general achieved only on similarly typed connectors.

BIP connectors are **reusable** w.r.t the set of components they connect. More precisely, connectors are in general unaware of the components to which they connect. In fact, connectors structure the component interactions in the sense that, independently from the actual data transformations, they allow only a certain set of interactions. Then, depending on the component behaviours and interaction priorities, an interaction can be scheduled or not. Similarly to Reo connectors, this means that the same connector can be reused for connecting different sets of components to give them the same structure of the possible interactions.

From the **tool support** side, the BIP framework is equipped with a toolkit (BIP toolkit). It generates an executable file for each BIP model. This executable file simulates the model with respect to the BIP semantics. Generally the model will run *ad infinitum* unless a deadlock is encountered. The executable may also be run in an interactive mode, whereby whenever multiple interactions are available, the user is prompted to resolve the choice, rather than taking into account the priorities. This mode also allows the user to query the values of variables defined in the model.

Evolution and **non-functional** specifications are not considered by the approach in [27]. **Compositional reasoning** is not explicitly discussed by the authors but compositional reasoning techniques might

be investigated. In fact, at least in principle, these techniques might be applied to the sub-set of similarly typed connectors for which compositionality holds.

2.3.5 Bigraphical Reactive Systems

Up until now we have considered formalisms that treat connectors as first-class entities. Each of the formalisms have well-defined semantics that map intuitively onto connector behaviours. Bigraphs, on the other hand, do not come accompanied with semantics; in that respect they are purely notational devices. As a result it is difficult to provide an analysis of the key dimensions for connectors in this formalism, because we need a mapping from our notion of connector onto bigraphs. Based on our analysis we are of the opinion that bigraphs are expressive enough to make the definition of such a mapping possible. However, in CONNECT, we do not yet have a sufficiently clear notion of a connector to attempt the definition of such a mapping at present; this will evolve over the course of the project. Consequently, we provide a high-level analysis of bigraphs, but make ourselves aware that this dissection may not necessarily reflect onto connectors.

Bigraphs as a mathematical concept are not quite as old as graph theory itself, but are certainly well-established. Recent developments by Robin Milner over the last decade have shown just how useful these objects can be in providing mathematical structure to computation. Milner proposes an extension of bigraphs to Bigraphical Reactive Systems (BRSs) [53], which consist loosely of bigraphs and a series of bigraphical rewrite rules for manipulating bigraphs. It has been successfully shown that BRSs can model CCS, π-calculus, mobile ambients, λ-calculus and numerous other process algebras. Considering the modelling of λ-calculus in conjunction with the Curry-Howard isomorphism suggests that BRSs have an elegant categorical interpretation, and indeed they do. Thus we can apply many of the generic results underpinning abstract algebra to BRSs, which allow us to reason about BRSs in a formal and systematic way. The main open question, then, concerns the fact of whether we can model connectors as BRSs. Given the aforementioned non-exhaustive enumeration of calculi that have been modelled as BRSs, the modelling of connectors seems a trivial task, and should certainly be possible. The only difficulty is ensuring that we encode connectors in the 'correct' way, so that we obtain maximum power from the underlying bigraphical framework.

In keeping with the established tradition of the preceding sections, we now present the WHAT-WHY-HOW description for BRSs:

- **WHAT -** A graphical representation for a formalism (examples include process calculi and algebras), together with a set of rewrite rules for transforming the state of the formalism (potentially corresponds to system evolution).
- **WHY -** To reason in a syntax-independent way about the behaviours of formalisms. Using bigraphs as a common intermediary notation allows us to compare the expressivity of different formalisms in a pointwise manner.
- **HOW -** By interpreting the behaviour of modelling formalisms through the use of graphs and graph transformations.

In the following section we begin to define bigraphs and BRSs, and see a few examples of how they might be used.

2.3.5.1 Overview

Bare bigraphs

A *bare bigraph* G' consists of a collection of (possibly nested) nodes, each of which has a number of ports, together with a link structure between ports on nodes, where each link may be connected to an arbitrary number of ports. A bare bigraph may be decomposed into two independent structures that represent both the nesting of nodes, which we refer to as the *forest*, as well as the linking of nodes, which we refer to as the *hypergraph*. These structures are shown in Figure 2.21.

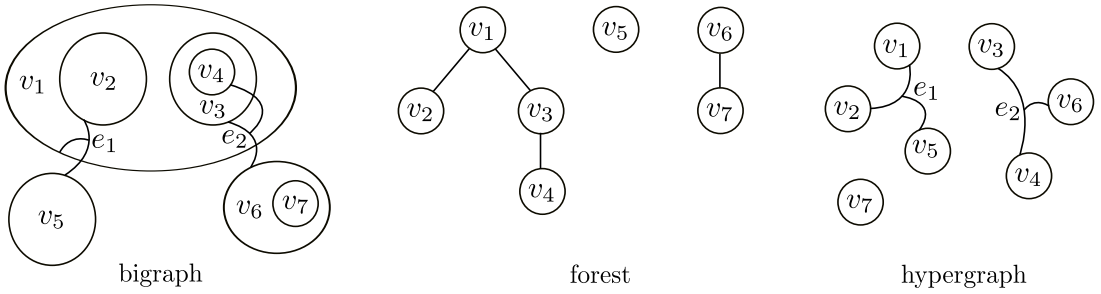

Figure 2.21: Bare bigraph together with the forest and hypergraph

Bigraphs & interfaces

We can extend a bare bigraph G' to a full bigraph G by the assignment of both an outer and an inner *interface*. These define signatures that allow us to combine bigraphs together, as we will see in a moment. Formally, an interface for a bigraph G is a pair (m, X) defined as follows:

Define $\mathcal{X}$ to be a global countably infinite labelling set. If (m, X) is an outer interface, then $m \in \mathbb{N}$ is interpreted as a set $\{0, 1, \ldots, m-1\}$, each element of which (except for possibly one) indexes the roots of the forest of G'. The distinguished element, if it exists, is used to represent the general space outside of the root nodes. $X \subseteq \mathcal{X}$ represents the set of edges in G' labelled by members of X that are visible through the outer interface.

Similarly, if (m, X) is an inner interface, then $X \subseteq \mathcal{X}$ represents the set of edges in G' labelled by members of X that are visible through the inner interface. Again, $m \in \mathbb{N}$ is interpreted as a set $\{0, 1, \ldots, m-1\}$, but in this case each element (except for possibly one) indexes the sites of G', i.e. a node within which we can choose to add more nodes. Yet again, the distinguished element, if it exists, refers to the space outside all nodes.

A bigraph G, written as $G : (m, X) \to (n, Y)$, consists of the forest and hypergraph of G' together with the inner and outer interfaces (m, X) and (n, Y) respectively. The arrow notation is correctly suggestive of categories, as the bigraph may be seen as a morphism in a precategory (objects correspond to interfaces).

Composition of bigraphs

Given two bigraphs $G : (l, X) \to (m, Y)$ and $H : (m, Y) \to (n, Z)$, their composition is a bigraph $H \circ G : (l, X) \to (n, Z)$, in which for each $i \in m$ root i of G is positioned in space i of H, and for each $x \in X$, the edge labelled by x in G is linked to the edge x in H.

Thus the bigraph $H \circ G$ has a forest obtained by taking the forest of H and grafting each root of G onto the corresponding space in H. The hypergraph is obtained by taking the union of the links in G and H, and connecting those links that are labelled from the vocabulary of Y.

In the case that the bigraph $H \circ G$ is defined, we say that H is a contextual or enclosing bigraph of G. This is equivalent to saying that G is a sub-bigraph of H.

Restrictions on bigraphs

In bigraphical modelling, it is common to want to ascribe information and meaning to certain nodes. Perhaps a node represents a term in the λ-calculus, in which case we would want to have nodes corresponding to variables, λ-abstractions and applications etc. Each of these node types should only allow a set number of links to be connected to them. Thus we define the notion of a signature.

A *signature* for a bigraph G in a pair $(\mathcal{K}, ar)$, where $\mathcal{K}$ is a finite set of node-types and $ar : \mathcal{K} \to \mathbb{N}$ defines the arity (or number of ports) on each node-type. Each node in G has a type k drawn from $\mathcal{K}$, and possesses exactly $ar(k)$ ports.

Bigraphical Reactive Systems

We now present a very informal definition of a Simple BRS. A Simple BRS consists of a bigraph G, together with a set of reaction rules $\mathcal{R}$. Each rule $R \in \mathcal{R}$ is a pair $R = (R_1, R_2)$, where $R_i : (m, X) \to (n, Y)$

are bigraphs. We say that R_1 is the *redex* and R_2 is the *reactum*. In a very waffly sense, if a redex R matches a portion of the bigraph G, we can pull R out and reconnect R' in its place, providing $(R, R') \in \mathcal{R}$. This new bigraph replaces the bigraph G in the BRS. The bigraph in a BRS evolves over time by the repeated application of reaction rules.

2.3.5.2 Scenario modelling

We now provide a model of the *Mary Scenario* in terms of a BRS. We begin by defining a signature for the components in the environment. This will simply be:

$$(\{phone, exchanger, contact\}, \{phone \mapsto 1, exchanger \mapsto 2, contact \mapsto 0\}). \tag{2.1}$$

The initial configuration of the environment is denoted by the following bigraph, which will be the initial bigraph in the BRS. The phones are initially disconnected from the exchanger, and $phone_1$ contains 3 contacts, whilst $phone_2$ contains a single contact.

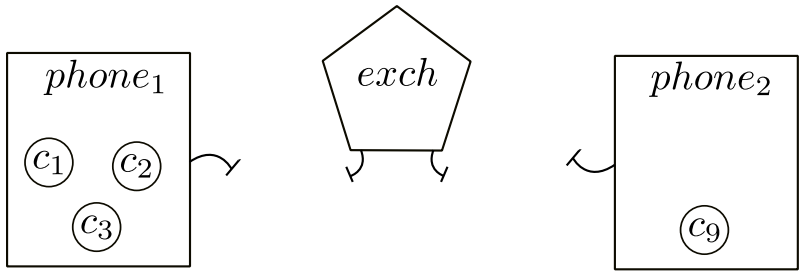

Figure 2.22: Initial bigraph for the Mary Scenario

We now need to define the reaction rules for transferring contacts from one phone to the other. These are shown in Figure 2.23. The rules adopt a number of conventions, as we explain:

1. Unshaded dashed boxes are used to indicate that there could be other entities in the environment that have the same locality as objects inside the box. Thus these boxes can be thought of as part of the outer interface of the bigraph in the redex/reactum. The state of objects located outside the dashed box remains untouched by the reaction, modulo changes described in point 4.

2. Shaded dashed boxes are used to indicate that there may be additional objects contained within the enclosing node. The contents of these boxes remain unchanged, except for changes explained in point 4. These boxes can be thought of as part of the inner interface of the redex/reactum bigraph.

3. A link that extends outside of the unshaded dashed box is *possibly* connected to a number of nodes outside of the redex/reactum bigraph, i.e., a node outside the unshaded dashed box, or a node inside a shaded dashed box. It is possible that the edge is not connected to anything outside, however.

4. It is possible to alter the links outside of the redex/reactum bigraph by connecting or disconnecting a link inside the bigraph that extends outside as described in point 3.

We now give an account of the reaction rules. The first reaction rule R_1 allows *either* phone to be connected to the left-most port of the exchanger, providing the phone is not already connected to anything. The phone may contain nodes, i.e. an arbitrary number of contacts, as may the exchanger. R_2 allows any disconnected phone to connect to the right-most port of the exchanger. R_3 and R_4 allow for the transfer of contacts between a phone and the exchanger, and vice versa, one at a time. R_5 and R_6 allow for the disconnection of phones from the ports of the exchanger.

There are many strategies for transferring the contacts from $phone_1$ to $phone_2$. One such strategy proceeds as follows:

$$G \xrightarrow{R_1} G_1 \xrightarrow{R_2} G_2 \xrightarrow{R_3} G_3 \xrightarrow{R_3} G_4 \xrightarrow{R_3} G_5 \xrightarrow{R_4} G_6 \xrightarrow{R_4} G_7 \xrightarrow{R_4} G_8 \xrightarrow{R_5} G_9 \xrightarrow{R_6} G' \tag{2.2}$$

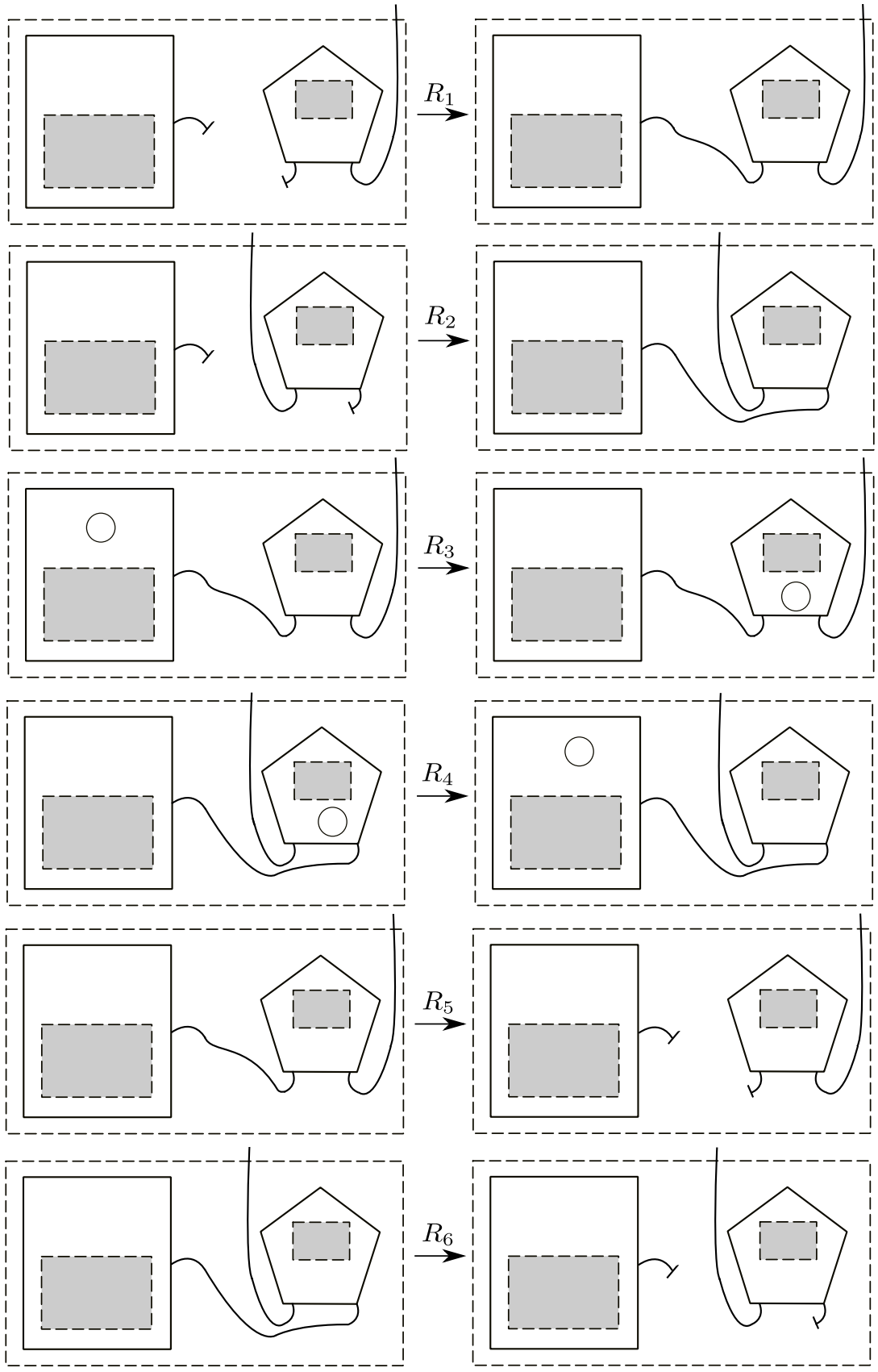

Figure 2.23: Reaction rules for the Mary Scenario

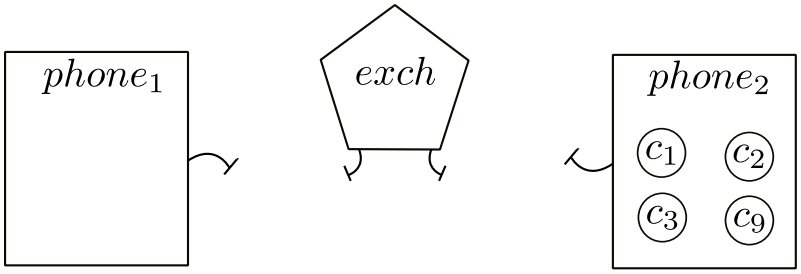

Figure 2.24: The bigraph G'

Under R_1, the $phone$ node in the redex is mapped onto $phone_1$ in G. The $phone$ node in R_2 is mapped onto $phone_2$. The applications of R_3 map onto any permutation of c_1, c_2 or c_3, and similarly for R_4. R_5 and R_6 are completely determined. The bigraph G' is identical to that of G, with the exception that c_1, c_2 and c_3 now reside on $phone_2$, along with c_9. This is shown in Figure 2.24.

2.3.5.3 Advantages and disadvantages

Before summarising the key features of BRSs, and highlighting their advantages and disadvantages, it is instructive to pose the following question to ourselves, "In what sense have we really modelled a connector for the *Mary Scenario*?"

This in turn raises the question of defining a connector in a BRS. From the perspective of the *Mary Scenario*, a connector corresponds loosely to a strategy in the BRS. The bigraph represents the state of the system under consideration, which we hope the Connect Enabler will be able to learn, or at least an abstraction of it. The reaction rules correspond to capabilities of the entities in the system; methods that we hope our Connect Enabler can interact with and instantiate. This then prescribes to us exactly what the Connect Enabler must do. It should traverse the execution tree of the BRS to find a bigraph that satisfies our goal conditions i.e. one where the contacts have been moved from one phone to the other. The sequence of reactions that get us to the goal bigraph is then a connector that achieves our tasks. Of course, we will need to perform systematic learning and discovery along the way, but bigraphs give us a high-level overview of what we should do (or at least one way of doing this).

We now return to our summarisation of BRSs. We saw early on that bigraphs are **compositional**, and so there is a natural way in which BRSs may be combined: combine the bigraphs and take the union of the reaction rules. As a result, we are also guaranteed **incrementality** of BRSs, because we can extend them adhoc by throwing in more nodes and links, as well as by adding more reaction rules.

The BRS notation is also **scalable**. This follows from the fact that we can build BRSs of any size, and that we can also change our level of abstraction.

It is ambiguous to say whether BRSs support **compositional reasoning**, as this is dependent on the property to be inferred. It certainly seems plausible that properties that can be inferred component-wise can also be interpreted in this notation using similar techniques.

As we remarked earlier, a connector is a series of instructions, therefore connectors in this notation are **reusable**. We can simply re-execute the sequential coordination actions. The BRS notation also supports **evolvability**, as we can, in theory, generate the strategy on the fly responding to any unexpected changes to the system as we try to achieve our goals. In BRSs, components are generally modelled as nodes that have a locality. Thus migration of components can be captured effortlessly in this notation.

There is a stochastic extension of BRSs that can be used for capturing non-functional properties. This extension allows for reasoning about **non-functional properties** by using quantitative model-checking techniques.

Finally, the IT University of Copenhagen has been working on developing a Bigraphical Programming Language [5], thus **toolkit support** is well on its way.

2.4 Quantitative connector algebras

Following on from our analysis of connector formalisms in Section 2.3, we now present a number of formalisms that can be used to capture quantitative behaviours and properties. It so happens that each of these formalisms is an extension of the exogenous coordination language Reo, as described in Section 2.3.2. For each approach we provide a high-level overview of the features of the formalism, together with the principal mathematical underpinnings. Where practicable we bestow a model of the *Mary Scenario*, and relay the main issues relating to this construction. Finally, for each formalism we conclude by summarising the main advantages and disadvantages with respect to the dimensions mentioned in Section 2.1.

2.4.1 Reo connectors with QoS guarantees

As discussed in Section 2.3.2, Reo connectors support the development and analysis of large scale distributed applications by allowing construction of complex component connectors out of simpler ones. Modelling, analysing, and ensuring end-to-end Quality of Service (QoS) represent key concerns in such large scale distributed applications.

In this section we introduce a compositional model of QoS based on *Quantitative Constraint Automata* (QCAs), which were introduced by Arbab et al in [13]. These models map intuitively onto Quantitative Reo, an extension of Reo in which channels are annotated with QoS values. Without digression, we provide the *WHAT-WHY-HOW* summarisation below.

- **WHAT -** To define a compositional and operational model for reasoning about general QoS properties of the exogenous channel-based coordination defined by Reo connectors.
- **WHY -** The ABT semantic model for Reo connectors, defined in Arbab's original Reo journal article [10], does not take into account QoS attributes of the component interactions, thus preventing non-functional analysis of them.
- **HOW -** Formalising the notion of QCAs as an automata-based formalism able to capture both the operational semantics of Reo connectors and their QoS attributes, whilst preserving their coalgebraic semantics in terms of timed data streams [10].

2.4.1.1 Overview

QCAs originally stem from Quantitative Reo, so it is highly motivational to begin by defining this formalism, even though it is possible to work directly with QCAs in an independent manner. The development of Quantitative Reo was highly influenced by the desire of the authors of [13] to coordinate component interaction by taking into account the cost of coordination.

Quantitative Reo

Quantitative Reo consists of the same set of primitive channels as defined in basic Reo, except that we can annotate channels with QoS values. These QoS values are typically known in advance; they can be found by measurement, or calculated statistically. Possible examples of QoS values relating to a channel are the bandwidth b in Mbps, and the associated transmission delay d in milliseconds, to name but a few.

In an identical way to basic Reo, networks of Quantitative Reo channels may be built up by connecting them via nodes. We do not impose QoS values on nodes, so do not distinguish nodes in a Quantitative Reo network from those in a basic Reo network. Figure 2.25 shows a collection of the Quantitative Reo channels together with their QoS values.

Let the d_i in Figure 2.25 represent the delay in milliseconds for data to be transmitted through the channel. The synchronous drain channel has delay d_1 in losing the data waiting to be written, when both ends are willing to synchronise. As for the FIFO channel, this has delay d_2 to write data to the buffer, and delay d_3 to read from the buffer. Regarding the lossy channel, this has delay d_4 in transferring data from source to sink when both ends are willing to synchronise, and a delay of d_5 to lose data waiting to be written when the sink is not willing to synchronise.

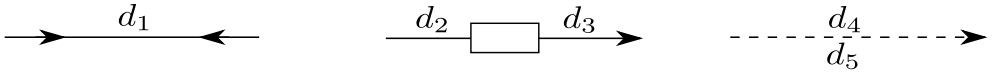

Figure 2.25: Quantitative Reo channels

Q-algebras

A Q-algebra is an algebraic structure $R = (C, \oplus, \otimes, \odot, 0, 1)$, inducing two semirings[9] $R_{\otimes} = (C, \oplus, \otimes, 0, 1)$ and $R_{\odot} = (C, \oplus, \odot, 0, 1)$, where:

- C is the domain of possible QoS values.
- $\oplus$ is a binary relation on C that for any pair of QoS values a and b selects the optimal value in the pair e.g. for bandwidth this would correlate with the smallest value of a and b. Of course, the term 'smallest' corresponds to the metric under consideration.
- $\otimes$ is a binary relation on C that combines QoS values sequentially.
- $\odot$ is a binary relation on C that combines QoS values concurrently.
- 0 corresponds to the identity element of $\oplus$.
- 1 corresponds to the identity elements of $\otimes$ and $\odot$.

A little thought reveals that we can take products of Q-algebras in the obvious way, thus allowing us to handle multiple QoS properties with a single Q-algebra. A QoS value in the product will thus be a tuple of QoS values from the constituent Q-algebras. Further details are available in [26, 56].

