Stochastic Models for Quality of Service of Component Connectors

Young-Joo Moon

Stochastic Models for Quality of Service of Component Connectors

Stochastic Models for Quality of Service of Component Connectors

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden
op gezag van de Rector Magnificus prof. mr. P. F. van der Heijden
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 25 oktober 2011
klokke 10.00 uur

door

Young-Joo Moon

geboren te Pohang, Zuid-Korea

Promotor:	Prof. Dr. F. Arbab	Universiteit Leiden
Co-promotor:	Dr. A. Silva	Radboud University Nijmegen
	Dr. E.P. de Vink	Technische Universiteit Eindhoven
Other members:	Prof. Dr. F.S. de Boer	Universiteit Leiden
	Dr. M.M. Bonsangue	Universiteit Leiden
	Prof. Dr. J.-M. Jacquet	University of Namur, Belgium
	Dr. J. Kleijn	Universiteit Leiden
	Prof. Dr. J.N. Kok	Universiteit Leiden
	Prof. Dr. R. van der Mei	Vrije Universiteit Amsterdam
	Prof. Dr. M. Sirjani	Reykjavik University, Iceland

The work reported in this thesis has been carried out at the Center for Mathematics and Computer Science (CWI) in Amsterdam and Leiden Institute of Advanced Computer Science at Leiden University, under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The research was supported by the grant from the GLANCE funding program of NWO through Coordination with Performance Guarantees (CooPer) project (600.643.000.05N12).

Cover design by Young-Joo Moon & Song-Hee Lee.
Printed by Ponsen & Looijen.

ISBN: 978–90–6464–506–8
IPA Dissertation Seriese 2011-17

Contents

1 Introduction **1**
- 1.1 Quantitative analysis of systems . . . 2
- 1.2 Thesis overview and contributions . . . 3
  - 1.2.1 Contributions . . . 5

2 Models for component coordination **7**
- 2.1 Reo language . . . 7
- 2.2 Stochastic Reo . . . 8
- 2.3 Semantic models for Reo . . . 11
  - 2.3.1 Constraint Automata . . . 11
  - 2.3.2 Intentional Automata . . . 12
  - 2.3.3 Reo Automata . . . 17
- 2.4 Markov Chains . . . 21
- 2.5 Interactive Markov Chains . . . 22
- 2.6 Related work . . . 23
  - 2.6.1 Other coordination languages . . . 23
  - 2.6.2 Continuous-Time Constraint Automata . . . 24
  - 2.6.3 Stochastic Process Algebra . . . 25
  - 2.6.4 Stochastic Petri Nets . . . 26
  - 2.6.5 Stochastic Automata Networks . . . 27

3 Quantitative Intentional Automata **29**
- 3.1 Introduction . . . 29
- 3.2 Quantitative Intentional Automata . . . 30
  - 3.2.1 Invariants . . . 31
  - 3.2.2 QIA composition . . . 32
- 3.3 Translation into a stochastic model . . . 37
  - 3.3.1 Micro-step transitions . . . 39
  - 3.3.2 Extracting a delay-sequence . . . 40
  - 3.3.3 Dividing macro-step transitions with a delay-sequence . . . 41

3.3.4 Preemptive request-arrivals . . . 45
3.4 Discussion . . . 47

4 Stochastic Reo Automata **49**
4.1 Introduction . . . 49
4.2 Stochastic Reo Automata . . . 50
4.2.1 Stochastic Reo Automata . . . 50
4.3 Reward model . . . 57
4.3.1 Stochastic Reo with reward information . . . 58
4.3.2 Stochastic Reo Automata with reward information . . . 61
4.4 Translation into CTMC . . . 64
4.4.1 Synchronized data-flows . . . 64
4.4.2 Deriving the CTMC . . . 64
4.4.3 Rewards . . . 66
4.5 Interactive Markov Chains and Reo . . . 69
4.5.1 Interactive Markov Chains . . . 70
4.6 Discussion . . . 75

5 Tool implementation **77**
5.1 Introduction . . . 77
5.2 Reo2MC: description and implementation . . . 77
5.2.1 Implementation . . . 78
5.2.2 Usage . . . 87
5.3 Discussion . . . 94

6 Case study **97**
6.1 Introduction . . . 97
6.2 The ASK system . . . 98
6.2.1 Overview of the ASK system . . . 100
6.3 Modeling the ASK system . . . 102
6.3.1 The Reception component . . . 102
6.3.2 Extracting distributions from logs . . . 104
6.4 QoS analysis . . . 105
6.4.1 Analysis on derived CTMC . . . 105
6.4.2 Simulation . . . 109
6.5 Discussion . . . 112

7 Conclusions and Future work **115**
7.1 Conclusions . . . 115
7.2 Future work . . . 116

Bibliography **119**

Abstract **127**

Samenvatting **129**

Chapter 1

Introduction

In Service-oriented Computing (SOC), services distributed over a network are composed according to the requirements of service consumers. Services are platform- and network-independent applications that support rapid, low-cost, loosely-coupled composition. Services typically run on the hardware of their own providers, in different containers, separated by fire-walls and other ownership and trust barriers. Their composition requires additional mechanisms (e.g., process work-flow engines, connectors, or glue code) to impose some form of coordination (i.e., orchestration and/or choreography).

The holy grail of service and component-based software engineering is to develop truly reusable software services and components that can be sold off-the-shelf and reused to build software systems [88]. Research on software composition plays a key role in this quest, as it offers flexible ways of plugging together components. Some approaches to software composition use textual glue code [64, 71, 38], usually in a scripting language, whereas others offer a more visual approach, where 'channels' or 'connectors' are used to compose components into a system (e.g. [2, 80, 14]). Connectors play the role of coordinating software, yet their functionality is traditionally more limited than scripting languages. This has changed with the advent of the notion of compositional connectors [2, 64]. In such a setting, connectors are formed by composing simpler connectors, such as channels, together. Several *coordination languages* have been proposed for software composition.

Coordination languages express various coordination patterns exhibiting combinations of synchronization, mutual exclusion, non-deterministic choice, and state-dependent behavior. Some have been used as component connector models, including Reo [2], Ptolemy [58, 36], Ptolemy II [59, 36], Orc [64], MoCha [80], Manifold [7], Linda [41], BIP [15], and pipe and filter architectures [81]. Although these models overlap in