Examples. The following QoS metrics and associated Q-algebras are reproduced courtesy of [13]:

- Shortest time for transmission: $(\mathbb{R}^{\infty}_{+}, \min, +, \max, \infty, 0)$.
- Bandwidth for transmission: $(\mathbb{N}^{\infty}, \min, \max, +, \infty, 0)$.
- Reliability (probability of successful transmission): $([0, 1], \max, *, *, 0, 1)$.

Quantitative Constraint Automata

Quantitative Reo circuits may be given semantics in terms of Quantitative Constraint Automata (QCAs), which are extensions of Constraint Automata (CAs). It is sufficient to know, although we do not provide details in this report, that basic Reo networks can be provided semantics in terms of CAs instead of the timed data streams mentioned in Section 2.3.2. Results to this effect are provided by Baier et al in [20]. In that respect, a QCA extends a CA by introducing a Q-algebra, and annotating the transitions in the CA with QoS values from the Quantitative Reo channels.

Definition. A *quantitative constraint automaton (QCA)* $\mathcal{A}$ is a tuple $(S, S_0, \mathcal{N}, R, \rightarrow)$ where:

- As for CAs, S is a set of states, $S_0 \subseteq S$ is the set of initial states, and $\mathcal{N}$ is a finite set of nodes.
- $R = (C, \oplus, \otimes, \odot, 0, 1)$ is a Q-algebra.
- $\rightarrow \subseteq S \times \{N\} \times DC(N) \times C \times S$ is a finite set called the transition relation, where $N \in 2^{\mathcal{N}}$.

[9] A semiring is a ring without the additive inverse axiom. For a definition of a ring, consult any standard text on abstract algebra; one suggestion is [69].

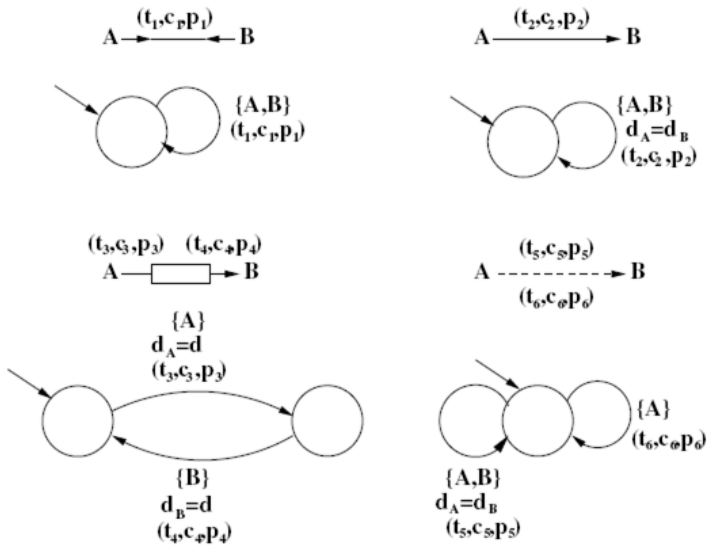

Figure 2.26: Quantitative Constraint Automata for Quantitative Reo channels

An arbitrary transition $s \xrightarrow{N,\delta,c} s'$ is interpreted as follows. Starting in state s, the automaton can move to state s' with cost c providing nodes in N are ready to synchronise and data constraint δ is satisfied. Definitions of node synchronisation and data constraint satisfaction ensue.

Recall that nodes are used to join channels together. According to the semantics of nodes, a mixed node is willing to synchronise providing all source channel ends incident on the node are willing to accept data, and there is at least one sink able to supply data. Synchronisation of source and sink nodes follows from mixed node synchronisation by dropping the condition on the availability of data at the sink and source channel ends respectively.

Data constraints on a set of nodes N are used to specify conditions on the data seen at those nodes. For a fixed set $Data$ of data values, $DC(N)$ is the set of data constraints δ, where

$$\delta \ ::= \ true \ \mid \ d_A = d \ \mid \ \delta \vee \delta \ \mid \ \neg\delta$$

where $A \in N$ and $d \in Data$. Thus $d_A = d$ corresponds to seeing datum d at node A.

Figure 2.26 shows the QCA for some of the Quantitative Reo channels.

Composition of QCAs

The join of two Quantitative Reo channels can be realised through the product automaton of the QCAs. The product QCA is defined in a way analogous to the product CA with the additional requirement that the two QCAs must share the same Q-algebra. If the QCAs do not share the same Q-algebra this is no problem, as there is a sequence of simple transformations that can be used to get the Q-algebras equal; see [31] for details.

Definition. The product of QCAs $\mathcal{A} = (S_A, S_{A,0}, \mathcal{N}_A, R, \rightarrow_A)$ and $\mathcal{B} = (S_B, S_{B,0}, \mathcal{N}_B, R, \rightarrow_B)$ with common Q-algebra R is a QCA

$$\mathcal{A} \bowtie \mathcal{B} \triangleq (S_A \times S_B, S_{A,0} \times S_{B,0}, \mathcal{N}_A \cup \mathcal{N}_B, R, \rightarrow).$$

The set of transitions of $\mathcal{A} \bowtie \mathcal{B}$, written $\rightarrow$, is defined to be the least set satisfying the following conditions:

- For transitions $s_A \xrightarrow{N_A,\delta_A,c_A}_A s'_A$ and $s_B \xrightarrow{N_B,\delta_B,c_B}_B s'_B$ that agree to synchronise on common nodes (this corresponds to the condition $N_A \cap \mathcal{N}_B = N_B \cap \mathcal{N}_A \neq \emptyset$), we should allow these to fire together.

As a result, we add the transition $\langle s_A, s_B\rangle \xrightarrow{N_A \cup N_B, \delta_A \wedge \delta_B, c_A \odot c_B} \langle s'_A, s'_B\rangle$, providing $\delta_A \wedge \delta_B$ is satisfiable.

- A transition $s_A \xrightarrow{N_A, \delta_A, c_A}_A s'_A$ whose set of nodes to synchronise does not occur in $\mathcal{N}_B$ may proceed independently. Thus we add the transition $\langle s_A, s_B\rangle \xrightarrow{N_A, \delta_A, c_A}_A \langle s'_A, s_B\rangle$.
- The final case is symmetric to the previous, in which we allow an independent transition of $\mathcal{B}$ to fire.

It is worthwhile noticing that the product on QCAs is associative, hence leading to a compositional operational semantic model for Quantitative Reo channels.

Further properties

- There is a natural way to transform a CA into a QCA for a given Q-algebra R – simply annotate all transitions on the CA with the identity object 1 in R. In fact, this property is used for transforming two QCAs so that they have a common Q-algebra.
- In keeping with CAs, there is a hiding operation defined on QCAs used to abstract away the details of internal nodes, whose behaviour is unobservable. Full details are provided in [13], so we choose not to reproduce them here. A point of consideration is that although the internal behaviour is hidden, the QoS effect should still be observable by the environment, so we need to take this into account.
- The notion of a run on a QCA is analogous to the notion of a run on a CA. The only relevant aspect to mention, here, is how to compute the cost of a run given the costs of its single transitions. For a run $r = s_0 \xrightarrow{N_0, \delta_0, c_0} s_1 \xrightarrow{N_1, \delta_1, c_1} s_2 \xrightarrow{N_2, \delta_2, c_2} \ldots$, the cost is simply equal to $c_0 \otimes c_1 \otimes c_2 \otimes \ldots$. Note that a run can contain hidden τ transitions of the form $s \xrightarrow{\tau, c} s'$, but this has no effect on the method for computing the cost of the run. Moreover, we can select amongst 'best' runs by employing the $\oplus$ operator on the cost of the runs. As ever, the formal details are contained in [13].
- Another interesting aspect of the QCA model defined in [13], that we omit since it is not crucial for the purposes of this survey, is the notion of *quality improving simulation*. This not only guarantees the inclusion of languages induced by Reo circuits, but allows for better (or at least equal) quality of service. For example, we may ask a connector implementation to be always more reliable, or faster than what is required by the specification, where both the specification and the implementation are given as QCAs.

Summing up, QCAs offers an operational model for reasoning about component connectors with QoS guarantees, together with notions of simulation that are preserved by the QCA product. The results discussed in [13] provide the basis for analysis of both functional and non-functional aspects of Reo component connectors.

2.4.1.2 Scenario modelling

To get an idea of how working with QCAs and Quantitative Reo feels, we model a portion of the *Mary Scenario* described in Section 2.3.2.2. Specifically, we concentrate on the ordering circuit, which is reproduced in Figure 2.27. Note that this is identical to the ABT ordering circuit provided in Section 2.3.2.2, except that we have now annotated the network with QoS values. The QCAs for the individual channels shown in the circuit are exhibited in Figure 2.26. Thus the QCA shown in Figure 2.27 is the result of the composition of these constituent QCAs.

The QoS metrics under consideration in the ordering circuit are detailed below

- t: *shortest time* for data transmission.
- c: allocated *memory cost* for the message transmission.
- p: *reliability* represented by the probability of successful transmission.

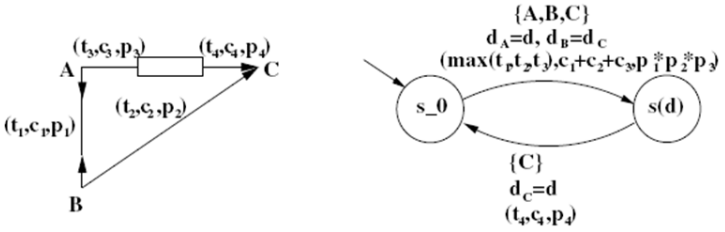

Figure 2.27: A Quantitative Reo circuit and its QCA

The Q-algebras for t and p are defined in the Overview section, whereas the Q-algebra for c is simply $(\mathbb{N}^{\infty}, \min, +, +, \infty, 0)$. The operational semantics of the circuit, as remarked in Section 2.3.2.2, can be seen as imposing an order on the flow of data items written to A and B, through C. s_0 stands for the initial configuration where the buffer is empty, while $s(d)$ represents the configuration where the buffer contains a data element d. If node C is ready for I/O operations in location $s(d)$ then C can take an element d from the buffer and this corresponds to the transition labelled with the set $\{C\}$, data constraint $d_C = d$, and the QoS values $t4, c4, p4$ for the shortest time, memory cost, and reliability values of the transition respectively. From the initial location s_0 we can input data from nodes A and B simultaneously. The data input at A will be stored in the buffer, whilst the data input at B will be directly taken by C if the node C is ready to accept it. The related QoS values are given as in the figure where t_i, c_i, p_i represents the related QoS values for the basic channels.

Non-functional requirement

In Figure 2.28 we show instances of QoS values for the basic channels constituting the ordering circuit. We consider the case in which there are two service providers, **provider1** and **provider2**, each offering different QoS parameters for the circuit's basic channels.

	provider1	**provider2**
SyncDrain	$(t_1 = 0, c_1 = 3, p_1 = 1)$	$(t_1 = 0.1, c_1 = 2, p_1 = 1)$
Sync	$(t_2 = 1, c_2 = 2, p_2 = 0.95)$	$(t_2 = 1, c_2 = 8, p_2 = 0.99)$
FIFO1	$(t_3 = 1, c_3 = 2, p_3 = 0.9,$ $t_4 = 0.5, c_4 = 2, p_4 = 0.9)$	$(t_3 = 1, c_3 = 5, p_3 = 1,$ $t_4 = 1, c_4 = 5, p_4 = 0.99)$

Figure 2.28: QoS values for the channels constituting the ordering circuit

Using these offerings, the QoS values for the two versions of $Ordering$ are computed as follows.

- **provider1**: $t = 1.5$, $c = 9$, $p = 0.7695$;
- **provider2**: $t = 2$, $c = 20$, $p = 0.9801$.

That is, by referring to the QCA shown in Figure 2.27, $t = max(t_1, t_2, t_3) + t_4$, $c = \sum_{i=1}^{4} c_i$, and $p = \prod_{i=1}^{4} p_i$. Suppose that, for the Mary Scenario, a QoS requirement on the ordering circuit states that its memory cost should be no more than 15 units and its reliability should be greater than 90 percent. Neither of the above service providers meet these requirements. However, we can choose $Sync$ offered by **provider1**, $SyncDrain$ and $FIFO_1$ offered by **provider2**, and compose them together to obtain a version of the circuit that is suitable with respect to the global QoS requirement. Now the QoS values for this version of the circuit are: $t = 2$, $c = 14$, and $p = 0.9405$; which satisfy the non-functional requirements.

It is worthwhile noticing that we can reason about sub-connectors of the *Mary Scenario*, much in the same way as we reasoned about the ordering circuit. This allows us to verify non-functional requirements of the entire scenario. This essentially follows from the compositionality of the operational semantics of Reo connectors and their QoS attributes in terms of QCA.

2.4.1.3 Advantages and disadvantages

Essentially, the approach described in [13] shares exactly the same benefits as those of basic Reo [10], since the underlying models are identical after abstracting away the quantitiative information. Therefore Quantitative Reo connectors modelled as QCAs enjoy the following properties: **compositionality**, **incrementality**, **scalability**, and **reusability**; and support both **compositional reasoning** and **evolution**, as discussed in Section 2.3.2.3. Furthermore, contrariwise to the ABT model, the QCAs model allows architects to take into account **non-functional properties** of the components' and connectors' protocol, hence implying further benefits:

- The ability to model QoS attributes of the component/connector parts.
- The ability to combine them to model/infer the QoS attributes of (composite) components and connectors.
- The ability to perform non-functional analysis of the system made of these components and connectors.

Finally, the Eclipse Coordination Tool [4] provides a graphical editor for (Q)CAs. Hence another beneficial feature enjoyed by Reo connectors modelled as QCAs is **tool support**.

2.4.2 Continuous-time probabilistic Reo connectors (QIA)

In the previous section we saw that we could augment Reo channels with a selection of QoS values pertaining to non-functional properties to be verified. We then introduced the notion of a Q-algebra; an algebraic structure used to generate QoS values for an entire Reo network by composing values from the constituent channels. These resultant QoS values may be used to verify properties of the system as a whole, such as, "Is the execution time less than 0.3 seconds?".

Despite the genial elegance of this formalism, it has a considerable shortcoming in that it does not take into account the environment in which the Reo network interacts. To fully appreciate this statement, consider a Reo network that forwards data from one entity to another. Suppose the source entity wishes to send 20 messages, and we wish to know how long this will take. Not only does the duration depend on the transmission time through the Reo network, but it is also dependent on the rate at which the source entity can emit the data (as well as the rate at which the target can receive messages). Without resorting to discourse, it is clear to see that Reo connectors with QoS guarantees are inadequate for modelling many non-functional properties.

To cater for this deficiency, we present continuous-time probabilistic Reo connectors along with their semantic model, quantitative intentional automata, as introduced by Arbab et al in [12]. First, however, we state the WHAT-WHY-HOW description:

- **WHAT -** To define a compositional and operational model for reasoning about Reo circuits exhibiting stochastic behaviour.
- **WHY -** Stochastic behaviours can greatly influence the performability of a system, and so we should try to model this.
- **HOW -** By introducing quantitative intentional automata, an extension of constraint automata that keeps track of the stochastic properties.

2.4.2.1 Overview

We begin by introducing Stochastic Reo, an extension of Reo in which channels are ascribed with stochastic information. Following on from this, we show how Stochastic Reo maps onto quantitative intentional automata, an extension of QCA, and briefly consider compositionality of such models.

Stochastic Reo

Stochastic Reo consists of the same set of underlying channels as basic Reo, except that we annotate them with stochastic rates in two ways:

1. *End-point annotations.* We assign a rate to each of the end-points of a channel. This corresponds to the rate at which data can flow into or out of the channel, depending on whether the end is a source or sink respectively. These rates are generally perceived to be determined by the environment.
2. *Channel annotations.* The internal behaviour of a channel can also exhibit stochastic behaviour, thus we assign a number of rates to the channel corresponding to its different 'behaviours'. This shall be made clear in the commentary below.

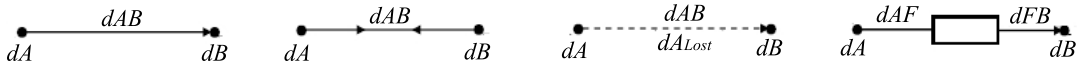

Figure 2.29: Stochastic Reo channels

Figure 2.29 shows four of the basic Reo channels augmented with the two types of stochastic rates as described previously. The synchronous channel receives data at a rate of dA and pushes it out to the environment at a rate of dB. The rate dAB corresponds to the time for the channel to process its data (i.e. push the data through the pipeline). The synchronous drain channel behaves analogously. The remaining two channels (the lossy synchronous and FIFO1 channels) have more than one internal behaviour. For example, the lossy synchronous channel can successfully transmit its data with a rate of dAB, but if the two end-points are not able to synchronise then it will lose its data item with a rate of dA_{Lost}. Similarly, the FIFO1 channel has a behaviour associated with writing to the buffer and another one relating to reading from the buffer. The associated rates for these operations are dAF and dFB respectively.

Quantitative Intentional Automata (QIA)

For Reo networks with QoS guarantees we made use of quantitative constraint automata (QCA) to model the behaviour of the system. We now introduce quantitative intentional automata (QIA) as an extension of QCA for modelling continuous-time probabilistic Reo connectors. The commonality between QCA and QIA is vast; in fact QIA just add more state to QCA in order to represent the status of pending synchronisations on nodes. These pending synchronisations become apparent because the channels are annotated with rates - synchronisation is no longer an atomic operation.

Definition. A *quantitative intentional automaton* is a tuple $\mathcal{A} = (S, S_0, \mathcal{N}, \rightarrow)$ where:

- $S \subseteq L \times 2^{\mathcal{N}}$ is a finite set of states, each having two components: one being a system configuration $l \in L$, where L is a finite set of system configurations; the other being a set $R \in 2^{\mathcal{N}}$, which is indicative of the pending nodes (i.e. nodes that are ready to synchronise, but are blocked by other nodes).
- $S_0 \subseteq S$ is the set of initial states.
- $\mathcal{N}$ is a finite set of nodes.
- $\rightarrow \subseteq \bigcup_{M,N \subseteq \mathcal{N}} S \times \{M\} \times \{N\} \times DC(N) \times 2^{DI} \times S$ is the transition relation, where $DI \subseteq 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathbb{R}_0^+$ is a finite set of delay information tuples.

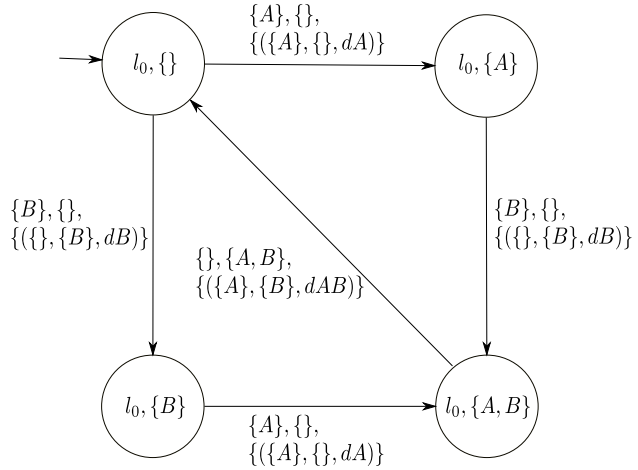

Figure 2.30: QIA of the Stochastic Reo synchronous channel shown in Figure 2.29

A transition of the form $\langle l, R\rangle \xrightarrow{M,N,g,D} \langle l', R'\rangle$ asserts that when the system is in state l with set of nodes R pending, the automaton can attempt to synchronise on nodes in M (they may block, however) after which nodes in N successfully synchronise and are released. This transition is subject to the data constraint g being satisfied and also induces a set D of delay information tuples, containing the stochastic behaviour. In this transition the automaton moves to a new control state l' and the set of pending nodes becomes R'.

In fact, the transitions as presented above are too general. We restrict the actions by imposing the following constraints:

1. $N \subseteq R \cup M$: data flow through nodes in N can only occur if they are enabled for synchronisation.
2. $M \cap R = \emptyset$: nodes that synchronise cannot be pending (i.e. blocked).
3. $(R \cup M) \setminus N = R'$: the resulting pending nodes are those that were originally pending, plus those wishing to synchronise, minus those that actually do synchronise.
4. $(N \neq \emptyset \Rightarrow M \subseteq N) \wedge (N = \emptyset \wedge M \neq \emptyset \Rightarrow |M| = 1)$: If nodes are to be released then all those in M must synchronise. If no nodes are released, then only one data request may be received at any given instance. This constraint seems a little peculiar at first, but ensures that the automaton obeys the stochastic race condition.
5. $\bigcup_{j\in J}(I_j \cup O_j) = N \cup M$, where $D = \{(I_j, O_j, r_j) : j \in J\}$: this ensures that there are stochastic rates associated with every node that takes part in a synchronisation. $(I, O, r) \in D$, where I is a set of source nodes and O is a set of sink nodes, has one of two interpretations. It may mean that with rate r data flows from I to O, or alternatively it could mean that data arrives at nodes in $I \cup O$ with rate r. Either way, the handling is the same.

Figure 2.30 shows the QIA for the basic synchronous Stochastic Reo channel presented in Figure 2.29.

Composition of QIA

In the previous section on Reo with QoS guarantees we considered the composition of two QCA via a product operator $\bowtie$. Intuitively, this operator forced the QCA to synchronise on the common nodes, but also allowed them to proceed independently when they didn't interact. This was achieved by a product construction on the two QCA, and a set of rules for combining the transitions of both.

In a similar manner, the QIA for a Stochastic Reo network is built by composition of the QIA of the constituent channels. Again, we make use of a product construction, but the transition relation has far more complexity than the case of QCA.

As a prerequisite for a data transfer to take place in Stochastic Reo, we require that all of the local end-points are ready to accept/push data. For QCA this is atomic, but in QIA it is possible that some nodes are ready to synchronise yet are blocked (i.e. they are pending) until some other nodes are ready to synchronise. We therefore have to ensure in the composition that for any given data flow through a channel, all of the incident nodes are active and ready to synchronise.

The method by which the transition relation achieves this is well-defined, although we choose not to reproduce it here (the construction may be consulted in [12]). Considering the transition relation pictorially, every data transfer transition is usually preceded by all permutations of transitions corresponding to nodes becoming enabled (for nodes that must be enabled to allow the data transfer). This is evident by virtue of the fact that the transition relation only allows one node to become enabled at any given time.

A further complication that must be considered is the fact that the end-point rates for channels that are connected to mixed nodes shouldn't exist in the resulting transition system, because mixed nodes are permanently enabled. Recall that the product operator can convert some source and sink nodes to mixed nodes in the composition, hence it is necessary to remove the rates from channels that will coincide with these nodes. This erasure of rates is coded in the product transition relation.

Conversion of QIA to CTMCs

Recall that transitions of QIA are annotated with collections of delay information tuples. Each tuple contains a rate together with a set of input ports and a set of output ports. These rates can be used to construct a CTMC model, whose transitions closely mirror the structure of the QIA. If there is more than one delay information tuple on a transition, we need to eliminate this so that the resulting CTMC is well-defined. There are two cases to consider:

- If there is a causality chain between some of the tuples, i.e. two tuples of the form $(A, B, r), (B, C, r')$, we split the transition into two sequential transitions that respect the causality.
- If there is no causality between the tuples, we take all sequential permutations of the tuples and interleave these.

It's probable that there are some delay information tuples that have a causality relation between them and some that don't for a given transition. In these cases we need to combine the two rules above in the intuitive way. The construction method can be made precise by employing delay-sequences, which are described in the Technical Report of [12].

2.4.2.2 Scenario modelling

We model the *Mary Scenario* in Stochastic Reo, by annotating our basic Reo model from Figure 2.9 in Section 2.3.2.2 with stochastic rates. A small portion of the resulting QIA, generated by ECT, is provided in Figure 2.31.

As this example has shown for even the simplest of Stochastic Reo networks, the underlying QIA grows at a truly horrendous rate. The generated QIA, under the not so certain assumption that ECT is implemented correctly, contains 2048 nodes[10]. Thankfully, the construction of the QIA from a Stochastic Reo network is completely mechanical and straightforward assuming we have sufficient processing power and memory. Moreover, the extension from basic Reo to Stochastic Reo is incredibly simple from a practitioner's perspective, providing the stochastic rates are readily available.

Incidentally, the first state s_1 in the diagram is correct as it has 10 outgoing edges, each of which coincides with one of the 10 ports in the original Stochastic Reo network. Thus each of the initial transitions correspond to an attempt to send or receive data on one of the ports depending on whether it is a sink or a source. The stochastic race condition ensures that only one data operation attempt can take place at a given moment, which is why there are 10 transitions instead of 2^{10}. As for the remaining 2047 nodes, that is anybody's guess.

Should we be surprised that there are 2048 states in the QIA? Of course not, as the 10 ports result in $2^{10} = 1024$ combinations of states pending reading or writing operations depending on whether they are a source or a sink. The FIFO buffer in the Stochastic Reo network then doubles the total number of states, giving us the requisite 2048.

[10] Verified by checking the XML description of the QIA.

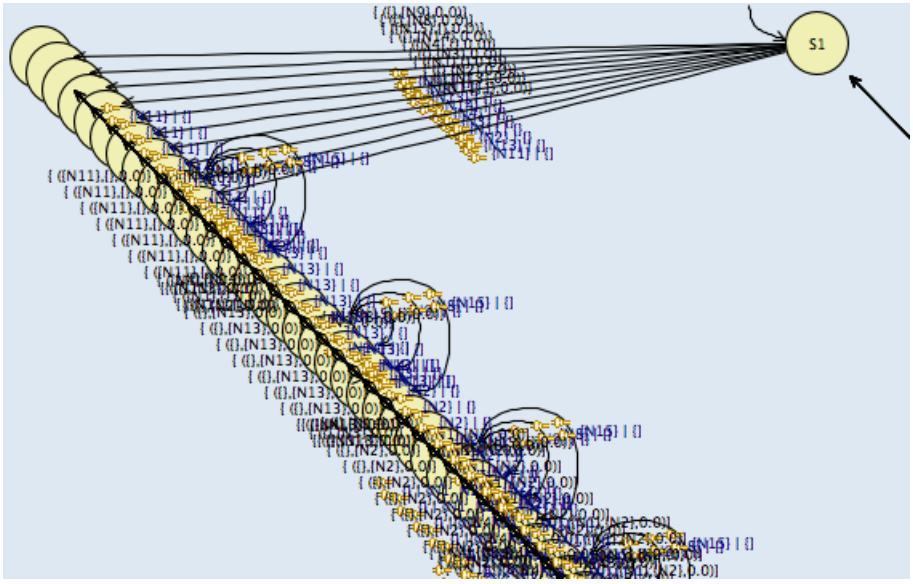

Figure 2.31: QIA for the Mary Scenario

2.4.2.3 Advantages and disadvantages

Naturally, continuous-time probabilistic Reo connectors share many of the advantages and disadvantages of basic Reo. **Compositionality** is obtained on account of the fact that Reo connectors can be connected to each other via nodes, and also that the underlying QIA can be composed. As a result, we also have **incrementality** and **scalability** of these connectors. **Compositional reasoning** will almost certainly be guaranteed for simple assertions, but in the case of complex assertions it is likely that we will have to compose the constituents together in order to perform some analysis. This is because composition of CTMCs is incredibly difficult and not always well-defined.

As these connectors are in essence based upon traditional Reo, **reusability** will hold, although we may choose to distinguish connectors that are essentially built the same way, but that are annotated with different stochastic information. This, of course, depends on our interpretation of the term reusable, but there should be no reason why a connector cannot be reusable.

It is ambiguous to say whether continuous-time probabilistic Reo connectors satisfy the **evolution** dimension. Certainly connectors can be disconnected from components and moved around, but evolution may also involve the alteration of stochastic rates, and it is not so clear to see whether this is supported. However, online verification may have an answer to this.

There is full **tool support** for these connectors in ECT [4]. Channels can be ascribed with stochastic rates, and the tool is able to produce the corresponding QIA (including in a graphical format), together with performing the composition of QIA. The tool is also able to automatically generate the corresponding CTMC, and there is a facility for exporting the CTMC directly as a PRISM model [15].

The key advantage of these Reo connectors is their ability to model the rates of data transmission in the environment. This gives us a good indication of how the connector will behave over time.

2.4.3 Discrete-time probabilistic Reo connectors (PCA)

In this section, we introduce Reo connectors that can exhibit both discrete probabilistic and non-deterministic behaviours. Up until now, Reo connectors have been fully deterministic - their actions being completely prescribed by the current configuration of the environment in which the connector is situated. However, this is a fairly unrealistic assumption to make as, invariably, the world in which our connectors will reside is inherently unpredictable. We now state the WHAT-WHY-HOW description:

- **WHAT -** To define a compositional and operational model for reasoning about unreliable Reo circuits.

- **WHY -** We often have quite dependable data telling us the probability that a system component can fail, so we should use this to determine probabilistic behaviour of the system as a whole.
- **HOW -** By introducing probabilistic constraint automata, an extension of constraint automata that captures the ability for Reo channels to exhibit probabilistic behaviour.

2.4.3.1 Overview

We will consider two types of discrete-time probabilistic Reo channels: *simple channels* and *non-simple channels*. The difference between simple and non-simple channels becomes apparent when we consider probabilistic constraint automata (PCA), which are semantic models for these Reo channels. Intuitively, however, simple channels have probabilistic failures that are independent of data-flow, whereas in non-simple channels, the probability of failure is dependent upon the I/O operations.

Figure 2.32: Examples of simple and non-simple channels

Figure 2.32 contains two channels of the simple and non-simple type respectively from left to right. The left-most channel can lose any data item about to be written to the buffer from A (resulting in the buffer *remaining* empty) with probability τ. This is not the same as the buffer losing a data item with probability τ that has already been written to it. The second, a probabilistic lossy channel, can transfer data from A to B with probability $1 - \tau$ providing A and B are willing to synchronise. Alternatively, the channel can lose any item to be written at A with probability τ, *but only providing A and B are willing to synchronise*.

Semantics

Simple and non-simple channels may be ascribed semantics in terms of simple probabilistic constraint automata (SPCA) and probabilistic constraint automata (PCA) respectively, with the latter being a generalisation of the former. These were introduced by Christel Baier in [18], and are closely related to Segala's simple and general probabilistic automata [62].

Definition. A *(simple) probabilistic constraint automaton* is a tuple $(Q, \mathcal{N}, \rightarrow, Q_0)$, where

- Q is a countable set of states
- $\mathcal{N}$ is a finite set of nodes
- $Q_0 \subseteq Q$ is a set of initial states
- $\rightarrow$ is the transition relation, and is defined as follows:
  - For SPCA:
$$\rightarrow \subseteq \bigcup_{N \subseteq \mathcal{N}} Q \times \{N\} \times DC(N) \times Distr(Q)$$
  - For PCA:
$$\rightarrow \subseteq Q \times Distr\left(\bigcup_{N \subseteq \mathcal{N}} (\{N\} \times DC(N) \times Q)\right).$$

For an arbitrary P, $Distr(P)$ is a function $P \rightarrow [0, 1]$ such that $\Sigma_{p \in P} p = 1$.

The transition system of an SPCA encodes the following behaviour. Assume that $(q, N, g, \pi) \in \rightarrow$ is the next transition to be made. So the automaton is currently in state q, and is about to perform I/O operations on the nodes in N, such that g holds. The resulting state will be q' with probability $\pi(q')$. There may have been multiple transitions enabled in q, so the choice of (q, N, g, π) is non-deterministic. After the non-determinism has been resolved, the resulting state is determined probabilistically.

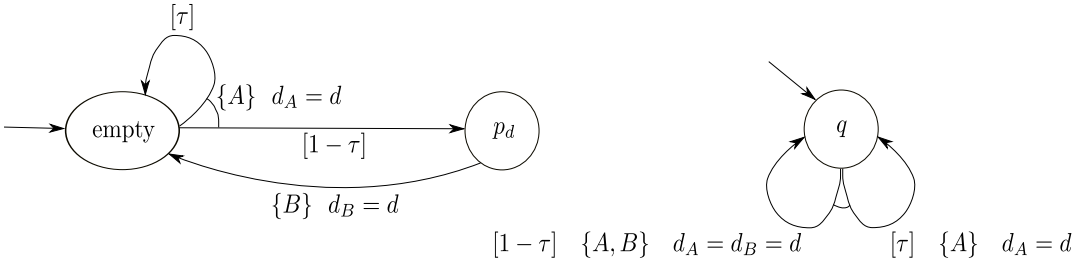

Figure 2.33: SPCA and PCA for the channels shown in Figure 2.32

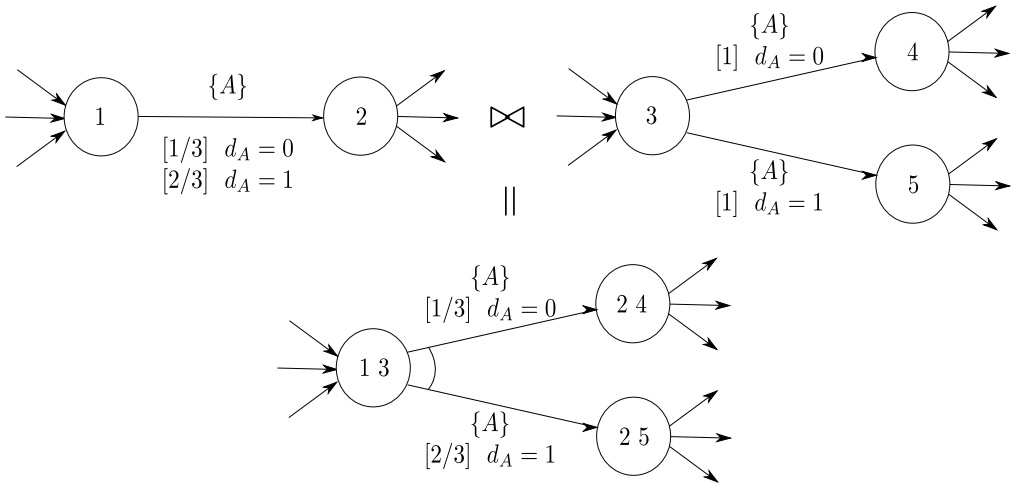

Figure 2.34: Joining transitions with different data constraints

The transition system of a PCA is far more elaborate, in that the choice of I/O operations is probabilistic (along with the successive state) instead of being non-deterministic. There are a few technical restrictions on the form of transitions in the system, specifically to deal with the data constraints. These rules are covered extensively in [18]. It is also clear to see that any SPCA can be embedded in a PCA.

Figure 2.33 shows both an SPCA (left) and a PCA (right) for the basic channels shown in Figure 2.32. Note that on SPCA, the data constraints are the same for all probabilistic transitions, whereas for the PCA they can differ.

Compositionality

As for ordinary constraint automata, both SPCA and PCA may be composed out of smaller automata by using a product construction. The case of SPCA is by far the easiest, as it closely follows the definition of the product operator on constraint automata. In fact, the only difference is that when two transitions are synchronised, the resulting probability distribution is the pointwise product of the two constituent distributions i.e. $(\pi_1 * \pi_2)\langle p_1, p_2\rangle = \pi_1(p_1) \cdot \pi_2(p_2)$.

The product operator on SPCA has all of the usual nice properties, in that it is both commutative and associative. There is also a hiding operator that can be used for abstraction.

The composition operator on PCA is thwart with technicalities to ensure that the resulting automaton respects the data constraints. Recall that for PCA, data constraints are not necessarily chosen non-deterministically, but also probabilistically. This means that the synchronisation of two nodes may result in the joining of multiple transitions with compatible data constraints. This is shown in Figure 2.34.

We now remark on key properties of the join operator on PCA. Thankfully, joining is a commutative operation, but associativity is not so clear cut. It turns out that the product is associative in the case that all of the nodes to be joined in one of the automata is exactly the set of all source nodes in that automaton. We refer to this restricted form of product as a concatenation. However, if this precondition is not met, then the product operator is not associative in general. Unfortunately, there is no acceptable solution to

this issue.

PCA to Markov Decision Processes

A PCA automatically has the semantics of a Markov Decision Process (MDP), because for each state we have a non-deterministic choice over probability distributions for the successive state. A PCA can therefore easily be converted to the textual representation of an MDP as input for quantitative analysis software, such as PRISM.

2.4.3.2 Scenario modelling

As mentioned in the previous section, the composition operator on PCA is abhorrent and furthermore there is no tool support at present. Under these conditions, we choose not to model any of the scenarios using PCA, as quite frankly, the resulting MDP does not convey that much comprehensible information, and the manual construction would be unwieldy.

2.4.3.3 Advantages and disadvantages

Again, there is much overlap between the features of ordinary Reo and those of discrete-time probabilistic Reo connectors. Although we have **incrementality** for these Reo connectors, we do not necessarily have **compositionality**. The SPCA fragment enjoys compositionality, but recall that composition in PCA is not generally associative. **Scalability** is retained as we still have the same method of connecting smaller channels and connectors via nodes.

It is difficult to know whether the formalism supports **compositional reasoning**, although we have made promising progress in showing this for SPCA. As for **reusability**, these connectors could be reused although we have to distinguish connectors consisting of the same topology of channels, but with different probabilistic information.

As for continuous-time probabilistic Reo connectors, we have **evolution** in the sense that connectors can be moved around and connected to different components. There is also likely to be support for the alteration of probabilistic values over time through the use of online verification techniques.

There is currently no **tool support** for discrete-time probabilistic Reo connectors. This is probably because of the lack of compositionality of PCA and the incredibly complex method of trying to join PCA together.

In summary, although discrete-time probabilistic Reo connectors offer us many useful features for modelling discrete events such as failures, the shortcomings in the formalism, namely the lack of compositionality and the cumbersome join operators, make it rather difficult to deal with. However, we still gain quite a bit of expressivity by just considering SPCA, which is more well behaved, so it is always a possibility to just deal with this fragment.

2.5 Conclusion

The main role of the survey is the evaluation of the connector algebra formalisms. The following tables summarise the outcomes of the comparison of the approaches discussed in Sections 2.3 and 2.4 and of their evaluation with respect to the eight dimensions of interest for CONNECT introduced in Section 2.1. We recall that the definition of these dimensions have been kept as general as possible and subsequently refined/instantiated in order to make them assessable for all the approaches described in this deliverable.

For a given formalism and dimension, in the following tables, we use “Yes”, “No”, “Maybe”, and “Partially” to indicate the evaluation result. Obviously, the first two values are used to indicate that the formalism under consideration enjoys or does not enjoy at all, respectively, the ability/property referred to by the indicated dimension. The evaluation result “Maybe” is used to indicate that the considered dimension is not supported due to the fact that the formalism makes use of a notation that limits the expressive power of the underlying theoretical formalism, and not due to the fact that the formalism does not support that dimension in principle. For instance, as discussed in Section 2.3.1.3, this is the case of role-glue connectors in WRIGHT (by Allen and Garlan) with respect to compositionality. “Partially” means that the considered dimension is in general supported only by a subset of all the connectors that can be specified

by the approach. For instance, this is the case for the connector algebra in [27], where compositionality is in general supported only on the subset of similarly typed connectors. "Partially" is also used for the *reusability* dimension to indicate that the connector models abstract connectors that cannot be reusable in any context, but they can be reusable under specific conditions (e.g., port-role compatibility). Clearly, this dimension also accounts for the different meanings that can be given to the notion of context, e.g., other connectors, system components, or both of them. For example, this is the case for the connector algebra in [27] but also for the role-glue connectors in WRIGHT (by Allen and Garlan). Note that being *partially reusable*, instead of completely reusable, is not always a complete limitation. In fact, although the contexts of reuse are sometimes limited, correctness of the reuse may be achieved by construction.

Approach	Compositionality	Incrementality	Scalability	Compositional reasoning
Role-glue connectors in WRIGHT	Maybe	No	No	Yes
Reo connectors as ABTs	Yes	Yes	Yes	Yes
Connectors as Kell calculus processes	Yes	Yes	Yes	Yes
BIP connectors	Partially	Yes	Yes	Maybe
Bigraphical Reactive System connectors	Yes	Yes	Yes	Maybe partially
Reo connectors with QoS as CA	Yes	Yes	Yes	Yes
Reo connectors as QIA	Yes	Yes	Yes	Partially
Reo connectors as PCA	Partially	Yes	Yes	Partially

Table 2.1: Summary of the evaluation results of existing formalisms (Part 1)

Approach	Reusability	Evolution
Role-glue connectors in WRIGHT	Partially	No
Reo connectors as ABTs	Yes	Partially
Connectors as Kell calculus processes	Partially	Yes
BIP connectors	Partially	No
Bigraphical Reactive System connectors	Yes	Yes
Reo connectors with QoS as CA	Yes	Partially
Reo connectors as QIA	Yes	Partially
Reo connectors as PCA	Yes	Partially

Table 2.2: Summary of the evaluation results of existing formalisms (Part 2)

Approach	Non-functional properties	Tool support
Role-glue connectors in WRIGHT	No	Yes
Reo connectors as ABTs	No	Yes
Connectors as Kell calculus processes	No	No
BIP connectors	No	Yes
Bigraphical Reactive System connectors	No	On the way
Reo connectors with QoS as CA	Yes, Q-algebras	Yes
Reo connectors as QIA	Yes, stochastic rates	Yes
Reo connectors as PCA	Yes, discrete probability	No

Table 2.3: Summary of the evaluation results of existing formalisms (Part 3)

Considering the above tables, we can conclude that none of the formalisms considered in Chapter 2 fulfils all eight dimensions of interest for CONNECT. Defining an appropriate connector algebra will be our main task in the next stage.

In addition to the eight dimensions, usability of the future connector algebra is an important issue. For example, we have used Reo to model the procedure for clients to browse products in the market in the popcorn scenario[11], in order to have a deeper insight of Reo. The Reo model is shown in Figure 2.35. The client sends a request to the market from its output port, and the market receives it from its input port. Then the market returns the catalogue from the output port and the client receives it from the input port. The CONNECTors "Market Interface" and "Client Interface" represent the external behaviour of the market and the client respectively. They can also be seen as the partial view of the components. Note that the client can receive the response only after it issues a request, and the market can send out the response only after it obtains the request. The FIFO buffers and SyncDrain channels in these two CONNECTors

[11] The detail of this scenario can be seen in the deliverable D1.1 [2].

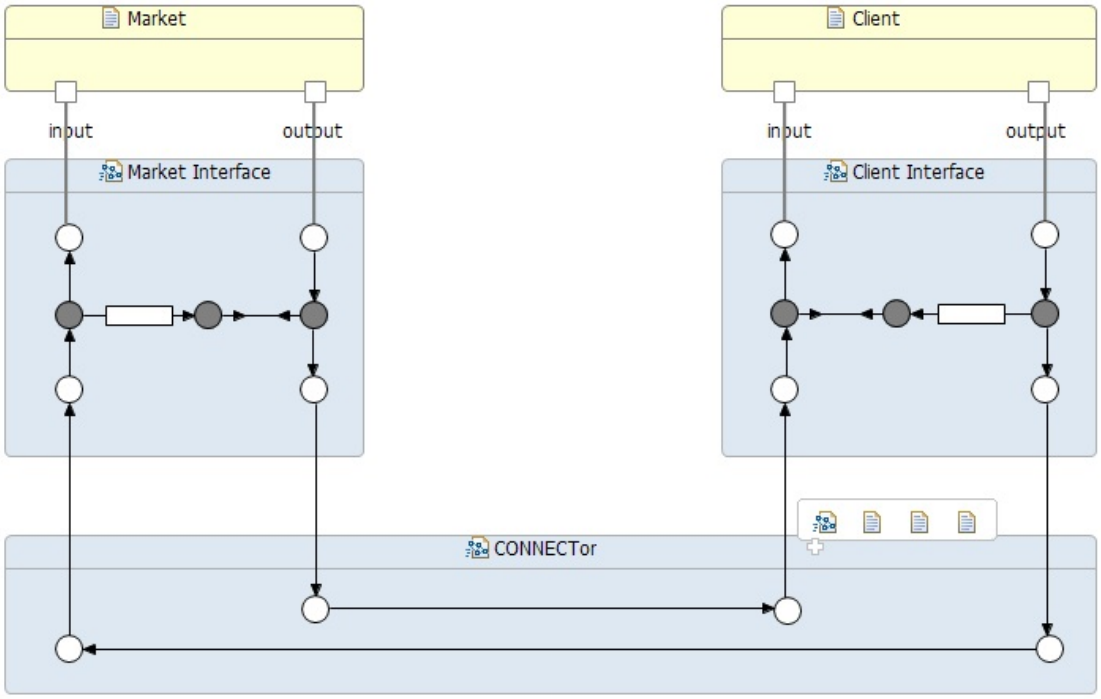

Figure 2.35: The Reo model for the browse procedure in the popcorn scenario

guarantee the above orders. The “CONNECTor” in the figure is a CONNECTor synthesised on the fly by the enabler to allow the communication. In this small example, it simply transmits the request and the response. During the modelling exercise, we realised that the notation used to model the scenario is quite complex and sometimes unintuitive (e.g., using the FIFO buffer to implement the data flow order). Thus, practitioners may be discouraged from using Reo.

3 Quantitative verification

As CONNECTors often work in a highly dynamic and distributed environment, they could behave in an unreliable way. For instance, a CONNECTor in the popcorn scenario may not always transmit a response message successfully to clients due to competition on available channels, interfere with other CONNECTors, etc. In many cases, unreliable behaviours can be formalised as probabilistic behaviours, such as the percentage of successful transmissions on average.

This chapter is devoted to quantitative verification for CONNECTors, i.e., the verification of their probabilistic behaviours, aiming to establish the probability of some event occurring, or the expected time or reward until a certain state is reached. In this chapter, we are focusing on quantitative models based on *labelled transition systems*, since they are common to the main CONNECT work packages, including synthesis and learning. We thus assume that there is an effective translation from the high-level connector algebra formalism to a low-level transition system. For instance, Section 2.4.2 presented an approach for converting QIA to CTMCs, which will be introduced in Section 3.1.2, and Section 2.4.3 explained that PCA is indeed an MDP.

Section 3.1 gives an overview of *classical* stochastic models, temporal logic formalisms and model-checking algorithms for offline verification, which rely on identifying all possible system states to establish probabilistic properties. This also includes one of our contributions in the project: *new* abstraction-refinement techniques devised and implemented for probabilistic real-time systems. Section 3.2 deals with non-functional characteristics of CONNECTors, usually arising from QoS requirements, that are expressible with *rewards* and the corresponding verification algorithms. The last two sections in this chapter describe new research results from WP2. In Section 3.3, we propose a compositional approach to handle verification of large systems which we aim to apply to CONNECTors. In Section 3.4, we discuss an online verification method (also known as run-time monitoring and adaptation), which employs offline verification using models and properties that are generated at run-time. This online method can be seen as a starting point for a model-checking approach to property verification for evolving connectors, which is relevant to dependability assurance in WP5.

3.1 Off-line probabilistic verification

In this section, we discuss several widely adopted models for probabilistic systems, and the model checking algorithms on these models. How these algorithms are used and implemented is explained in Deliverable D5.1 which concerns V&V for dependability.

In order to perform offline probabilistic verification on CONNECTors, we first translate them into probabilistic models and apply the algorithms presented here.

3.1.1 Model checking DTMC/MDP

DTMCs (Discrete-Time Markov Chains) are the simplest models we employ to perform verification. A well known characteristic of DTMCs is the memoryless property (Markov property): in any state of a system, the next state the system evolves into is determined independently of the execution history, i.e., all states the system passes before entering the current state have no influence on the decision of a successor state. This property makes the computation of probability of taking a path fairly easy. It is done in an incremental way: the probability of reaching the $k+1$th state (s_{k+1}) in the path is the multiplication of the probability of reaching the kth state (s_k) and the probability of executing the transition from s_k to s_{k+1}. We formalise the above concepts as follows.

Definition 3.1 *A DTMC is a tuple* $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ *where*

- S *is a finite set of states;*
- $\overline{s} \in S$ *is the initial state,*
- $\mathbf{P} : S \times S \to [0,1]$ *is a* transition probability matrix*, such that*

$$\sum_{s' \in S} \mathbf{P}(s, s') = 1$$

for all states $s \in S$.

- $L : S \to 2^{AP}$ *is a* labelling function *over the set* AP *of atomic propositions;*

Each entry $\mathbf{P}(s, s')$ in the transition probability matrix gives the probability of making a transition from state s to state s'. If a state has no successor states, we assume it has a self loop on itself with probability 1. The matrix also indicates that DTMCs defined here are *homogeneous*, which means the transition probabilities are independent of the time when transitions are taken.

To model functional requirements, we label each state with a set of atomic propositions that hold in the state (done by the labelling function L). Characterising probabilistic behaviours is specified by PCTL (Probabilistic Computational Tree Logic) [41, 17], which is a probabilistic extension of the temporal logic CTL, over the set AP of atomic propositions and system executions. In the following, we give the formal description of executions (or paths) first and then the syntax and semantics of PCTL.

A path ω is a non-empty sequence of states $s_0 s_1 s_2 \ldots$, where $s_i \in S$ and $\mathbf{P}(s_k, s_{k+1}) > 0$ for all $i \geq 0$. If the path is finite, we denote it by ω_{fin}. The ith state in the path is denoted by $\omega(i)$. A finite path ω_{fin} of length n (i.e., there are n transition in the path) is a *prefix* of an infinite path ω if $\omega_{fin}(i) = \omega(i)$ for $0 \leq i \leq n$. Let $Path_s$ and $Path_s^{fin}$ be the set of infinite paths and finite paths starting in state s respectively. The probability of taking a path is computed through a probability measure $Prob_s$ over $Path_s$ for each state $s \in S$. We first define the probability $\mathbf{P}_s(\omega_{fin})$ for a finite path $\omega_{fin} \in Path_s^{fin}$ to be

$$\mathbf{P}_s(\omega_{fin}) \stackrel{def}{=} \begin{cases} 1 & \text{if } n = 0 \\ \mathbf{P}(\omega(0), \omega(1)) \ldots \mathbf{P}(\omega(n-1), \omega(n)) & \text{otherwise} \end{cases}$$

where n is the length of ω_{fin}. Let $C(\omega_{fin})$ be the set of all infinite paths having prefix ω_{fin}, i.e.,

$$C(\omega_{fin}) \stackrel{def}{=} \{\omega \in Path_s \mid \omega_{fin} \text{ is a prefix of } \omega\}.$$

Let Σ_s be the smallest σ-algebra on $Path_s$ containing all the set $C(\omega_{fin})$ where ω_{fin} ranges over paths in $Path_s^{fin}$. The probability measure $Prob_s$ on Σ_s is the unique measure such that

$$Prob_s(C(\omega_{fin})) = \mathbf{P}(\omega_{fin}) \quad \text{for all } \omega_{fin} \in Path_s^{fin}.$$

Definition 3.2 *The syntax of PCTL is as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi], \quad \psi ::= X\ \phi \mid \phi\ \mathcal{U}^{\leq k}\ \phi \mid \phi\ \mathcal{U}\ \phi$$

where a *is an atomic proposition, and* $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$, $c \in \mathbb{R}^{\geq 0}$ *and* $k \in \mathbb{N}$.

In the above definition, ϕ denotes state formulae and ψ path formulae. The former is evaluated over states and the latter over paths. The formula $\mathcal{P}_{\bowtie p}[\psi]$ holds in a state s if the probability of taking a path from s satisfying ψ is in the interval specified by $\bowtie p$. Let $\omega = s_0 s_1 \ldots$ be a path, where s_i $(i \geq 0)$ is a state. The formula $X\ \phi$ is true in the path ω if ϕ is satisfied in the second state of ω, i.e., s_1; the formula $\phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ is true if ϕ_2 is satisfied within k time-steps and ϕ_1 holds in all states before that point; $\phi_1\ \mathcal{U}\ \phi_2$ is similar to the previous one except there is no bound on the time-steps when ϕ_2 holds. The formal semantics of PCTL is given as follows.

Definition 3.3 *Let* $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L, \mathbf{C})$ *be a DTMC. For any state* $s \in S$*, the satisfaction relation* $\models$ *is defined inductively by:*

- $s \models true$ *for all* $s \in S$,
- $s \models a$ *iff* $a \in L(s)$,
- $s \models \neg\phi$ *iff* $s \not\models \phi$,
- $s \models \phi_1 \wedge \phi_2$ *iff* $s \models \phi_1$ *and* $s \models \phi_2$,
- $s \models \mathcal{P}_{\bowtie p}[\psi]$ *iff* $p_s(\psi) \bowtie p$

where

$$p_s(\psi) \stackrel{def}{=} Prob_s(\{\omega \in Path_s \mid \omega \models \psi\})$$

and for any path $\omega \in Path_s$*:*

- $\omega \models X\ \phi$ *iff* $\omega(1) \models \phi$*,*
- $\omega \models \phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ *iff* $\exists i \leq k\ .\ (\omega(i) \models \phi_2 \wedge \omega(j) \models \phi_1\ \forall j < i)$*,*
- $\omega \models \phi_1\ \mathcal{U}\ \phi_2$ *iff* $\exists k \geq 0\ .\ (\omega \models \phi_1\ \mathcal{U}^{\leq k}\ \phi_2)$*.*

The PCTL model checking algorithm for DTMCs was first presented in [33, 35, 41]. The algorithm takes a DTMC $(S, \overline{s}, \mathbf{P}, L)$ and a PCTL formula ϕ, and produces a set $Sat(\phi)$ of states such that $Sat(\phi) = \{s \in S \mid s \models \phi\}$. Usually, we are only interested in whether the initial state $\overline{s}$ satisfies the formula ϕ, though the algorithm returns the whole set of states in which ϕ holds. To compute $Sat(\phi)$, the algorithm recursively computes the set $Sat(\phi')$ of states satisfying each subformula ϕ' as follows.

- $Sat(true) = S$,
- $Sat(a) = \{s \mid a \in L(s)\}$,
- $Sat(\neg\phi) = S \backslash Sat(\phi)$,
- $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cap Sat(\phi_2)$,
- $Sat(\mathcal{P}_{\bowtie p}[\psi]) = \{s \in S \mid p_s(\psi) \bowtie p\}$.

In the above procedure, checking Boolean operators such as $\wedge$ and $\neg$ are straightforward. Here we only explain the detail for checking operator $\mathcal{P}_{\bowtie p}[\psi]$. The key step is to compute $p_s(\psi)$ for each state $s \in S$. We need to distinguish three cases $\psi = X\ \phi$, $\phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ and $\phi_1\ \mathcal{U}\ \phi_2$.

The $\mathcal{P}_{\bowtie p}[X\ \phi]$ **operator**. Computing $p_s(X\ \phi)$ for each state $s \in S$ is done directly from the transition probability matrix after the set $Sat(\phi)$ is obtained:

$$p_s(X\ \phi) = \sum_{s' \in Sat(\phi)} \mathbf{P}(s, s').$$

The $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}^{\leq k}\ \phi_2]$ **operator**. In this case, we need to partition the set S into three subsets S^{yes}, S^{no} and $S^{?}$, which are defined as follows.

$$S^{yes} = Sat(\phi_2),\ \ S^{no} = S \backslash (Sat(\phi_1) \cup Sat(\phi_2)),\ \ Sat^{?} = S \backslash (S^{yes} \cup S^{no}).$$

Obviously, the set S^{yes} of states satisfy the formula $\phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ (i.e., $p_s(\phi_1\ \mathcal{U}^{\leq k}\ \phi_2) = 1$), and the set S^{no} of states do not (i.e., $p_s(\phi_1\ \mathcal{U}^{\leq k}\ \phi_2) = 0$). The probability p_s for the set $S^{?}$ of states is computed recursively as follows.

$$p_s(\phi_1\ \mathcal{U}^{\leq k}\ \phi_2) = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot p_{s'}(\phi_1\ \mathcal{U}^{\leq k-1}\ \phi_2) & \text{if } k \geq 1 \end{cases}$$

The $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}\ \phi_2]$ **operator**. Like the previous case, we partition S into S^{yes}, S^{no} and $S^{?}$, and $p_s(\phi_1\ \mathcal{U}\ \phi_2) = 1$ and 0 for S^{yes} and S^{no} respectively. However, the procedures to determine these three sets are more complex than the "bounded until" $\mathcal{U}^{\leq k}$ case.

$$\begin{gathered} S^{no} = Prob0(Sat(\phi_1), Sat(\phi_2)), \\ S^{yes} = Prob1(Sat(\phi_1), Sat(\phi_2), S^{no}), \\ S^{?} = S \backslash (S^{yes} \cup S^{no}). \end{gathered}$$

The **while** loop in $Prob0$ in Algorithm 1 incrementally computes the set of states from which with non-zero probability, a state satisfying ϕ_2 can be reached without leaving states satisfying ϕ_1. Indeed, it implements a fix point computation. By subtracting this set from S, we obtain S^{no}. The **while** loop in $Prob1$ in Algorithm 2 works in a similar way. It incrementally computes the set of states that cannot satisfy $\phi_1\ \mathcal{U}\ \phi_2$ with probability 1. Such a state must satisfy one of the two conditions:

Algorithm 1 $Prob0(Sat(\phi_1), Sat(\phi_2))$

1: $R := Sat(\phi_2)$
2: $done := \mathbf{false}$
3: **while** $done = \mathbf{false}$ **do**
4: $\quad R' := R \cup \{s \in Sat(\phi_1) \mid \exists s' \in R \,.\, \mathbf{P}(s, s') > 0\}$
5: $\quad$ **if** $R' = R$ **then**
6: $\quad\quad done := \mathbf{true}$
7: $\quad$ **end if**
8: $\quad R := R'$
9: **end while**
10: **return** $S \backslash R$

Algorithm 2 $Prob1(Sat(\phi_1), Sat(\phi_2), S^{no})$

1: $R := S^{no}$
2: $done := \mathbf{false}$
3: **while** $done = \mathbf{false}$ **do**
4: $\quad R' := R \cup \{s \in (Sat(\phi_1) \backslash Sat(\phi_2)) \mid \exists s' \in R \,.\, \mathbf{P}(s, s') > 0\}$
5: $\quad$ **if** $R' = R$ **then**
6: $\quad\quad done := \mathbf{true}$
7: $\quad$ **end if**
8: $\quad R := R'$
9: **end while**
10: **return** $S \backslash R$

1. It is in S^{no};
2. ϕ_1 holds in the state, but ϕ_2 does not. Further, it can reach a state satisfying the condition 1 or 2 in one step with non-zero probability.

After obtaining S^{no}, S^{yes} and $S^{?}$, we compute $p_s(\phi_1 \; \mathcal{U} \; \phi_2)$ for all states in S recursively by letting $x_s = p_s(\phi_1 \; \mathcal{U} \; \phi_2)$ and solving the linear equation system in x_s:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot x_{s'} & \text{if } s \in S^{?} \end{cases}$$

MDPs (Markov decision processes) are a generalisation of DTMCs. They allow both probabilistic behaviour and non-deterministic behaviour. It is very useful when we model concurrency, e.g., several probabilistic transitions in parallel, or when the exact probability distribution of a transition is not known or not relevant. Let $Dist(S)$ be the set of all probability distributions over S, i.e., the set of functions $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. An MDP is formally defined as follows.

Definition 3.4 *An MDP is a tuple* $\mathcal{M} = (S, \overline{s}, Act, Steps, L)$ *where*

- S *is a finite set of* states*,*
- $\overline{s} \in S$ *is the* initial state*,*
- Act *is a set of* actions*,*
- $Steps : S \to 2^{Act \times Dist(S)}$ *is the* probabilistic transition function*.*
- $L : S \to 2^{AP}$ *is a* labelling function*,*

An MDP is similar to a DTMC except that the transition probability matrix $\mathbf{P}$ is replaced by $Steps$, which maps each state $s \in S$ to a finite non-empty subset of $Act \times Dist(S)$. For each state $s \in S$, $Steps$ defines a set of enabled actions that can be performed in s and associates a probability distribution function μ to each enabled action a. There are two steps to determine a successor state of s: first, an action is chosen non-deterministically from the set of enabled actions; next, the probability of moving to state s' is decided by μ, i.e., $\mu(s')$.

A *path* in an MDP is a non-empty sequence of the form

$$s_0 \xrightarrow{(a_1,\mu_1)} s_1 \xrightarrow{(a_2,\mu_2)} s_2 \ldots$$

where $s_i \in S$, $(a_{i+1}, \mu_{i+1}) \in Steps(s_i)$ and $\mu_{i+1}(s_{i+1}) > 0$ for all $i \geq 0$. We still use $\omega(i)$ to denote the ith state in the path ω, ω^{fin} to denote a finite path, $last(\omega_{fin})$ to denote the last state in ω_{fin}, and $Path_s^{fin}$ ($Path_s$) to denote the set of all finite (infinite) paths starting in state s. Moreover, the ith action in ω is represented by $step(\omega, i)$.

To resolve the non-deterministic choices when we execute an MDP, we employ an *adversary* to select a choice based on the history of choices made so far.

Definition 3.5 *An adversary A of an MDP $\mathcal{M}$ is a function mapping every finite path ω_{fin} onto an element $A(\omega_{fin})$ of the set $Steps(last(\omega_{fin}))$. Let $Adv_{\mathcal{M}}$ denote the set of all possible adversaries of the MDP and, for any adversary A, let $Path_s^A$ denote the subset of $Path_s$ which corresponds to A.*

Since an adversary of an MDP $\mathcal{M} = (S, \overline{s}, Act, Steps, L)$ removes non-deterministic behaviour, the behaviour from $s \in S$ forms an infinite-state DTMC $\mathcal{D}^A = (S^A, \overline{s}^A, \mathbf{P}^A, L^A)$ such that

- $S^A = Path_s^{fin}$,
- $\overline{s}^A = s$,
- $L^A(s^A) = L(last(s^A))$ for all state $s^A \in S^A$

and for two finite paths $\omega, \omega' \in S^A$:

- $\mathbf{P}^A(\omega, \omega') = \begin{cases} \mu(s') & \text{if } \omega' \text{ is of the form } \omega \xrightarrow{(a,\mu)} s' \\ 0 & \text{otherwise.} \end{cases}$

From the above construction, we know that each state s^A in $\mathcal{D}^A$ represents a path $\omega_{fin} \in Path_s^A$ of $\mathcal{M}$, and the path from $\overline{s}^A$ to s^A can be mapped to ω_{fin} uniquely. Therefore, we are able to use the probability measure over DTMCs to define a probability measure $Prob_s^A$ over the set of path $Path_s^A$.

PCTL defined over DTMCs can be naturally extended to MDPs. The syntax is the same as before. The major difference for MDPs is the semantics of the $\mathcal{P}_{\bowtie p}[\psi]$ operator:

- $s \models \mathcal{P}_{\bowtie p}[\psi]$ iff $p_s^A(\psi) \bowtie p$ for all $A \in Adv_{\mathcal{M}}$

where for any adversary $A \in Adv_{\mathcal{M}}$:

$$p_s^A(\psi) \stackrel{def}{=} Prob_s^A(\{\omega \in Path_s^A \mid \omega \models \psi\}).$$

In addition, we may like to reason about the *existence* of an adversary that satisfying a property, instead of asking all adversaries to satisfy the property. This can be done via translation to a dual property in the following way:

- $p_s^A(\psi) > p$ for some adversary A is equivalent to $\neg\mathcal{P}_{\leq p}[\psi]$,
- $p_s^A(\psi) \leq p$ for some adversary A is equivalent to $\neg\mathcal{P}_{>p}[\psi]$.

The PCTL model checking algorithm for MDPs works in the same way as the one for DTMCs does. It takes a model $\mathcal{M} = (S, \overline{s}, Act, Steps, L)$ and a PCTL formula ϕ as input, and recursively build the set $Sat(\phi) \subseteq S$ of states that satisfy the formula. The Boolean operators are handled in the same way too. But we use a different procedure to deal with $\mathcal{P}_{\bowtie p}[\psi]$, as we need to check whether $p_s(\psi)$ satisfies the bound

$\bowtie p$ for all adversaries A. If $\bowtie$ is $\leq$ or $<$, then we compute the maximum probability for all adversaries and obtain

$$Sat(\mathcal{P}_{\bowtie p}[\psi]) = \{s \in S \mid p_s^{max} \mid (\psi) \bowtie p\}.$$

If $\bowtie$ is $\geq$ or $>$, then we compute the minimum probability for all adversaries and obtain

$$Sat(\mathcal{P}_{\bowtie p}[\psi]) = \{s \in S \mid p_s^{min} \mid (\psi) \bowtie p\}.$$

Now we present how to compute p_s^{max} and p_s^{min} for $\psi = X\ \phi$, $\phi_1\ \mathcal{U}^{\leq k}\phi_2$ and $\phi_1\ \mathcal{U}\phi_2$ [23, 34, 36, 37, 38].

The $\mathcal{P}_{\bowtie p}[X\ \phi]$ operator. For this operator, p_s^{max} and p_s^{min} can be computed directly from function $Steps$.

$$p_s^{max}(X\ \phi) = \max_{(a,\mu)\in Steps(s)} \left\{ \sum_{s'\in Sat(\phi)} \mu(s') \right\},$$

$$p_s^{min}(X\ \phi) = \min_{(a,\mu)\in Steps(s)} \left\{ \sum_{s'\in Sat(\phi)} \mu(s') \right\},$$

The $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}^{\leq k}\ \phi_2]$ operator. Like the DTMC case, we partition S into S^{no}, S^{yes} and $S^?$:

$$S^{no} = S\backslash(Sat(\phi_1) \cup Sat(\phi_2)),\ \ S^{yes} = Sat(\phi_2),\ \ S^? = S\backslash(S^{no} \cup S^{yes}),$$

and compute p_s^{max} and p_s^{min} recursively. S^{no} contains states from which no path satisfies $\phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ under any adversary, and S^{yes} contains states from which all paths satisfy $\phi_1\ \mathcal{U}^{\leq k}\ \phi_2$ under all adversaries.

$$p_s^{max}(\phi_1\ \mathcal{U}^{\leq k}\ \phi_2) = \begin{cases} 0 & \text{if } (s \in S^{no}) \vee (s \in S^? \wedge k = 0) \\ 1 & \text{if } s \in S^{yes} \\ \max\limits_{(a,\mu)\in Steps(s)} \left\{ \sum\limits_{s'\in Sat(\phi)} \mu(s') \cdot p_s^{max}(\phi_1\ \mathcal{U}^{\leq (k-1)}\ \phi_2) \right\} & \text{if } s \in S^? \wedge k > 0, \end{cases}$$

$$p_s^{min}(\phi_1\ \mathcal{U}^{\leq k}\ \phi_2) = \begin{cases} 0 & \text{if } (s \in S^{no}) \vee (s \in S^? \wedge k = 0) \\ 1 & \text{if } s \in S^{yes} \\ \min\limits_{(a,\mu)\in Steps(s)} \left\{ \sum\limits_{s'\in Sat(\phi)} \mu(s') \cdot p_s^{min}(\phi_1\ \mathcal{U}^{\leq (k-1)}\ \phi_2) \right\} & \text{if } s \in S^? \wedge k > 0, \end{cases}$$

The $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}\ \phi_2]$ operator. We partition S into S^{no}, S^{yes} and $S^?$ for this operator too. But we use different procedures to compute these three sets for p_s^{max} and p_s^{min} separately. For $_s^{max}$, we have

$$\begin{array}{rcl} S^{no} & = & Prob0A(Sat(\phi_1), Sat(\phi_2)), \\ S^{yes} & = & Prob1E(Sat(\phi_1), Sat(\phi_2)), \\ S^? & = & S\backslash(S^{yes} \cup S^{no}). \end{array}$$

and for p_s^{min}, we have

$$\begin{array}{rcl} S^{no} & = & Prob0E(Sat(\phi_1), Sat(\phi_2)), \\ S^{yes} & = & Sat(\phi_2), \\ S^? & = & S\backslash(S^{yes} \cup S^{no}). \end{array}$$

Algorithm $Prob0A$ first computes the set of states, each of which either satisfies ϕ_2, or satisfies ϕ_1 and with non-zero probability, can reach a state in $Sat(\phi_2)$ without leaving ϕ_1 under any adversary. Then it returns the complement of this set under S as S^{no}. For each state $s \in S^{no}$, no path in $Path_s^A$ satisfies the formula $\phi_1\ \mathcal{U}\ \phi_2$ with non-zero probability under any adversary.

Algorithm $Prob1E$ computes a double fix point to return the set S^{yes} of states, each of which has $p_s^A(\phi_1\ \mathcal{U}\ \phi_2) = 1$ for some adversary. The outer loop identifies states from which no adversary can make $p_s^A(\phi_1\ \mathcal{U}\ \phi_2) = 1$, and remove those states from S. The inner loop collects states from which one cannot reach a state in $Sat(\phi_2)$ without passing through either a state not in $sat(\phi_1)$ or a state already removed from S.

For p_s^{min}, states in S^{no} have $p_s^A(\phi_1\ \mathcal{U}\ \phi_2) = 0$ for some adversary. Algorithm $Prob0E$ computes a subset of $Sat(\phi_1)$ such that with non-zero probability, each state can reach a state in $Sat(\phi_2)$ without

Algorithm 3 $Prob0A(Sat(\phi_1), Sat(\phi_2))$

$R := Sat(\phi_2)$
$done :=$ **false**
while $done =$ **false do**
  $R' := R \cup \{s \in Sat(\phi_1) \mid \exists (a, \mu) \in Steps(s) \,.\, \exists s' \in R \,.\, \mu(s') > 0\}$
  if $R' = R$ **then**
    $done :=$ **true**
  end if
  $R := R'$
end while
return $S \backslash R$

Algorithm 4 $Prob1E(Sat(\phi_1), Sat(\phi_2))$

$R := S$
$done :=$ **false**
while $done =$ **false do**
  $R' := Sat(\phi_2)$
  $done' :=$ **false**
  while $done' =$ **false do**
    $R'' := R' \cup \{s \in Sat(\phi_1) \mid \exists (a, \mu) \in Steps(s) \,.\, (\forall s' \in S \,.\, \mu(s') > 0 \rightarrow s' \in R) \wedge (\exists s' \in R' \,.\, \mu(s') > 0)\}$
    if $R'' = R'$ **then**
      $done' :=$ **true**
    end if
    $R' := R''$
  end while
  if $R' = R$ **then**
    $done :=$ **true**
  end if
  $R := R'$
end while
return R

Algorithm 5 $Prob0E(Sat(\phi_1), Sat(\phi_2))$

$R := Sat(\phi_2)$
$done :=$ **false**
while $done =$ **false do**
  $R' := R \cup \{s \in Sat(\phi_1) \mid \forall (a, \mu) \in Steps(s) \,.\, \exists s' \in R \,.\, \mu(s') > 0\}$
  if $R' = R$ **then**
    $done :=$ **true**
  end if
  $R := R'$
end while
return $S \backslash R$

passing a state outside the subset. It then returns the complement of the union of the subset and $Sat(\phi_2)$ under S. The set S^{yes} should contain all states such that $p_s^A(\phi_1 \,\mathcal{U}\, \phi_2) = 1$ for every adversary. To simplify the computation, we use the set $Sat(\phi_2)$, which satisfy the condition trivially.

After we obtain S^{no}, S^{yes} and $S^?$, we can compute p_s^{max} and p_s^{min} for $S^?$ by solving a linear optimisation problem over the set of variables $\{x_s \mid s \in S^?\}$. Letting x_s be p_s^{max}, we compute p_s^{max} by minimising $\sum_{s \in S^?} x_s$ under the constraints

$$x_s \geq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s') \quad \text{for all } s \in S^? \text{ and all } (a, \mu) \in Steps(s).$$

Letting x_s be p_s^{min}, we compute p_s^{min} by maximising $\sum_{s \in S^?} x_s$ under the constraints

$$x_s \leq \sum_{s' \in S^?} \mu(s') \cdot x_{s'} + \sum_{s' \in S^{yes}} \mu(s') \quad \text{for all } s \in S^? \text{ and all } (a, \mu) \in Steps(s).$$

3.1.2 Model checking CTMCs

In DTMC and MDP models, probabilistic systems move between their states in discrete time-steps. But there are systems evolving in a continuous way, i.e., a transition in such systems can occur in real-time. A common model for probabilistic continuous time systems are CTMCs (continuous time Markov chains) because they preserve the memoryless property like DTMCs. For CTMCs, the memoryless property not only requires that the probability of firing a transition totally depends on the current state, but also asks the probability to be independent of the elapsed time so far. The only continuous probability distribution exhibiting this property is the exponential distribution, which associates a *rate* to each transition in CTMCs. The rate, denoted by λ, is the only parameter in an exponential distribution

$$f(x; \lambda) = \begin{cases} \lambda \cdot e^{-\lambda \cdot x} & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

The rate can be understood as the average number of times we can execute the transition per unit of time. The probability of executing a transition from the current state within t time units is $1 - e^{-\lambda \cdot t}$. Formally, a CTMC is defined as follows.

Definition 3.6 *A CTMC is a tuple* $(S, \overline{s}, \mathbf{R}, L)$ *where*

- S *is a finite set of* states*,*
- $\overline{s} \in S$ *is the* initial state*,*
- $\mathbf{R} : S \times S \to \mathbb{R}^{\geq 0}$ *is the* transition rate matrix*,*
- $L : S \to 2^{AP}$ *is a* labelling function*.*

Each entry in $\mathbf{R}$ represents a rate between a pair of states. A transition can only occur from state s to state s' if $\mathbf{R}(s, s') > 0$. If more than one transition can be executed in state s, the successor state is determined by the first transition being taken. The amount of time for which the system stays in s before any transition occurs expects an exponential distribution with rate $E(s)$ such that

$$E(s) \stackrel{def}{=} \sum_{s' \in S} \mathbf{R}(s, s').$$

From a CTMC, we can generate a DTMC to show the actual probability of each state s' being the successor state to which a transition is made from state s, independent of the time at which this occurs. This DTMC is referred to as the embedded Markov chain.

Definition 3.7 *The* embedded Markov chain $emb(\mathcal{C})$ *of a CTMC* $\mathcal{C} = (S, \overline{s}, \mathbf{R}, L)$ *is the DTMC* $(S, \overline{s}, \mathbf{P}, L)$ *where for* $s, s' \in S$*:*

$$\mathbf{P}(s, s') = \begin{cases} \mathbf{R}(s, s')/E(s) & \text{if } E(s) \neq 0, \\ 1 & \text{if } E(s) = 0 \text{ and } s = s', \\ 0 & \text{otherwise.} \end{cases}$$

Since a CTMC can have (absorbing) states that do not have outgoing transitions, (in this case $E(s) = 0$), which means the system can stay in those states permanently, we have to add self-loops with probability 1 to the embedded DTMC.

The following matrix is needed when we perform analysis on CTMCs.

Definition 3.8 *The* infinitesimal generator matrix *for the CTMC* $(S, \overline{s}, \mathbf{R}, L)$ *is the matrix* $\mathbf{Q} : S \times S \to \mathbb{R}$ *defined as*

$$\mathbf{Q} : (s, s') = \begin{cases} \mathbf{R}(s, s') & \text{if } s \neq s', \\ -\sum_{s'' \neq s} \mathbf{R}(s, s'') & \text{otherwise.} \end{cases}$$

A path ω in a CTMC is a non-empty sequence $s_0 t_0 s_1 t_1 s_2 \ldots$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t \in \mathbb{R}^{>0}$ for all $i \geq 0$. The value t_i, also denoted by $time(\omega, i)$, represents the amount of time spent in the state s_i. The ith state in ω and the state at time t are denoted by $\omega(i)$ and $\omega@t$ respectively. Let $Path_s$ be the set of all infinite paths starting in state s. The probability measure $Prob_s$ over $Path_s$ is defined as follows. If the states $s_0, \ldots, s_n \in S$ satisfy $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $0 \leq i < n$ and $I_0, \ldots, I_{n-1}$ are non-empty intervals in $\mathbb{R}^{>0}$, then the *cylinder* set $C(s_0, I_0, \ldots, I_{n-1}, s_n)$ is defined to be the set containing all infinite paths $s'_0 t'_0 s'_1 t'_1 s'_2 \ldots$ where $s_i = s'_i$ for $i \leq n$ and $t_i \in I_i$ for $i < n$. Then let Σ_s be the smallest σ-algebra on $Path_s$ which contains all the cylinder sets $C(s_0, I_0, \ldots, I_{n-1}, s_n)$, where $s_0, \ldots, s_n \in S$ range over all sequences of states with $s_0 = s$ and $\mathbf{R}(s_i, s_{i+1}) > 0$ for $0 \leq i < n$, and $I_0, \ldots, I_{n-1}$ ranger over all sequences of non-empty intervals in $\mathbb{R}^{>0}$. The probability measure $Prob_s$ on Σ_s is then the unique measure defined inductively by $Prob_s(C(s_0)) = 1$ and $Prob_s(C(s_0, \ldots, s_n, I_n, s_{n+1}))$ equal to

$$Prob_s\big(C(s_0, \ldots, s_n)\big) \cdot \mathbf{P}(s_n, s_{n+1}) \cdot (e^{-\inf I_n \cdot E(s_n)} - e^{-\sup I_n \cdot E(s_n)})$$

where $\mathbf{P}$ is the transition probability matrix of the CTMC's embedded Markov chain.

In addition to path probabilities, *transient* behaviour and *steady* behaviour are interesting properties for CTMC models. The former considers the state of the model at a particular time instant, while the latter focuses on the state of the model in the long run. The transient probability $\pi_{s,t}(s')$ is the probability of being in state s' at instant t, having started at state s.

$$\pi_{s,t}(s') \stackrel{def}{=} Prob_s(\{\omega \in Path_s \mid \omega@t = s'\}).$$

The steady-state probability $\pi_s(s')$ is the probability of being in state s' in the long run, having started in state s.

$$\pi_s(s') \stackrel{def}{=} \lim_{t \to \infty} \pi_{s,t}(s').$$

The values $\pi_s(s')$ for all $s' \in S$ form the steady-state probability distribution, which can be used to infer the percentage of time that the model spends in each state in the long run.

The behaviour of CTMCs can be specified by CSL (Continuous Stochastic Logic) [16, 19], which extends PCTL on DTMCs and MDPs to include not only path-based behaviour, but also steady-state and transient behaviour.

Definition 3.9 *The syntax of CSL is as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi] \mid, \quad \psi ::= X\ \phi \mid \phi\, \mathcal{U}^I \phi \mid \phi\, \mathcal{U}\ \phi$$

where a *is an atomic proposition,* $\bowtie \in \{\leq, <, \geq, >\}$*,* $p \in [0, 1]$*,* $c \in \mathbb{R}^{\geq 0}$ *and* I *is an interval of* $\mathbb{R}^{\geq 0}$*.*

Most operators in CSL are explained in the same way as in PCTL. The differences are the time bound I in $\phi_1\, \mathcal{U}^I \phi_2$ and the steady-state operator $\mathcal{S}$. The formula $\phi_1\, \mathcal{U}^I \phi_2$ holds in a path if ϕ_1 holds continuous from the beginning until a time instant in I at which ϕ_2 holds. The formula $\mathcal{S}_{\bowtie p}[\phi]$ holds in a state s if the steady-state probability of being in a state where ϕ holds, having started from s, satisfies the bound $\bowtie p$. The formal semantics of CSL is defined as follows.

Definition 3.10 *Let* $\mathcal{C} = (S, \overline{s}, \mathbf{R}, L)$ *be a CTMC. For any state* $s \in S$*, the satisfaction relation* $s \models \phi$ *is defined inductively by:*

- $s \models true$ *for all* $s \in S$,
- $s \models a$ *iff* $a \in L(s)$,
- $s \models \neg\phi$ *iff* $s \not\models \phi$,
- $s \models \phi_1 \wedge \phi_2$ *iff* $s \models \phi_1$ *and* $s \models \phi_2$,
- $s \models \mathcal{P}_{\bowtie p}[\psi]$ *iff* $p_s(\psi) \bowtie p$
- $s \models \mathcal{S}_{\bowtie p}[\psi]$ *iff* $\sum_{s' \models \phi} \pi_s(s') \bowtie p$

where

$$p_s(\psi) \stackrel{def}{=} Prob_s(\{\omega \in Path_s \mid \omega \models \psi\})$$

and for any path $\omega \in Path_s$*:*

- $\omega \models X\ \phi$ *iff* $\omega(1) \models \phi$,
- $\omega \models \phi_1\ \mathcal{U}^I \phi_2$ *iff* $\exists t \in I\ .\ (\omega@t \models \phi_2 \wedge \omega@x \models \phi_1\ \forall x \in [0,t))$,
- $\omega \models \phi_1\ \mathcal{U}\ \phi_2$ *iff* $\exists k \geq 0\ .\ (\omega(k) \models \phi_2 \wedge \omega(j) \models \phi_1\ \forall j < k)$*).*

Note that the transient probability of being in a state satisfying ϕ in time instant t can be specified implicitly by the formula

$$\mathcal{P}_{\bowtie p}[\diamond^{[t,t]}\phi]$$

where

$$\mathcal{P}_{\bowtie p}[\diamond^I \phi] \equiv \mathcal{P}_{\bowtie p}[true\ \mathcal{U}^I \phi].$$

The model checking procedures for CSL operators $true$, a, $\wedge$, and $\neg$ are the same as those for PCTL on DTMCs. Moreover, the $\mathcal{P}_{\bowtie p}[\phi]$ and $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}\ \phi_2]$ operators are computed on the embedded Markov chain of the CTMC using the procedures for the same PCTL operators, as they are not related to continuous time aspects. We only discuss $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}^{\leq k}\ \phi_2]$ and $\mathcal{S}_{\bowtie p}[\phi]$ here.

The $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}^{\leq k}\ \phi_2]$ **operator**. For this operator, we distinguish four cases regarding I. Note that we do not consider the case $I = [0, \infty)$ because

$$p_s(\phi_1\ \mathcal{U}^{[0,\infty)}\ \phi_2) = p_s(\phi_1\ \mathcal{U}\ \phi_2).$$

- $I = [t,t]$ for $t \geq 0$. As mentioned before, this is the transient property. We use the technique *uniformisation* to compute the transient probabilities $\pi_{s,t}(s')$ for all states $s, s' \in S$. Let Π_t be the matrix such that $\Pi_t(s,s') = \pi_{s,t}(s')$. We construct the *uniformised* DTMC for the CTMC with the transition probability matrix **P**:

  $$\mathbf{P} = \mathbf{I} + \mathbf{Q}/q \quad \text{where } q \geq \max_{s \in S} |\mathbf{Q}(s,s)|.$$

  Then we obtain

  $$\Pi_t = \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \mathbf{P}^i \quad \text{where } \gamma_{i,q\cdot t} = e^{-q\cdot t} \cdot (q \cdot t)^i / i!.$$

  Intuitively, each step of the uniformised DTMC is an exponentially distributed delay with parameter q in the CTMC. All entries of **P** are in the range $[0,1]$ and all rows sum to 1. The infinite sum is easy to truncate.

- $I = [0,t]$ for $t \geq 0$. For states satisfying ϕ_2, the probability is 1, and for those satisfying $\neg(\phi_1 \vee \phi_2)$, it is simply 0. For other states, the probability is computed numerically. First, we make states that satisfy $\neg(\phi_1 \vee \phi_2)$ or ϕ_2 absorbing, i.e., no outgoing transitions. The new CTMC for a CTMC $\mathcal{C} = (S, \overline{s}, \mathbf{R}, L)$ is defined as

  $$\mathcal{C}[\phi] \stackrel{def}{=} (S, \overline{s}, \mathbf{R}[\phi], L) \quad \text{where } \mathbf{R}[\phi] = \begin{cases} \mathbf{R}(s,s') & \text{if } s \not\models \phi \\ 0 & \text{otherwise.} \end{cases}$$

Now the probability of the path formula $\phi_1\ \mathcal{U}^{[0,t]}\ \phi_2$ in $\mathcal{C}$ is transformed to the transient probability of being in a state satisfying ϕ_2 at time t in $\mathcal{C}[\neg\phi_1 \vee \phi_2]$, given $\neg(\phi_1 \vee \phi_2) \vee \phi_2$.

$$p_s^{\mathcal{C}}(\phi_1\ \mathcal{U}^{[0,t]}\ \phi_2) = \sum_{s' \models \phi_2} \pi_{s,t}^{\mathcal{C}[\neg\phi_1 \vee \phi_2]}(s')$$

- $I = [t, t']$ for $0 < t \leq t'$. The computation is done in two steps:
  1. Computing the probability of staying in states satisfying ϕ_1 for time t;
  2. Computing the probability of reaching a state satisfying ϕ_2 within time $t' - t$.

  The first step is transformed into transient probabilities in a CTMC $\mathcal{C}[\phi]$, in which ϕ_1 states are made absorbing. The overall probability is done as follows:

$$p_s^{\mathcal{C}}(\phi_1\ \mathcal{U}^{[t,t']}\ \phi_2) = \sum_{s' \models \phi_1} \pi_{s,t}^{\mathcal{C}[\neg\phi_1]} \cdot p_{s'}^{\mathcal{C}}(\phi_1\ \mathcal{U}^{[0,t'-t]}\ \phi_2).$$

- $I = [t, \infty)$ for $t > 0$. Except the unbounded until operator, this case is the same as the previous one.

$$p_s^{\mathcal{C}}(\phi_1\ \mathcal{U}^{[t,\infty)}\ \phi_2) = \sum_{s' \models \phi_1} \pi_{s,t}^{\mathcal{C}[\neg\phi_1]} \cdot p_{s'}^{\mathcal{C}}(\phi_1\ \mathcal{U}\ \phi_2).$$

The $\mathcal{S}_{\bowtie p}[\phi]$ operator. Recall that the formula $\mathcal{S}_{\bowtie p}[\phi]$ holds in state s if $\sum_{s' \models \phi} \pi_s(s') \bowtie p$, which means we have to compute the steady-state probability $\pi_s(s')$ for all states s and s'. Let $\pi(s')$ denote the steady-state probability of being in state s', and $\vec{\pi}$ the vector of all such values for $s' \in S$. For irreducible CTMCs, the probabilities can be obtained by solving a linear equation system

$$\vec{\pi} \cdot \mathbf{Q} = \vec{0} \quad \text{and} \quad \sum_{s' \in S} \pi(s') = 1.$$

For reducible CTMCs, we compute *bottom strongly connected components* (BSCC). In each BSCC, there exits a path between every pair of states, and is no transition in the CTMC that leaves the component. In fact, a BSCC B can be seen as an irreducible CTMC. We can compute $\pi^B(s')$ for each $s' \in B$. If B contains only one state, $\pi^B(s') = 1$. Now we need to compute the probability of reaching each BSCC B from state s. Let a_B be an atomic proposition that holds only in states in B. The probability above can be formalised as $p_s(\diamond a_B)$. Since $\mathcal{P}_{\bowtie p}[\diamond a_B] \equiv \mathcal{P}_{\bowtie p}[true\ \mathcal{U}\ a_B]$, which can be computed by a linear equation system, we obtain

$$\pi_s(s') = \begin{cases} p_s(\diamond a_B) \cdot \pi^B(s') & \text{if } s' \text{ is in a BSCC B} \\ 0 & \text{otherwise.} \end{cases}$$

3.1.3 Model checking PTAs

Probabilistic Timed Automata (PTAs) [49] are a probabilistic model combining probabilistic choice, nondeterminism and dense time. Unlike continuous distributions in CTMCs, transitions in PTAs have discrete probabilistic distributions as in the case of DTMC and MDP. Indeed, PTAs are an extension of the well-known real-time model *Timed Automata* (TAs). PTAs can be used in many scenarios. An example is a randomised distributed protocol with time delays such as IEEE 802.11 Wireless LAN MAC protocol. In TAs, time is measured by dense time clocks, which run at the same speed. The reading of clocks is a non-negative real number. Let $\mathcal{X}$ be a finite set of clocks, and $x \in \mathcal{X}$ a clock. A *clock valuation* ν assigns a value to each clock in $\mathcal{X}$. Let $\nu(x)$ be the value of clock x in ν, $\nu + t$ the clock valuation which increments all clock values in ν by t, and $\mathbb{R}^{\mathcal{X}}$ the set of all clock valuations. By slightly misusing the notation, we say that x is a variable representing the reading of a clock in $\mathcal{X}$. A clock *zone* is a set of clock valuations that can be specified by a conjunction of propositional constraints over clock variables, such as $(x_1 < 5) \wedge (x_2 \geq 1)$. Formally, given the set of clocks $\mathcal{X}$, a zone[1] ζ over $\mathcal{X}$ is defined as follows:

$$\zeta ::= true \mid x \sim c \mid x - y \sim c \mid \zeta \wedge \zeta$$

[1] In this report, we only consider *diagonal-free* and *closed* zones. The former excludes constraints of the form $x - y \sim c$, and the latter excludes the cases $\sim = <$ or $>$.

where $x, y \in \mathcal{X}$, $c \in \mathbb{N}$ and $\sim \in \{<, \leq, >, \geq\}$. For a subset $X \subseteq \mathcal{X}$ of clocks, let $\nu[X := 0]$ the clock valuation obtained from ν by set the value of every clock in X to zero, i.e.,

$$\nu[X := 0](x) = \begin{cases} 0 & \text{if } x \in X, \\ \nu(x) & \text{otherwise.} \end{cases}$$

Given a zone ζ, the zone $\nearrow \zeta$ contains all clock valuations that can be reached from a valuation in ζ by letting time pass. Conversely, $\swarrow \zeta$ contains those that can reach ζ by letting time pass. Let $\zeta[X := 0]$ be the zone obtained by set the variables in X to zero in every valuation in ζ, i.e., $\zeta[X := 0] = \{\nu[X := 0] \mid \nu \in \zeta\}$. The clock valuation ν satisfies zone ζ, denoted $\nu \triangleleft \zeta$, if and only if ζ resolves to true after substituting each clock $x \in \mathcal{X}$ with the corresponding clock value from ν. A zone is *satisfiable* if there exists a valuation that satisfies the zone. Let $Zones(\mathcal{X})$ be the set of all satisfiable zones over $\mathcal{X}$.

Definition 3.11 *A* probabilistic timed automaton *is a tuple* $\mathcal{T} = (Loc, \bar{l}, Act, inv, enab, prob)$ *where:*

- Loc *is a finite set of locations;*
- $\bar{l} \in Loc$ *is the* initial location*.*
- Act *is a finite set of* actions*;*
- $inv : Loc \to Zones(\mathcal{X})$ *is the* invariant condition*;*
- $enab : Loc \times Act \to Zones(\mathcal{X})$ *is the* enabling condition*;*
- $prob : Loc \times Act \to Dist(2^{\mathcal{X}} \times Loc)$ *is the* probabilistic transition function*.*

A *state* of a PTA is a pair $(l, \nu) \in Loc \times \mathbb{R}^{\mathcal{X}}$ such that $\nu \triangleleft inv(l)$. The initial state is $(\bar{l}, \mathbf{0})$, where all clocks are set to 0. In any state (l, ν), a certain amount of time $t \in \mathbb{R}$ can elapse, after which an action $a \in Act$ is performed. The choice of t requires that, while time passes, the invariant $inv(l)$ remains continuously satisfied. Each action a can be only chosen if it is *enabled*, i.e., the zone $enab(l, a)$ is satisfied by $\nu + t$. Once action a is chosen, a set of clocks to reset and successor location are selected at random, according to the distribution $prob(l, a)$. We call each element $(X, l') \in 2^{\mathcal{X}} \times Loc$ in the support of $prob(l, a)$ an *edge* and, for convenience, assume that the set of such edges, denoted $edges(l, a)$, is an ordered list $\langle e_1, \ldots, e_n \rangle$.

Formally, the semantics of a PTA is defined as an (infinite-state) Markov decision process[2].

Definition 3.12 *Let* $\mathcal{T} = (Loc, \bar{l}, Act, inv, enab, prob)$ *be a PTA. The semantics of* $\mathcal{T}$ *is defined as the (infinite-state) MDP* $[\![\mathcal{T}]\!] = (S, \bar{s}, \mathbb{R} \times Act, Steps_{\mathcal{T}})$ *where:*

- $S = \{(l, \nu) \in Loc \times \mathbb{R}^{\mathcal{X}} \mid \nu \triangleleft inv(l)\}$ *and* $\bar{s} = (\bar{l}, \mathbf{0})$*;*
- $Steps_{\mathcal{T}}((l, \nu), (t, a)) = \lambda$ *iff* $v + t' \triangleleft inv(l)$ *for all* $0 \leq t' \leq t$, $\nu + t \triangleleft enab(l, a)$ *and, for any* $(l', \nu') \in S$*:*

$$\lambda(l', \nu') = \sum \Big\{ prob(l, a)(X, l') \mid X \in 2^{\mathcal{X}} \wedge \nu' = (\nu + t)[X := 0] \Big\}.$$

Each transition of the semantics of the PTA is a time-action pair (t, a), representing time passing for t time units, followed by a discrete a-labelled transition. If $Steps_{\mathcal{T}}((l, v), (t, a))$ is defined and $edges(l, a) = \langle (X_1, l_1), \ldots, (X_n, l_n) \rangle$, we write $(l, v) \xrightarrow{t,a} \langle s_1, \ldots, s_n \rangle$ where $s_i = (l_i, (v + t)[X_i := 0])$ for all $1 \leq i \leq n$. Recall that for a fixed adversary A, we can define a probability measure over the set of paths from a state s and, in particular, the probability $p_s^A(F)$ of reaching a target $F \subseteq S$ from s under A. We are typically interested in the *minimum* and *maximum* reachability probabilities for F:

$$p_{\mathcal{M}}^{\min}(F) \stackrel{def}{=} \inf_A p_{\bar{s}}^A(F) \text{ and } p_{\mathcal{M}}^{\max}(F) \stackrel{def}{=} \sup_A p_{\bar{s}}^A(F).$$

In many cases, it is intuitive to give system specification in terms of a parallel composition of components. A component can change locations independently or synchronise with other component to make a joint move. In this way, we avoid explicitly specifying a large single PTA representing the overall behaviour of the system. Instead, The single PTA is computed as the product of component automata.

[2]Here we do not consider the labels.

Definition 3.13 *Let $\delta_{(X,l)}$ be a point distribution for a location l and a set of resetting clocks $X \subseteq \mathcal{X}$, and $\mathcal{T}_i = (Loc_i, \bar{l}_i, Act_i, inv_i, enab_i, prob_i)$ for $i = 1, 2$ such that $\mathcal{X}_i \cap \mathcal{X}_2 = \emptyset$. The* parallel composition *of two probabilistic time automata $\mathcal{T}_1$ and $\mathcal{T}_2$ is the PTA $\mathcal{T}_1 \parallel T_2 = (Loc_1 \times Loc_2, (\bar{l}_1, \bar{l}_2), \mathcal{X}_1 \cup \mathcal{X}_2, Act_1 \cup Act_2, inv, enab, prob)$ where $inv(l, l') = inv_1(l) \wedge inv_2(l')$ for all $(l, l') \in Loc_1 \times Loc_2$. The enabling condition $enab$ and the probabilistic transition function $prob$ is determined as follows:*

- *For $a \in Act_1 \backslash Act_2$ and $(l_1, a, p_1) \in prob_1$ such that $p = p_1 \otimes \delta_{(\emptyset, l_2)}$, we have $enab\big((l_1, l_2), a\big) = enab_1(l_1, a)$ and $\big((l_1, l_2), a, p\big) \in prob$;*
- *For $a \in Act_2 \backslash Act_1$ and $(l_2, a, p_2) \in prob_2$ such that $p = \delta_{(\emptyset, l_1)} \otimes p_2$, we have $enab\big((l_1, l_2), a\big) = enab_2(l_2, a)$ and $\big((l_1, l_2), a, p\big) \in prob$;*
- *For $a \in Act_1 \cap Act_2$, $(l_1, a, p_1) \in prob_1$ and $(l_2, a, p_2) \in prob_2$ such that $p = p_1 \otimes p_2$, we have $enab\big((l_1, l_2), a\big) = enab_1(l_1, a) \wedge enab_2(l_2, a)$ and $\big((l_1, l_2), a, p\big) \in prob$;*

where for any $l_1 \in Loc_1$, $l_2 \in Loc_2$, $X \subseteq \mathcal{X}_1$ and $X_2 \subseteq \mathcal{X}_2$:

$$p_1 \otimes p_2\big(X_1 \cup X_2, (l_1, l_2)\big) = p_1(X_1, l_1) \cdot p_2(X_2, l_2).$$

Properties in PTAs can be specified by the probabilistic real-time logic PTCTL (Probabilistic Time Computation Tree Logic). One approach to model check PTAs against PTCTL formulae is based on the *region graph*. However, the number of states of the region graph is exponential in the number of clocks and size of constants, which makes verification prohibitively expensive. Another way is to convert dense time clocks into digital clocks to obtain a more amendable (but less expressive) model. In practise, many PTCTL properties can be dealt with reachability analysis, which can be performed efficiently without losing expressive power. In the rest of this section, we will present this technique, which uses stochastic game semantics [48].

Probabilistic Reachability. The minimum and maximum probabilities of reaching, from the initial state of a PTA $\mathcal{T}$, a certain target $F \subseteq Loc$ are:

$$p^{\min}_{\mathcal{T}}(F) = p^{\min}_{[\![\mathcal{T}]\!]}(S_F) \text{ and } p^{\max}_{\mathcal{T}}(F) = p^{\max}_{[\![\mathcal{T}]\!]}(S_F)$$

where $S_F = \{(l, v) \mid v \triangleleft inv(l) \wedge l \in F\}$.

Symbolic states and operations. In order to represent sets of PTA states, we use the concept of a *symbolic state*: a pair $\mathbf{z} = (l, \zeta)$, comprising a location l and a zone ζ over $\mathcal{X}$, representing the set of PTA states $\{(l, v) \mid v \triangleleft \zeta\}$. We use the notation $(l, v) \in (l, \zeta)$ to denote inclusion of a PTA state in a symbolic state. We will use the *time successor* and *discrete successor* operations of [42, 68]. For a symbolic state (l, ζ), action a, and edge $e = (X, l') \in edges(l, a)$, we define:

- $tsuc(l, \zeta) \stackrel{def}{=} (l, inv(l) \wedge \nearrow \zeta$ is the *time successor* of (l, ζ);
- $dsuc[a, e](l, \zeta) \stackrel{def}{=} (l', (\zeta \wedge enab(l, a))[X := 0] \wedge inv(l'))$ is the *discrete successor* of (l, ζ) with respect to e;
- $post[a, e](l, \zeta) \stackrel{def}{=} tsuc(dsuc[a, e](l, \zeta))$ is the *post* of (l, ζ) with respect to e.

The **c-closure** of a zone ζ is obtained by removing any constraint that refers to integers greater than c. For a given c, there are only a finite number of c-closed zones. For the remainder of this paper, we assume that all zones are c-closed where c is the largest constant appearing in the PTA.

Automated Abstraction-Refinement Using Stochastic Games. *Stochastic two-player* games [64, 32] extend MDPs by allowing two types of nondeterministic choice, controlled by separate players. We use stochastic games in the manner proposed in [47] to represent an abstraction of an MDP.

Definition 3.14 *A stochastic game $\mathtt{G}$ is a tuple $(S, \overline{s}, Act, Steps_{\mathtt{G}})$ where: S is a set of states, $\overline{s} \in S$ is the initial states, Act is a set of actions and $Steps_{\mathtt{G}} : S \times Act \to 2^{Dist(S)}$ is the probabilistic transition function.*

Each transition of a stochastic game $\mathtt{G}$ comprises three choices: first, like for an MDP, player 1 picks an available action $a \in Act$; next, player 2 selects a distribution λ from the set $Steps_{\mathtt{G}}(s,a)$; finally, a successor state is chosen at random according to λ. A resolution of the nondeterminism in $\mathtt{G}$ (the analogue of an MDP adversary) is a pair of *strategies* σ_1, σ_2 for the players, under which we can define the probability $p_s^{\sigma_1,\sigma_2}(F)$ of reaching a target $F \subseteq S$ from a state s.

Intuitively, the idea of [47] is that, in a stochastic game $\mathtt{G}$, representing an abstraction of an MDP $\mathcal{M}$, player 2 choices represent nondeterminism present in $\mathcal{M}$ and player 1 choices represent additional nondeterminism introduced through abstraction. By quantifying over strategies for players 1 and 2, we can obtain both lower bounds (lb) and upper bounds (ub) on the minimum and maximum reachability probabilities of $\mathcal{M}$. If $\mathtt{G}$ is constructed from $\mathcal{M}$ using the approach of [47], then, in the case of maximum probabilities, for example:

$$p_{\mathtt{G}}^{lb,\max}(F) \leq p_{\mathcal{M}}^{\max}(F) \leq p_{\mathtt{G}}^{ub,\max}(F)$$

where, in the stochastic game $\mathtt{G}$:

$$p_{\mathtt{G}}^{lb,\max}(F) \stackrel{def}{=} \inf_{\sigma_1} \sup_{\sigma_2} p_{\bar{s}}^{\sigma_1,\sigma_2}(F)$$

$$p_{\mathtt{G}}^{ub,\max}(F) \stackrel{def}{=} \sup_{\sigma_1} \sup_{\sigma_2} p_{\bar{s}}^{\sigma_1,\sigma_2}(F)$$

Using similar techniques as those for MDPs, we can efficiently compute these values and strategies for players 1 and 2 that result in them [32].

We now describe how to, from a reachability graph, construct a stochastic game $\mathtt{G}$ that yields both lower and upper bounds. The game G is an abstraction of the infinite-state MDP semantics of the PTA, whose state space is the symbolic states $\mathtt{Z}$.

A reachability graph captures information about the transitions in a PTA. It comprises $\mathtt{Z}$ and a set $\mathtt{R} \subseteq \mathtt{Z} \times Act \times \mathtt{Z}^+$ of *symbolic transitions*. Each symbolic transition $\theta \in \mathtt{R}$ takes the form:

$$\theta = \big((l,\zeta), a, \langle (l_1,\zeta_1), \ldots, (l_n,\zeta_n)\rangle\big)$$

where $n = |edges(l,a)|$. Intuitively, θ represents the possibility of taking action a from a PTA state in (l,ζ) and, for each edge $(X_i, l_i) \in edges(l,a)$, reaching a state in (l_i,ζ_i). A key property of symbolic transitions is the notion of *validity*:

$$valid(\theta) \stackrel{def}{=} \zeta \wedge \swarrow \big(enab(l,a) \wedge (\bigwedge_{i=1}^{n} ([X_i := 0]\zeta_i))\big)$$

which gives precisely the set of clock valuations satisfying ζ from which it is possible to let time pass and perform the action a such that taking the ith edge (X_i, l_i) gives a state in (l_i,ζ_i). A symbolic transition θ is valid if the zone $valid(\theta)$ is non-empty. This leads to the following formal definition of a reachability graph.

Definition 3.15 *A* reachability graph *for a PTA* $\mathcal{T} = (Loc, \bar{l}, Act, inv, enab, prob)$ *and target* F*, is a pair* $(\mathtt{Z}, \mathtt{R})$ *where:*

1. $\mathtt{Z} \subseteq Loc \times Zones(\mathcal{X})$ *is a multiset of symbolic states where* $\{s \in z \mid z \in \mathtt{Z} = S$*;*
2. $\mathtt{R} \subseteq \mathtt{Z} \times Act \times \mathtt{Z}^+$ *is a set of valid symbolic transitions;*

and, if $z = (l,\zeta) \in \mathtt{Z}$*,* $l \notin F$*,* $s \in z$ *and* $s \xrightarrow{t,a} \langle s_1, \ldots, s_n\rangle$*, then* $\mathtt{R}$ *contains a symbolic transition* $(z, a, \langle z_1, \ldots, z_n\rangle)$ *such that* $s_i \in z_i$ *for all* $1 \leq i \leq n$*.*

We utilise the approach of [47] to represent an abstraction of an MDP as a stochastic two-player game. The basic idea is that the two players in the game represent nondeterminism introduced by the abstraction and nondeterminism from the original model. In a symbolic state (l,ζ) of the game abstraction of a PTA, player 1 first picks a PTA state $(l,v) \in (l,\zeta)$ and then player 2 makes a choice over the actions that become enabled after letting time pass from (l,v).

For any symbolic state $(l,\zeta) \in \mathtt{Z}$ and set of symbolic transitions $\Theta \subseteq \mathtt{R}(l,\zeta)$, let:

$$valid(\theta) \stackrel{def}{=} (\bigwedge_{\theta \in \Theta} valid(\theta)) \wedge (\bigwedge_{\theta \in \mathtt{R}(l,\zeta) \setminus \Theta} \neg valid(\theta).$$

By construction, $valid(\theta)$ identifies precisely the clock valuations $v \triangleleft \zeta$ such that, from (l, v), it is possible to perform a transition encoded by any symbolic transition $\theta \in \Theta$, but it is not possible to perform a transition encoded by any other symbolic transition of $\mathtt{R}(l, \zeta)$.

The algorithm `BuildGame` in Algorithm 6 describes how to construct, from a reachability graph R, a stochastic game with symbolic states Z. In a state z of the game, player 1 chooses between any non-empty *valid* set of symbolic transitions $\Theta \subseteq \mathtt{R}(z)$. Player 2 then selects a symbolic transition $\theta \in \Theta$. As the following result demonstrates, this game yields lower and upper bounds on either minimum or maximum reachability probabilities of the PTA.

Algorithm 6 `BuildGame(Z, R)`

1: $\overline{\mathtt{z}} := (\bar{l}, \zeta) \in \mathtt{Z}$
2: **for all** $(l, \zeta) \in \mathtt{Z}$ **do**
3: **for all** $\Theta \subseteq \mathtt{R}(l, \zeta)$ **such that** $\Theta \neq \emptyset$ **and** $valid(\Theta)$ **do**
4: $Steps_{\mathtt{G}}((l, \zeta), \Theta) := \{\lambda_\theta \mid \theta \in \Theta\}$
5: **end for**
6: **end for**
7: **return** $\mathtt{G} = (\mathtt{Z}, \overline{z}, 2^{\mathtt{R}}, Steps_{\mathtt{G}})$

Theorem 3.16 *Let $\mathcal{T}$ be a PTA with target F. If $(\mathtt{Z}, \mathtt{R})$ is a reachability graph for $(\mathcal{T}, F)$ and G is the stochastic game returned by* `BuildGame(Z, R)` *(see Algorithm 6), then $p^{lb,*}_{\mathtt{G}}(\mathtt{Z}_F) \leq p^{*}_{\mathcal{T}}(F) \leq p^{ub,*}_{\mathtt{G}}(\mathtt{Z}_F)$ for $* \in \{\min, \max\}$.*

The game-based abstraction approach of [47] has been extended with *refinement* techniques in [45, 46]. Inspired by non-probabilistic counterexample-guided abstraction refinement, the idea is that an initially coarse abstraction is iteratively refined until it is precise enough to yield useful verification results. Crucial to this approach is the use of the lower and upper bounds provided by a stochastic game abstraction as a *quantitative measure* of the preciseness of the abstraction.

The refinement algorithm. Our refinement algorithm takes a reachability graph $(\mathtt{Z}, \mathtt{R})$, splits one or more of the symbolic states in Z and then modifies the symbolic transitions of R accordingly. This process is guided by the analysis of the stochastic game constructed from $(\mathtt{Z}, \mathtt{R})$, i.e., the bounds for the probability of reaching the target and player 1 strategies that attain these bounds.

We now outline the refinement of a single symbolic state (l, ζ) for which the bounds differ and for which distinct player 1 strategies yield each bound. A player 1 strategy chooses, for any state in the stochastic game, an action available in the state. By construction, an available action in (l, ζ) is a valid set of symbolic transitions from $\mathtt{R}(l, \zeta)$. We let $\Theta_{lb}, \Theta_{ub} in \mathtt{R}(l, \zeta)$ denote the distinct player 1 strategy choices for the lower and upper bound respectively. Since the validity conditions for Θ_{lb} and Θ_{ub} identify precisely the clock valuations in ζ for which the corresponding transitions of $[\![\mathcal{T}]\!]$ are possible, we split (l, ζ) into:

$$\big(l, valid(\Theta_{lb})\big), \big(l, valid(\Theta_{ub})\big) \text{ and } \big(l, \zeta \wedge \neg(valid(\Theta_{lb}) \vee valid(\Theta_{ub}))\big).$$

By construction, $valid(\Theta_{lb})$ and $valid(\Theta_{ub})$ are both non-empty. Furthermore, since $\Theta_{lb} \neq \Theta_{ub}$, from the definition of validity, we have $valid(\Theta_{lb}) \wedge valid(\Theta_{ub}) = \emptyset$; and hence the split of (l, ζ) produces a strict refinement of Z. The complete refinement algorithm is shown in Algorithm 7. Lines 1-4 refine Z, as just described, and lines 5-15 update the set of symbolic transitions R. The result is a new reachability graph, for which the corresponding stochastic game is a refined abstraction of the PTA, satisfying the following properties.

Theorem 3.17 *Let $\mathcal{T}$ be a PTA with target F and $(\mathtt{Z}, \mathtt{R})$ be a reachability graph for $(\mathcal{T}, F)$. If $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ is the result of applying algorithm* `Refine` *(see Algorithm 7) to $(\mathtt{Z}, \mathtt{R})$, $\mathtt{G} =$* `BuildGame`$(\mathtt{Z}, \mathtt{R})$ *and $\mathtt{G}^{ref} =$* `BuildGame`$(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$*, then:*

1. *$(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$ is a reachability graph for $(\mathcal{T}, F)$;*
2. *$p^{lb,\star}_{ttG}(\mathtt{Z}_F) \leq p^{lb,\star}_{\mathtt{G}^{ref}}(\mathtt{Z}_F)$ and $p^{ub,\star}_{\mathtt{G}^{ref}}(\mathtt{Z}_F)(\mathtt{Z}_F) \leq p^{ub,\star}_{\mathtt{G}}(\mathtt{Z}_F)$ for $\star \in \{\min, \max\}$.*

Algorithm 7 $\texttt{Refine}(\mathtt{Z}, \mathtt{R}, (l,\zeta), \Theta_{lb}, \Theta_{ub})$

$\zeta_{lb} := valid(\Theta_{lb})$
$\zeta_{ub} := valid(\Theta_{ub})$
$\mathtt{Z}^{new} := \{(l,\zeta_{lb}), (l,\zeta_{ub}), (l,\zeta \wedge \neg(\zeta_{lb} \vee \zeta_{ub}))\}\backslash\{\emptyset\}$
$\mathtt{Z}^{ref} := (\mathtt{Z}\backslash\{(l,\zeta)\}) \uplus \mathtt{Z}^{new}$
$\mathtt{R}^{ref} := \emptyset$
for all $\theta = (z_0, a, \langle z_1, \ldots, z_n\rangle) \in \mathtt{R}$ **do**
  if $(l,\zeta) \notin \{z_0, z_1, \ldots, z_n\}$ **then**
    $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta\}$
  else
    $\Theta^{new} := \{(z'_0, a, \langle z'_1, \ldots, z'_n\rangle) \mid z'_i \in \mathtt{Z}^{new}$ if $z_i = (l,\zeta)$ and $z'_i = z_i$ otherwise$\}$
    for all $\theta^{new} \in \Theta^{new}$ **such that** $valid(\theta^{new}) \neq \emptyset$ **do**
      $\mathtt{R}^{ref} := \mathtt{R}^{ref} \cup \{\theta^{new}\}$
    end for
  end if
end for
return $(\mathtt{Z}^{ref}, \mathtt{R}^{ref})$

This refinement scheme, applied in a iterative manner, provides a way of computing exact values for minimum or maximum reachability probabilities of a PTA. This algorithm, outlined in Algorithm 8, starts with the reachability graph constructed through forwards reachability and then repeatedly:

1. builds a stochastic game;
2. solves the game to obtain lower and upper bounds;
3. refines the reachability graph, based on an analysis of the game.

The iterative process terminates when the difference between the bounds falls below a given level of precision ε. In fact, as the following result states, this process is guaranteed to terminate, in a finite number of steps, with the precise answer.

Algorithm 8 $\texttt{AbstractRefine}(\mathcal{T}, F, \star, \varepsilon)$

$(\mathtt{Z}, \mathtt{R}) := \texttt{BuildReachGraph}(\mathcal{T}, F)$
$\mathtt{G} := \texttt{BuildGame}(\mathtt{Z}, \mathtt{R})$
$(p_{\mathtt{G}}^{lb,\star}, p_{\mathtt{G}}^{ub,\star}, \sigma_1^{lb}, \sigma_1^{ub} := \texttt{AnalyseGame}(\mathtt{G}, F, \star)$
while $p_{\mathtt{G}}^{ub,star} - p_{ttG}^{lb,\star} > \varepsilon$ **do**
  $\mathbf{choose}(l,\zeta) \in \mathtt{Z}$
  $(\mathtt{Z}, \mathtt{R}) := \texttt{Refine}(\mathtt{Z}, \mathtt{R}, (l,\zeta), \sigma_1^{lb}(l,\zeta), \sigma_1^{ub}(l,\zeta))$
  $\mathtt{G} := \texttt{BuildGame}(\mathtt{Z}, \mathtt{R})$
  $(p_{\mathtt{G}}^{lb,\star}, p_{\mathtt{G}}^{ub,\star}, \sigma_1^{lb}, \sigma_1^{ub} := \texttt{AnalyseGame}(\mathtt{G}, F, \star)$
end while

Theorem 3.18 *Let $\mathcal{T}$ be a PTA with target F and $\star \in \{\min, \max\}$. The algorithm* $\texttt{AbstractRefine}(\mathcal{T}, F, \star, \mathbf{0})$ *(see Algorithm 8) terminates after a finite number of steps and returns $[p_{\mathtt{G}}^{lb,\star}, p_{\mathtt{G}}^{ub,\star}]$ where $p_{\mathtt{G}}^{lb,\star} = p_{\mathcal{T}}^{\star}(F) = p_{\mathtt{G}}^{ub,\star}$.*

3.2 Verification of non-functional requirements

Non-functional requirements/properties are of particular importance for CONNECTing heterogeneous networked systems. A simple example of such a property could be the cost of CONNECTors being selected. Given a set of CONNECTors that have the same functionality, but different costs, users might prefer to

Algorithm 9 `BuildReachGraph`$(\mathcal{T}, F)$

```
1: Z := ∅
2: Y := {tsuc(l̄, 0)}
3: while Y ≠ ∅ do
4:    choose(l, ζ) ∈ Y
5:    Y := Y\{(l, ζ)}
6:    Z := Z ∪ {(l, ζ)}
7:    for all a ∈ Act such that enab(l, a) ∧ ζ ≠ ∅ do
8:       for all e_i ∈ edges(l, a) = ⟨e_1, ..., e_n⟩ do
9:          (l'_i, ζ'_i) := post[(l, a), e_i](l, ζ
10:         if (l'_i, ζ'_i) ∉ Z and l'_i ∈ F then
11:            Y := Y ∪ {(l'_i, ζ_i)}
12:         end if
13:      end for
14:   end for
15: end while
16: return (Z, R)
```

choose the one that guarantees the reliability with minimum cost. Many non-functional properties are either instantaneous or cumulative properties. The former means the expected value of the property at some time point, e.g., the expected number of message delivered by the message warning server after exactly 90 seconds; while the latter means the expected cumulated value over some period, e.g., the expected time for the client registration process to terminate. Usually, non-functional properties are handled by extra labels in probabilistic models. In this section, we present the techniques to verify non-functional properties in DTMCs, MDPs and CTMCs.

The sets of labels used to handle non-functional requirements are often known as *cost structures* (also *reward structures*). Adding a cost structure to a DTMC, we obtain a labelled DTMC.

Definition 3.19 *A* labelled *DTMC is a tuple* $(\mathcal{D}, C)$ *where*

- $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ *is a DTMC;*
- $\mathbf{C} : S \times S \to \mathbb{R}^{\geq 0}$ *is a* cost structure*.*

The cost structure assigns every transition a real number. In Section 3.1.1, we have defined the probability measure $Prob_s$ for paths in DTMCs. Given a target set F of states, we can use $Prob_s$ to define the cost $cost(F)(\omega)$ of reaching a state in F along a path $\omega \in Path_s$.

$$cost(F)(\omega) \stackrel{def}{=} \begin{cases} \sum_{i=1}^{\{j|\omega(j)\in F\}} \mathbf{C}\big(\omega(i-1), \omega(i)\big) & \text{if } \exists j \in \mathbb{N} \,.\, \omega(j) \in F \\ \infty & \text{otherwise} \end{cases}$$

Let $E_s(cost(F))$ be the expected cost from s to states in F. It is the expectation of the cost of reaching any state in F via any path in $Path_s$. Since the cost is defined to be ∞ if a path does not pass any target state, the expected cost would be ∞ if the states in F cannot be reached from s with probability 1.

To verify nonfunctional requirements, PCTL has been extended to include an operator to reason about expected cost $\mathcal{E}$.

Definition 3.20 *The syntax of extended PCTL is as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{E}_{\bowtie c}[\phi], \;\; \psi ::= X\ \phi \mid \phi\, \mathcal{U}^{\leq k}\ \phi \mid \phi\, \mathcal{U}\ \phi$$

where a *is an atomic proposition, and* $\bowtie \in \{\leq, <, \geq, >\}$*,* $p \in [0, 1]$*,* $c \in \mathbb{R}^{\geq 0}$ *and* $k \in \mathbb{N}$*.*

The formula $\mathcal{E}_{\bowtie c}[\phi]$ holds in state s if starting from s, the expected cost to reach a state satisfying ϕ satisfies the condition $\bowtie c$. Formally,

$$s \models \mathcal{E}_{\bowtie c}[\phi] \;\text{ iff } e_s(\phi) \bowtie c$$

where $e_s(\phi) \stackrel{def}{=} E_s\big(cost(\{s' \in S \mid s' \models \phi)\big)$. Model checking procedure $Sat(\mathcal{E}_{\bowtie c}[\phi])$ returns the set of states, each of which satisfies the condition $\bowtie c$ with respect to the expected cost of reaching ϕ states, i.e.,

$$Sat(\mathcal{E}_{\bowtie c}[\phi]) = \{s \in S \mid e_s(\phi) \bowtie c\}.$$

To compute the expected cost for each state in S, we partition S into three sets S^0, S^∞ and $S^?$. States in S^0 have the expected cost 0, which simply means these states satisfy ϕ. States in S^∞ have the expected cost ∞, i.e., they cannot reach the ϕ states with probability 1. This set is computed using Algorithms $Prob1$ and $Prob0$ defined in Section 3.1.1.

$$\begin{aligned} S^0 &= Sat(\phi), \\ S^\infty &= S \backslash Prob1\Big(S, Sat(\phi), Prob0\big(S, Sat(\phi)\big)\Big), \\ S^? &= S \backslash (S^0 \cup S^\infty). \end{aligned}$$

Computing $e_s(\phi)$ for states in $S^?$ is done by letting $x_s = e_s(\phi)$ for all $s \in S^?$ and solving the linear equation system:

$$x_s = \sum_{s \in S^?} \mathbf{P}(s, s') \cdot \big(\mathbf{C}(s, s') + x_{s'}\big).$$

The cost structure in MDP is slightly different from the one in DTMC. We name it *cost function*.

Definition 3.21 *A labelled MDP is a tuple* $(\mathcal{M}, \mathbf{c})$ *where*

- $\mathcal{M}$ *is an MDP,*
- $\mathbf{c} : S \to \mathbb{R}^{\geq 0}$ *is the cost function.*

The cost function gives each pair of state and action a cost. In this case, all transitions labelled with the same action in a probability distribution have the same cost. Given a adversary A with the measure $Prob_S^A$, the expected cost of reaching a set F of target states, $E_s^A\big(cost(F)\big)$, is computed based on the cost of paths $\omega \in Path_s^A$:

$$cost(F)(\omega) \stackrel{def}{=} \begin{cases} \displaystyle\sum_{i=0}^{\min\{i \mid \omega(i) \in F\}} \mathbf{c}\big(step(\omega, i-1)\big) & \text{if } \exists i \in \mathbb{N} \,.\, \omega(i) \in F \\ \infty & \text{otherwise.} \end{cases}$$

The extended PCTL for DTMC can be used to reason about cost-related properties in MDP as well. However, the semantics of $\mathcal{E}_{\bowtie c}$ is different: $\mathcal{E}_{\bowtie c}[\phi]$ holds in state s if and only if the expected cost of reaching the ϕ states from s is satisfied for $\bowtie c$ for all adversaries. We have

$$s \models \mathcal{E}_{\bowtie c}[\phi] \text{ iff } e_s^A(\phi) \bowtie c \text{ if for all } A \in Adv_{\mathcal{M}},$$

where $e_s^A(\phi) \stackrel{def}{=} E_s^A\big(cost(\{s' \in S \mid s' \models \phi)\big)$ for any adversary $A \in Adv_{\mathcal{M}}$.

To model check $\mathcal{E}_{\bowtie c}[\phi]$, we need to deal with $\{<, \leq\}$ and $\{>, \geq\}$ separately. For $\bowtie$ is $<$ or $\leq$,

$$Sat(\mathcal{E}_{\bowtie c}[\phi]) = \{s \in S \mid e_s^{\max}(\phi) \bowtie c\};$$

For $\bowtie$ is $>$ or $\geq$,

$$Sat(\mathcal{E}_{\bowtie c}[\phi]) = \{s \in S \mid e_s^{\min}(\phi) \bowtie c\}.$$

The technique to compute $e_s^{\max}(\phi)$ and $e_s^{\min}(\phi)$ is similar to the one for $\mathcal{P}_{\bowtie c}[\phi]$. We first partition S into S^0, S^∞ and $S^?$. In both cases, S^0 contains all states that satisfy ϕ, and therefore, have expected cost 0. For $e_s^{\max}(\phi)$, S^∞ contains states that have $p_s^A(\diamond\phi) < 1$ for some adversary A, and can be computed by the model checking procedure in Section 3.1.1 for formula $\neg\mathcal{P}_{\geq 1}[\diamond\phi]$; For $e_s^{\min}(\phi)$, S^∞ contains states that have $p_s^A(\diamond\phi) < 1$ for all adversaries A, and is computed by model checking $\mathcal{P}_{<1}[\diamond\phi]$.

The computation of $e_s^{\max}$ for $S^? = S \backslash (S^0 \cup s^\infty)$ can be done in two ways. One is to solve a linear optimisation problem over variables $\{x_s \mid s \in S^?\}$ ($e_s^{\max} = x_s$): Minimising $\sum_{s \in S^?} x_s$ under the constraints

$$x_s \geq \mathbf{c}(a) + \sum_{s' \in S^?} \mu(s') \cdot x_{s'}$$

for all $s \in S^?$ and all $(a, \mu) \in Steps(s)$. The other way computes $e_s^{\max} = \lim_{n\to\infty} e_s^{\max(n)}$ iteratively where

$$e_s^{\max(n)} = \begin{cases} \infty & \text{if } s \in S^\infty \\ 0 & \text{if } s \in S^0 \\ 0 & \text{if } s \in S^? \text{ and } n = 0 \\ \max\limits_{(a,\mu)\in Steps(s)} \left\{ \mathbf{c}(a) + \sum\limits_{s' \in S^?} \mu(s') \cdot e_s^{\max(n-1)} \right\} & \text{if } s \in S^? \text{ land } n > 0. \end{cases}$$

Similarly, the computation of $e_s^{\min}$ for $S^?$ can be done by maximising $\sum\limits_{s\in S^?} x_s$ under the constraints

$$x_s \leq \mathbf{c}(a) + \sum_{s' \in S^?} \mu(s') \cdot x_{s'}$$

for all $s \in S^?$ and all $(a, \mu) \in Steps(s)$, or computing $e_s^{\min} = \lim_{n\to\infty} e_s^{\min(n)}$ iteratively where

$$e_s^{\min(n)} = \begin{cases} \infty & \text{if } s \in S^\infty \\ 0 & \text{if } s \in S^0 \\ 0 & \text{if } s \in S^? \text{ and } n = 0 \\ \min\limits_{(a,\mu)\in Steps(s)} \left\{ \mathbf{c}(a) + \sum\limits_{s' \in S^?} \mu(s') \cdot e_s^{\min(n-1)} \right\} & \text{if } s \in S^? \text{ land } n > 0. \end{cases}$$

The extended PCTL fro DTMCs/MDPs does not distinguish instantaneous and cumulative rewards because we can push cost assigned to states to transitions without weakening the power of the cost structure. For CTMCs, however, we added two different types of cost for instantaneous and cumulative rewards separately due to the continuous probability distributions. Every transition is associated with an instantaneous cost and every state has a cumulative cost.

Definition 3.22 *A labelled CTMC is a tuple* $(\mathcal{C}, \mathbf{C}, \mathbf{c})$ *where*

- $\mathcal{C} = (S, \overline{s}, \mathbf{R}, L)$ *is a CTMC,*
- $\mathbf{C} : S \times S \to \mathbb{R}^{\geq 0}$ *is an* instantaneous cost function*,*
- $\mathbf{c} : S \to \mathbb{R}^{\geq 0}$ *is a* cumulative cost function*.*

In the above definition, $\mathbf{C}(s, s')$ is the actual cost incurred when the system moves from state s to state s'; while $\mathbf{c}(s)$ is the coefficient, at which cost is incurred in state s, for the amount of time t spent in s, i.e., the actual cost would be $\mathbf{c}(s) \cdot t$. As for DTMCs and MDPs, we can define the expected cost of reaching a set of target states F through paths $\omega \in Path_s$:

$$cost(F)(\omega) \stackrel{def}{=} \begin{cases} \sum\limits_{i=1}^{\min\{j|\omega(j)\in F\}} \mathbf{C}\big(\omega(i-1), \omega(j)\big) + \mathbf{c}\big(\omega(i-1)\big) \cdot time(\omega, i-1) & \text{if } \exists j \in \mathbb{N} \,.\, \omega(j) \in F \\ \infty & \text{otherwise.} \end{cases}$$

We define the cost for a path that does not pass any target state to be ∞ too. Thus, the expected cost of reaching a state in F from state s is finite if all non-zero probability paths starting from s pass a state in F.

We extend CSL to include formulae for rewards now, as we did for DTMCs and MDPs.

Definition 3.23 *The syntax of CSL is as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi] \mid \mathcal{E}_{\bowtie c}[\phi], \quad \psi ::= X\ \phi \mid \phi\, \mathcal{U}^I \phi \mid \phi\, \mathcal{U}\ \phi$$

where a *is an atomic proposition,* $\bowtie \in \{\leq, <, \geq, >\}$*,* $p \in [0, 1]$*,* $c \in \mathbb{R}^{\geq 0}$ *and* I *is an interval of* $\mathbb{R}^{\geq 0}$*.*

Similarly to the $\mathcal{E}_{\bowtie c}$ operator for DTMCs and MDPs, $\mathcal{E}_{\bowtie c}[\phi]$ in CSL means that the expected cost to reach a state satisfying ϕ meets the bound $\bowtie c$ except that the expected cost here involves instantaneous and cumulative costs.

Let E_s denote the expectation with respect to the measure $Prob_s$. The satisfaction of $\mathcal{E}_{\bowtie c}[\phi]$ is defined as follow:

$$s \models \mathcal{E}_{\bowtie c}[\phi] \text{ iff } e_s(\phi) \bowtie c,$$

where $e_s(\phi) \stackrel{def}{=} E_s\big(cost(\{t \in S \mid t \models \phi\})\big)$. The corresponding model checking procedure is based on the embedded DTMC. We first identify the sets S^0, S^∞ and $S^?$ of states. The states in S^0 have $e_s(\phi) = 0$, and simply are the states that satisfy ϕ. The set S^∞ contains states from which the probability of reaching a ϕ state is less than 1. This set is computed in the same manner as the computation of S^∞ for $\mathcal{E}_{\bowtie c}[\phi]$ in PCTL. The set $S^? = S \backslash (S^0 \cup S^\infty)$ is computed by solving the linear equation system in variables $\{x_s \mid s \in S^?\}$:

$$x_s = \Big(\sum_{s \in S^?} \mathbf{P}(s,s') \cdot \big(\mathbf{C}(s,s') + x_{s'}\big) \Big) + \frac{1}{E(S)} \cdot \mathbf{c}(s)$$

and then letting $e_s(\phi) = x_s$ for $s \in S^?$.

3.3 Compositional verification

We have seen several proper models for CONNECTors and efficient verification techniques for both functional and non-functional requirements in previous sections. However, applying these techniques to large, real-life systems remains challenging. In this section, we propose a set of compositional verification techniques, known as assume-guarantee methods, to tackle the challenge. The basic idea is that we partition a system into small components, and verify each component individually under some assumption. The assumption can come from the system specification, or the verification result of other components. Once the verification on individual components completes, the correctness of the whole system can be inferred, thus avoiding the construction of the product state space. Similar techniques have been studied intensively for non-quantitative verification in the past. Until very recently, however, there has been little progress towards compositional verification for systems exhibiting both probabilistic and nondeterministic behaviour. In this section, we apply the assume-guarantee methods to devise a quantitative probabilistic model checking approach to handle non-functional requirements for CONNECTors.

A large-scale system usually is composed of a group of components. To verify such systems, an intuitive solution would be to verify each component separately and deduce the correctness of the full system from the verification results for the components, if we cannot verify the overall system as a whole due to memory and time limits. In this solution, we need to make some assumption when we verify each component, i.e., we assume the outside world around the component exhibits a certain kind of behaviour. Under the given assumption, if the component does not guarantee some behaviour, the system on the whole fails to pass the verification. Otherwise, we continue to verify other components until we traverse all the components.

In a non-quantitative assume-guarantee approach, we usually write $\langle A \rangle\ M\ \langle G \rangle$ to represent the fact that *the component M is guaranteed to satisfy the property G under the assumption A*. Since components interact with each other, the assumption for one component may come from the behaviour of other components. For example, a system with two components M_1 and M_2 satisfies the property G if we know $\langle true \rangle\ M_1\ \langle A \rangle$ and $\langle A \rangle\ M_2\ \langle G \rangle$. When we extend the above idea to deal with probabilistic systems, we associate probability to both assumptions and guarantees such that $\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{\geq p_G}$, which means "whenever the environment around M satisfies A with probability at least p_A, then M satisfies G with probability at least p_G".

In this work, we suppose that both assumptions and guarantees are *safety properties*, which describe a wide range of useful properties like "the maximum probability of an error occurring is at most 0.01". A *regular safety property* A represents a set of infinite words, denoted $L(A)$, that is characterised by a regular language of bad prefixes, that is, finite words whose extension is not in $L(A)$. More precisely, we will define a regular safety property A by a (complete) deterministic finite automaton (DFA) $A^{err} = (Q, \overline{q}, \alpha_A, \delta_A, F)$, comprising states Q, initial state $\overline{q} \in Q$, alphabet α_A, transition function $\delta_A : Q \times \alpha_A \to Q$ and accepting states $F \subseteq Q$. The DFA A^{err} defines, in standard fashion, a regular language $L(A^{err}) \subseteq (\alpha_A)^*$. The language $L(A)$ is then defined as $L(A) = \{w \in (\alpha_A)^\omega \mid \text{ no prefix of } w \text{ is in } L(A^{err})\}$. Frequently, safety properties can also be specified by LTL formulae, such as $\Box ap$ (ap always holds) and $ap_1\ R\ ap_2$ (ap_1

releases ap_2). In the rest of the section, we only discuss the assume-guarantee approach with DFAs, but it applies to LTL safety formulae as well.

Verification of $\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{\geq p_G}$ requires the use of multi-objective model checking [39]. The conventional single-objective approach allows us to check whether, for all adversaries (or, dually, for at least one adversary), the probability of some property is above (or below) a given bound. Multi-objective queries allow us to check the existence of an adversary satisfying multiple properties of this form by a reduction to a linear programming (LP) problem.

We use *probabilistic automata* [61, 63], which are similar to MDPs, to establish a sequence of assume-guarantee rules to obtain compositional quantitative-probabilistic verification.

Definition 3.24 *Let $Dist(S)$ be the set of all discrete probability distributions over a set S, and τ "internal actions". A probabilistic automaton (PA) is a tuple $M = (S, \overline{s}, \alpha_M, \delta_M, L)$ where S is a set of states, $\overline{s} \in S$ is an initial state, α_M is an alphabet, $\delta_M \subseteq S \times (\alpha_M \cup \{\tau\}) \times Dist(S)$ is a probabilistic transition relation and $L : S \to 2^{AP}$ is a labelling function, assigning atomic propositions from a set AP to each state.*

A *transition* with an action a and a discrete probability distribution μ over states, denoted $s \xrightarrow{a} \mu$ is available in state $s \in S$ if $(s, a, \mu) \in \delta_M$. To reason about probabilistic systems comprising multiple components, we need the notion of *parallel composition*.

Definition 3.25 (Parallel compositions of PAs) *Let η_s be the point distribution on $s \in S$, and $M_1 = (S_1, \overline{s_1}, \alpha_{M_1}, \delta_{M_1}, L_1)$ and $M_2 = (S_2, \overline{s_2}, \alpha_{M_2}, \delta_{M_2}, L_2)$ two PAs. The parallel composition of M_1 and M_2, denoted $M_1 \parallel M_2$, is given by the PA $(S_1 \times S_2, (\overline{s_1}, \overline{s_2}), \alpha_{M_1} \cup \alpha_{M_2}, \delta_{M_1 \| M_2}, L)$ where $\delta_{M_1 \| M_2}$ is defined such that $(s_1, s_2) \xrightarrow{a} \mu_1 \times \mu_2$ iff one of the following holds:*

- $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$ *and* $a \in \alpha_{M_1} \cap \alpha_{M_2}$
- $s_1 \xrightarrow{a} \mu_1$, $\mu_2 = \eta_{s_2}$ *and* $a \in (\alpha_{M_1} \backslash \alpha_{M_2}) \cup \{\tau\}$
- $s_2 \xrightarrow{a} \mu_2$, $\mu_1 = \eta_{s_1}$ *and* $a \in (\alpha_{M_2} \backslash \alpha_{M_1}) \cup \{\tau\}$

and $L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$.

Using standard automata-based techniques for model checking PAs [34], verifying correctness of probabilistic safety properties reduces to model checking the product of a PA and a DFA.

Definition 3.26 (PA-DFA product) *The product of a PA $M = (S, \overline{s}, \alpha_M, \delta_M, L)$ and DFA $A^{err} = (Q, \overline{q}, \alpha_A, \delta_A, F)$ with $\alpha_A \subseteq \alpha_M$ is given by the PA $M \otimes A^{err} = (S \times Q, (\overline{s}, \overline{q}), \alpha_M, \delta', L')$ where*

- $(s, q) \xrightarrow{a} \mu \times \eta_{q'}$ *if* $s \xrightarrow{a} \mu$, *and* $q' = \delta_A(q, a)$ *if* $a \in \alpha_A$ *or* $q' = q$ *otherwise;*
- $L'(s, q) = L(s) \cup \{err_A\}$ *if* $q \in F$ *and* $L'(s, q) = L(s)$ *otherwise.*

The first assume-guarantee proof rule we consider is asymmetric, in the sense that we require only a single assumption about one component. Experience in the non-probabilistic setting [57] indicates that, despite its simplicity, rules of this form are widely applicable.

Theorem 3.27 *If M_1, M_2 are probabilistic automata and $\langle A \rangle_{\geq p_A}$, $\langle G \rangle_{\geq p_G}$ probabilistic safety properties such that $\alpha_A \subseteq \alpha_{M_1}$ and $\alpha_G \subseteq \alpha_{M_2} \cup \alpha_A$, then the following assume-guarantee proof rule holds:*

$$\frac{\langle true \rangle\ M_1\ \langle A_1 \rangle_{\geq p_A} \qquad \langle A \rangle_{\geq p_A}\ M_2\ \langle G \rangle_{\geq p_G}}{\langle true \rangle\ M_1 \parallel M_2\ \langle G \rangle_{\geq p_G}} \quad \text{(ASYM)}$$

$$\frac{\langle true\rangle\ M_1\ \langle A_1,\ldots,A_k\rangle_{\geq p_1,\ldots,p_k} \quad \langle A_1,\ldots,A_k\rangle_{\geq p_1,\ldots,p_k}\ M_2\ \langle G\rangle_{\geq p_G}}{\langle true\rangle\ M_1 \parallel M_2\ \langle G\rangle_{\geq p_G}} \qquad \text{(ASym-Mult)}$$

This theorem means that, given an appropriate assumption $\langle A\rangle_{\geq P_A}$, we can check the correctness of a probabilistic safety property $\langle G\rangle_{\geq P_G}$ on $M_1 \parallel M_2$, without constructing and model checking the full model. Instead, we perform one instance of (standard) model checking on M_1 (to check the first condition of rule (ASym)) and one instance of multi-objective model checking on $M_2[\alpha_A] \otimes A^{err}$ (to check the second). If A^{err} is much smaller than M_1, we can expect significant gains in terms of the verification performance.

Let $\langle A_1,\ldots,A_k\rangle_{\geq p_1,\ldots,p_k}$ be the conjunction of probabilistic safety properties $\langle A_i\rangle_{\geq p_i}$ for $i = 1,\ldots,k$. We can generalise the rule (ASym) as follows:

We also observed that, through repeated application of (ASym), we obtain a rule of the following form for n components:

$$\frac{\begin{array}{c}\langle true\rangle\ M_1\ \langle A_1\rangle_{\geq p_1} \\ \langle A_1\rangle_{\geq p_1}\ M_2\ \langle A_2\rangle_{\geq p_2} \\ \ldots \\ \langle A_{n-1}\rangle_{\geq p_{n-1}}\ M_n\ \langle G\rangle_{\geq p_G}\end{array}}{\langle true\rangle\ M_1 \parallel \ldots \parallel M_2\ \langle G\rangle_{\geq p_G}} \qquad \text{(ASym-N)}$$

One potential limitation of the rule (ASym) is that we may not be able to show that the assumption A_1 about M_1 holds without making additional assumptions about M_2. This can be overcome by using the following circular proof rule:

Theorem 3.28 *If M_1, M_2 are PAs and $\langle G\rangle_{\geq p_G}$, $\langle A_1\rangle_{\geq p_1}$ and $\langle A_2\rangle_{\geq p_2}$ probabilistic safety properties such that $\alpha_{A_2} \subseteq \alpha_{M_2}$ and $\alpha_{A_1} \subseteq \alpha_{M_1} \cup \alpha_{A_2}$ and $\alpha_G \subseteq \alpha_{M_2} \cup \alpha_{A_1}$, then the following circular assume-guarantee proof rule holds:*

$$\frac{\begin{array}{c}\langle true\rangle\ M_2\ \langle A_2\rangle_{\geq p_2} \\ \langle A_2\rangle_{\geq p_2}\ M_1\ \langle A_1\rangle_{\geq p_1} \\ \langle A_1\rangle_{\geq p_1}\ M_2\ \langle G\rangle_{\geq p_G}\end{array}}{\langle true\rangle\ M_1 \parallel M_2\ \langle G\rangle_{\geq p_G}} \qquad \text{(Circ)}$$

Sometimes, part of a system comprises several asynchronous components, that is, components with disjoint alphabets. In such cases, it can be difficult to establish useful probability bounds on the combined system if the fact that the components act independently is ignored. To overcome this problem, we introduce the following rule.

Theorem 3.29 *For any PAs M_1, M_2 and probabilistic safety properties $\langle A\rangle_{\geq p_A}$, $\langle A_1\rangle_{\geq p_1}$ and $\langle A_2\rangle_{\geq p_2}$ such that $\alpha_{M_1} \cap \alpha_{M_2} = \emptyset$, $\alpha_{A_1} \subseteq \alpha_{M_1} \cup \alpha_A$ and $\alpha_{A_2} \subseteq \alpha_{M_2} \cup \alpha_A$, we have the following asynchronous assume-guarantee proof rule:*

We have implemented our compositional verification approach in a prototype tool. Using the rules given above, verification requires both standard (automata-based) model checking and multi-objective model checking. Our tool is based on the probabilistic model checker PRISM [43], which already supports LTL model checking of probabilistic automata. Model checking of probabilistic safety properties, represented by DFAs, can be achieved with existing versions of PRISM, since DFAs can easily be encoded in PRISM's modelling language. For multi-objective model checking, we have extended PRISM with an implementation of the techniques in [39]. This requires the solution of Linear Programming (LP) problems, for which we use the ECLiPSe Constraint Logic Programming system with the COIN-OR CBC solver, implementing a branch-and-cut algorithm.

$$\frac{\langle A\rangle_{\geq p_A}\ M_1\ \langle A_1\rangle_{\geq p_1} \quad \langle A\rangle_{\geq p_A}\ M_2\ \langle A_2\rangle_{\geq p_2}}{\langle A\rangle_{\geq p_A}\ M_1 \parallel M_2\ \langle A_1 \vee A_2\rangle_{\geq p_1+p2-p_1\cdot p_2}} \quad \text{(ASYNC)}$$

3.4 Online method

In CONNECT, heterogeneous networked systems are able to join and leave a CONNECTed system, and move within the system or across CONNECTed systems. Thus, CONNECTors are likely to be created and destroyed on-the-fly. They should be able to evolve to meet the different QoS requirements. Verification techniques that can deal with the dynamism are needed. In the previous sections, several probabilistic models for CONNECTors were introduced. Now we present a framework to address the issue of self-adaption of CONNECTors [29]. By monitoring system properties regularly, based on snapshots of a system, the system itself can change its behaviour accordingly. This work was not specifically targeted at CONNECTed systems, but it can be adapted to CONNECT without difficulty. Figure 3.1 depicts the high-level architecture of an autonomic CONNECTed system corresponding to the one proposed in [29]. We assume that each CONNECTor has an interface exposed to the outside world, e.g., Enabler. In the interface, there are some parameters whose value can be changed during run-time. Given a set of pre-defined policies (i.e., system-wide QoS objectives), an autonomic manager monitors CONNECTors through sensors, uses its knowledge to analyse their state and to plan changes in their configurable parameters, and implements (or "executes") these changes through effectors.

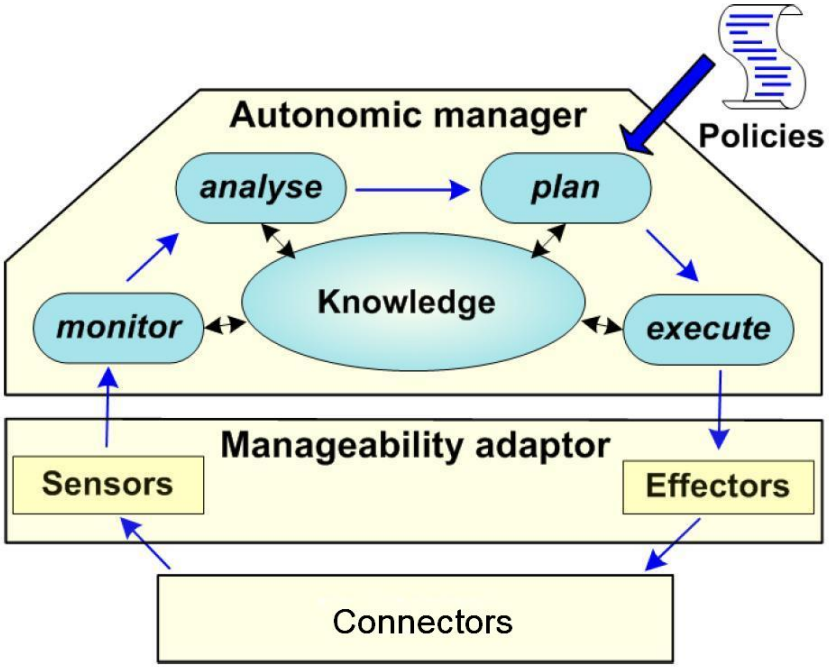

Figure 3.1: Autonomic management system

The novel characteristics of our framework are:

- The knowledge within the autonomic manager consists of a continuous- or discrete-time Markov chain that models the behaviour of the managed CONNECTors.
- Runtime quantitative analysis of the Markov chain is employed for the analysis step in Figure 3.1.
- The autonomic manager can be integrated within enablers or implemented as a web service that integrates the generic policy engine and the quantitative analysis tool PRISM.

- Off-the-shelf tools are used for the computer-assisted generation of the manageability adaptor from Figure 3.1 starting from the Markov chain.

The autonomic systems developed using the framework can implement a rich and flexible set of high-level policies that is unavailable in existing autonomic solutions. This is due to the broad spectrum of quantitative properties that can be specified in the temporal logics supported by PRISM, the quantitative analysis tool integrated within our autonomic manager. The decisions taken by the autonomic manager are based on an exhaustive analysis of the user-specified policies and of the managed CONNECTors. This powerful capability is made possible by our use of runtime quantitative analysis.

The generic method for the development of autonomic CONNECTor is composed of three stages.

- The manageability adaptor required to organise an existing CONNECTor into an autonomic system is devised by the system developer during the generation stage.
- In the deployment stage, this adaptor is deployed, and the policy engine is configured by the system administrator;
- Policies expressing the high-level system objectives are specified by the end user in the exploitation stage.

In the rest of the section, we propose an implementation of the generic method, which is based on the policy engine presented in [28]. Figure 3.2 shows the basic steps in the system development.

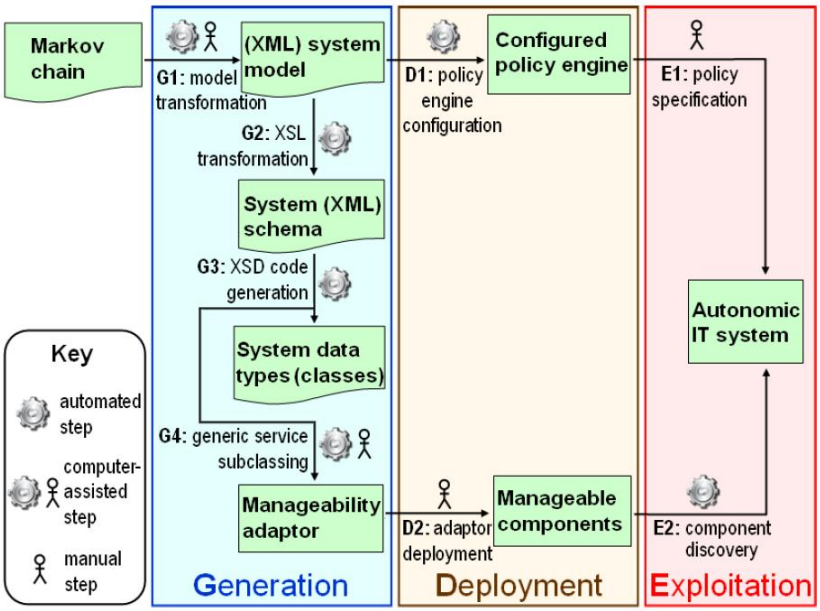

Figure 3.2: Autonomic system development

1. The first stage of the method starts from a PRISM-encoded Markov chain describing the behaviour of the CONNECTor(s) to be included in the autonomic system.

Step G1 In the first generation step, the Markov chain is used to derive an XML model for the configuration of the policy engine.

Step G2 In this step, a standard XSLT engine is used to apply a simple XSL transformation to the XML system model, and thus to automatically extract an XML schema specification for the targeted CONNECTor(s).

Step G3 A standard data type generator is employed to generate the set of data types associated with the XML schema.

Step G4 Finally, a simple transformation was implemented to automate the generation of manageability adaptor stubs for the components in the CONNECTed system.

2. The second stage is the deployment.

Step D1 In the first deployment step, the XML system model from **Step G1** is supplied to the running instance of the policy engine that will be used in the autonomic system. Given the implementation of the engine as a web service, this involves the invocation of one of its web methods.

Step D2 The second development step consists of setting up the manageability adaptors built during the generation stage, and connecting them to the actual CONNECTors to be included in the system. The first part of the operation represents a standard deployment of a web service, whereas the second part depends on the interface between the adaptor and the system components, but it typically involves configuring the two elements so that they know each other's address.

3. The last stage is the exploitation.

Step E1 In this step, user-specified policies that express the objectives of the system as functions of its parameters are devised and supplied to the policy engine, e.g., by using the web client that was employed to upload the system model in **Step D1**. Typically, these policies are modified over time to reflect changes in the system goals. Depending on the configuration of the policy engine, the policies are evaluated and implemented either periodically or when the engine receives notifications about changes in the values of the system parameters from the manageability adaptors.

Step E2 The policy engine detects automatically all manageable CONNECTors that have been registered with its component discovery service, and whose types are specified in the system model used for its configuration.

3.5 PRISM

PRISM [43] is a popular probabilistic model checker, which was started at University of Birmingham, UK, and is maintained and updated at the University of Oxford, UK, following the transfer of the PRISM team to Oxford. PRISM can handle discrete and continuous time Markov chain models (DTMCs, CTMCs) and Markov decision processes (MDPs). It is able to verify DTMC/MDP models against PCTL for probabilistic temporal logic formulae and cost/reward-based properties, and verify CTMCs against CSL for logic and reward formulae. Both states and transitions in a system can be associated with rewards, and both instantaneous and cumulative rewards properties can be checked. The support for PTAs is partial and currently under development. The linear temporal logic (LTL) [58] is another widely adopted formalism for model checking software. Recently, PRISM was extended to include LTL model checking for DTMC/MDP.

A system specified in the input format for PRISM is contained in a textual file. At the beginning of the model, a keyword of *CTMC*, *DTMC* or *MDP* indicates the type of Markov chain model. PRISM allows the parallel composition of system modules via labelled transitions. Similar modules can be generated using a module template. Parameters can be passed to each module instance. A system state is a valuation of the set of system variables of integer and real number types. Each module contains a list of guarded transitions among states.

The GUI of PRISM is equipped with an editor for creating and editing probabilistic models. The editor displays keywords in different colours and detects syntax errors automatically. It also builds an overview of the model to show the modular structure and variables. Figure 3.3 illustrates a screen shot of the editor. The property verification window in the GUI allows users to create and edit PCTL/CSL properties for verification. The list of properties can be saved to a file, or loaded from the file. Additionally, to verify the properties directly in the model, the GUI has a very useful feature: a parameter can have a sequence of values in which case verification is performed on each value. A curve is displayed on the screen to demonstrate the trend of the verification results over the sequence of values specified. Figure 3.4 shows a screen shot of the window with a verification result. The third window in the GUI is the simulation window. It allows users to execute the model step by step by choosing the sequence of transitions manually. In

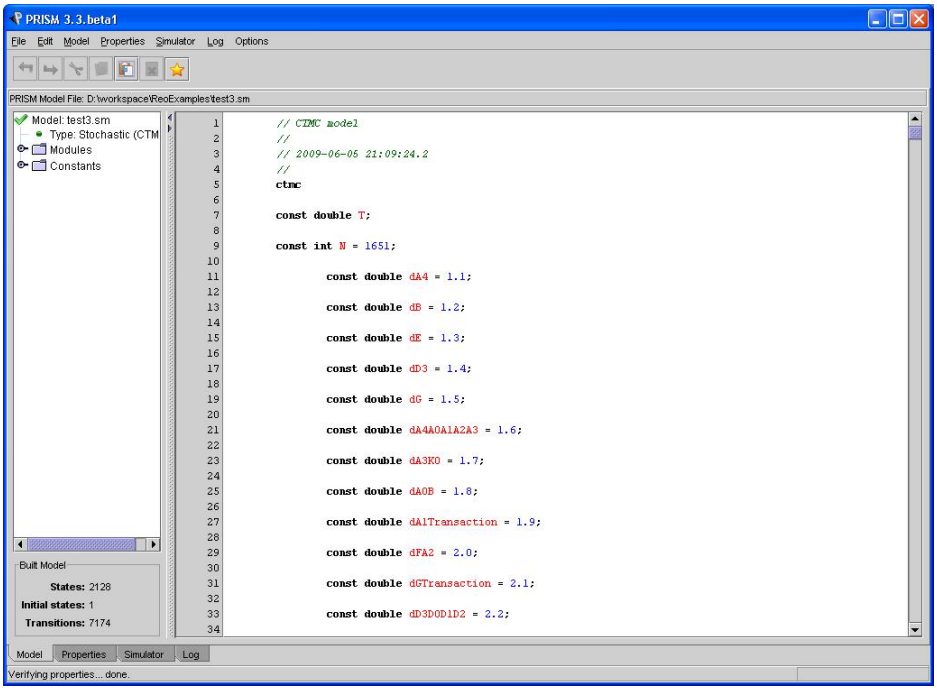

Figure 3.3: The editor of PRISM

this way, a deep understanding of the model can be gained. This feature is very helpful in debugging the model.

So far PRISM has been applied to numerous probabilistic models, such as network protocols, security protocols, randomised distributed algorithms, biological processes, etc. In the case of some of these models, which contain more than 10^8 states, PRISM has been shown to cope admirably. The verification engine may also be fine-tuned by the user to reduce the overall memory and time consumption.

Verification of CONNECTors: A Case Study

In order to demonstrate the potential of probabilistic model checking of CONNECTors, we used the tool chain Reo2MC [15] to construct an experiment, which is based on a simplified version of the *Mary Scenario* presented in Section 2.2. We assume there are three contacts that need to be transferred from the old phone to the new phone. Figure 3.5 shows the Reo circuit connecting two phones. The new phone "CM2" sends three requests sequentially, each of which is used to get a contact, through the output port "out". The old phone "CM1" receives the requests from the input port "in" and sends through its "out" port the contacts, which are received by "CM2" in the "in" port. The Reo connector system guarantees the ordering among the data flow: each request is followed by a reply before the next request is issued. Figure 3.6 shows the trend of the probability of completing the contact transmission within different intervals, which is specified as the following PRISM CSL formula

$$P =?[true\ U[0,T]\ TransComplete].$$

The generated CTMC model has 2128 states and 7174 transitions.

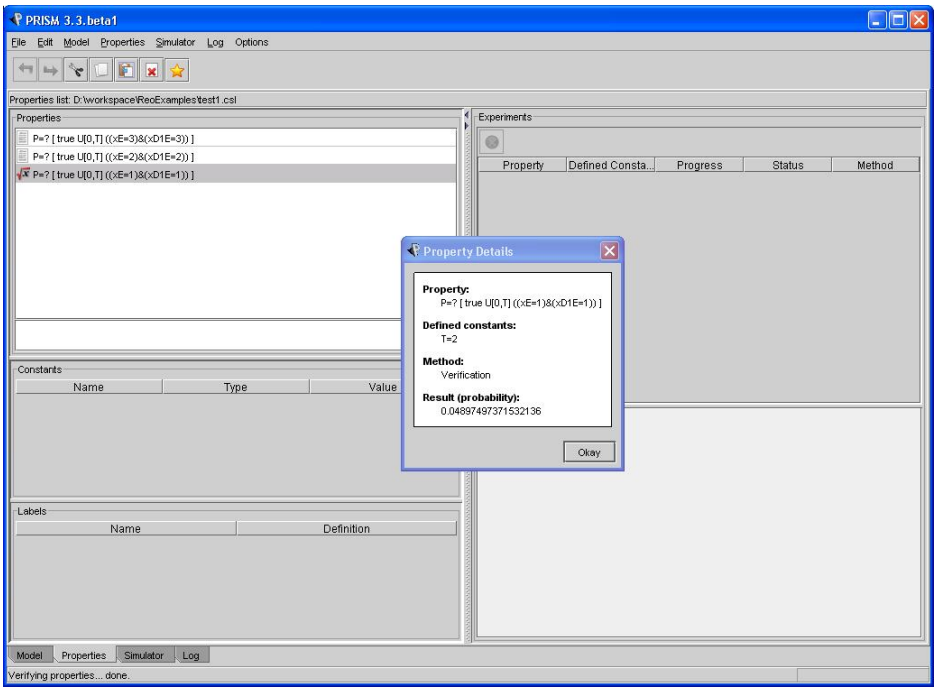

Figure 3.4: The property verification window

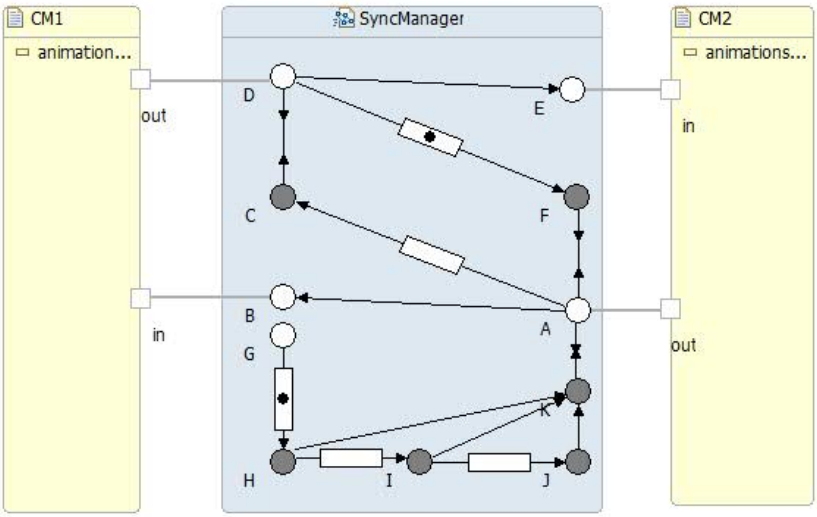

Figure 3.5: The simplified Mary Scenario

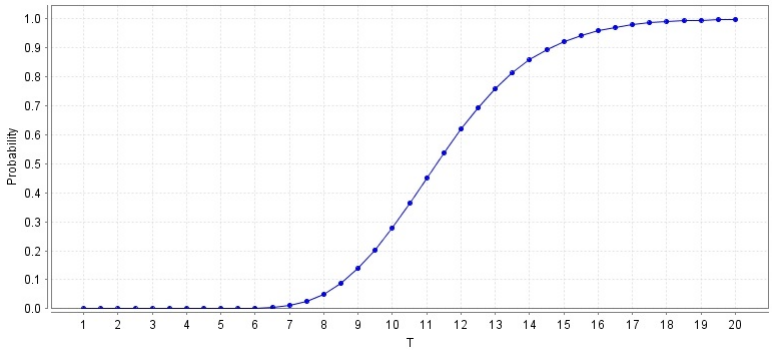

Figure 3.6: The experimental results

4 Future work

In the future, we will continue working on a high-level formalism for connectors that meets the eight dimensions identified in CONNECT. In particular, we will focus on interaction types and composition operators. Moreover, we will aim for such a formalism to have an intuitive semantic mapping onto labelled transition system models, such as deterministic automata (used for learning from scenarios and synthesis) or probabilistic automata (used for QoS analysis). This requires that we investigate relations between the defined high-level formalism for connectors and formalisms available for non-functional analyses as well.

During the design of the formalism, we need to answer the following questions regarding the relationship between components and connectors: is a specific connector type "strictly tied" to the specific component type(s)? In other words, can a connector be defined independently from the specific component type(s) it will be attached to? Is it possible and does it make any sense to be able to define interaction without a specific underlying component model?

Among the connectors we surveyed in this document, some of them have the ability to specify both components and connectors, e.g., WRIGHT, while others do not, e.g., Reo. On one hand, the dimension "reusability" sometimes requires connectors to be definable independently from the components. On the other hand, we need to be able to specify (or simulate) components' behaviour in order to verify the system's behaviour as a whole.

Furthermore, we intend to continue the scientific advances concerning compositional quantitative (not only probabilistic) verification in the style of assume-guarantee. One direction would be to generalise the proof rules in Section 3.3 to check more properties. Moreover, the multi-objective model checking technique [39] used in Section 3.3 provides a theoretical basis for assume-guarantee methods that utilise rewards for the verification of non-functional requirements. This technique verifies whether the multiple objective queries are satisfied on a model, and, if yes, computes an adversary satisfying all the queries. For each transition in the model, the adversary gives the expected number of times that we execute the transition in order to satisfy all the queries. We can compute the expected reward for reaching a set of target states under the adversary satisfying all the queries as a weighted sum. If there exists more than one adversary satisfying the queries, we can obtain the one that maximises the expected reward. Another future direction will be to adapt the proof rules in Section 3.3, or establish new rules, for rewards.

If time permits, we will study parametric probabilistic model checking for online verification. Currently, some work has been done on probabilistic reachability analysis, e.g., [40], which computes a polynomial function over the parameters. The probabilistic reachability can be calculated directly using the function with concrete values of parameters. Thus there is no need to build the model and search the state space every time we obtain different parameter values, which would speed up online monitoring to a great extent. We would like to explore the possibility to generate (maybe non-polynomial) functions for PCTL formulae in the future.

Last but not least, we will develop techniques for compositional modelling and verification of timing properties, such as communication delays and jitters. Such properties can be modelled by extensions of automata or labelled-transition system formalisms (which can be seen as an example of using Q-algebras). The analysis suffers the usual problem of state-space explosion for larger systems. We will therefore also investigate how such properties can be specified in a less expressive formalism in which compositional analysis can be performed more efficiently.

Bibliography

[1] CONNECT consortium. CONNECT Annex I: Description of Work. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[2] CONNECT consortium. CONNECT Deliverable D1.1: Initial CONNECT Architecture. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[3] CONNECT consortium. CONNECT Deliverable D5.1: Conceptual models for assessment and assurance of dependability, security and privacy in the eternal CONNECTed world. FET IP CONNECT EU project, FP7 grant agreement number 231167, http://connect-forever.eu/.

[4] Eclipse Coordination Tools. http://reo.project.cwi.nl/.

[5] ITU Bigraphical Programming Language. http://www.itu.dk/research/bpl/.

[6] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

[7] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[8] F. Arbab. The IWIM model for coordination of concurrent activities. In P. Ciancarini and C. Hankin, editors, *COORDINATION*, volume 1061 of *Lecture Notes in Computer Science*, pages 34–56. Springer, 1996.

[9] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[10] F. Arbab. Abstract Behavior Types: a foundation model for components and their composition. *Science of Computer Programming*, 55(1-3):3–52, March 2005.

[11] F. Arbab, M. M. Bonsangue, and F. S. de Boer. A coordination lanuage for mobile components. In *SAC (1)*, pages 166–173, 2000.

[12] F. Arbab, T. Chothia, R. Mei, S. Meng, Y. Moon, and C. Verhoef. From coordination to stochastic models of QoS. In *COORDINATION '09: Proceedings of the 11th International Conference on Coordination Models and Languages*, pages 268–287, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] F. Arbab, T. Chothia, S. Meng, and Y.-J. Moon. Component connectors with QoS guarantees. In A. L. Murphy and J. Vitek, editors, *COORDINATION*, volume 4467 of *Lecture Notes in Computer Science*, pages 286–304. Springer, 2007.

[14] F. Arbab, F. S. de Boer, and M. M. Bonsangue. A logical interface description language for components. In A. Porto and G.-C. Roman, editors, *COORDINATION*, volume 1906 of *Lecture Notes in Computer Science*, pages 249–266. Springer, 2000.

[15] F. Arbab, S. Meng, Y.-J. Moon, M. Kwiatkowska, and H. Qu. Reo2mc: a tool chain for performance analysis of coordination models. In *The 7th joint meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*, pages 287–288. ACM, 2009.

[16] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T. Henzinger, editors, *Proc. 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[17] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In P. Wolper, editor, *Proc. 7th International Conference on Computer Aided Verification (CAV'95)*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.

[18] C. Baier. Probabilistic models for reo connector circuits. *Journal of Universal Computer Science*, 11(10):1718–1748, 2005.

[19] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

[20] C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.

[21] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[22] G. Berry and G. Boudol. The chemical abstract machine. In *POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94, New York, NY, USA, 1990. ACM.

[23] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[24] P. Bidinger, A. Schmitt, and J.-B. Stefani. An abstract machine for the kell calculus. In M. Steffen and G. Zavattaro, editors, *FMOODS*, volume 3535 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2005.

[25] P. Bidinger and J.-B. Stefani. The kell calculus: Operational semantics and type system. In E. Najm, U. Nestmann, and P. Stevens, editors, *FMOODS*, volume 2884 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2003.

[26] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.

[27] S. Bliudze and J. Sifakis. The Algebra of Connectors - Structuring Interaction in BIP. *IEEE Transactions on Computers*, 57(10):1315–1330, 2008.

[28] R. Calinescu. Implementation of a generic autonomic framework. In D. G. et al., editor, *Proc. 4th International Conference on Autonomic and Autonomous Systems (ICAS'08)*, pages 124–129. IEEE Computer Society Press, March 2008.

[29] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic it systems. In *Proc. 31st International Conference on Software Engineering*, pages 100–110. IEEE Computer Society Press, May 2009.

[30] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *FoSSaCS*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[31] T. Chothia and J. Kleijn. Q-automata: Modelling the resource usage of concurrent components. *Electr. Notes Theor. Comput. Sci.*, 175(2):153–167, 2007.

[32] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992.

[33] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society Press, 1988.

[34] C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. In M. Paterson, editor, *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 336–349. Springer, 1990.

[35] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[36] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[37] L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In J. Baeten and S. Mauw, editors, *Proc. 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 66–81. Springer, 1999.

[38] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[39] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In O. Grumberg and M. Huth, editors, *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, pages 50–65. Springer, 2007.

[40] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. In *SPIN*, volume 5578 of *LNCS*, pages 88–106. Springer, 2009.

[41] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[42] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.

[43] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[44] P. Inverardi, P. Pelliccione, and M. Tivoli. Towards an assume-guarantee theory for adaptable systems. In *International workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS 2009) co-located with ICSE 2009. Vancouver, BC, Canada*, May 18-19, 2009.

[45] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. Technical Report RR-08-06, Oxford University Computing Laboratory, February 2008.

[46] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In *Proc. 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09)*, 2009. To appear.

[47] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.

[48] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine and F. Vaandrager, editors, *Proc. 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *LNCS*, pages 212–227. Springer, 2009.

[49] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[50] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 178–187, New York, NY, USA, 2000. ACM.

[51] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[52] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge Univ. Press, 1999.

[53] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, New York, NY, USA, 2009.

[54] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[55] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.

[56] R. D. Nicola, G. L. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A process calculus for qos-aware applications. In J.-M. Jacquet and G. P. Picco, editors, *COORDINATION*, volume 3454 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2005.

[57] C. Pasareanu, D. Giannakopoulou, M. Bobaru, J. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, June 2008.

[58] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[59] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, Inc., New York, USA, 1998.

[60] A. Schmitt and J.-B. Stefani. The kell calculus: A family of higher-order distributed process calculi. In C. Priami and P. Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2004.

[61] R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[62] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 2005.

[63] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250 – 273, 1995.

[64] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39:1095–1100, 1953.

[65] M. Shaw and D. Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[66] J. Sifakis. A framework for component-based construction extended abstract. In *SEFM '05: Proceedings of the Third IEEE International Conference on Software Engineering and Formal Methods*, pages 293–300, Washington, DC, USA, 2005. IEEE Computer Society.

[67] J.-B. Stefani. A calculus of kells. *Electr. Notes Theor. Comput. Sci.*, 85(1), 2003.

[68] S. Tripakis. *Te formal analysis of timed systems in practice*. PhD thesis, Université Joseph Fourier, 1998.

[69] D. Wallace. *Groups, Rings & Fields*. Springer, 1998.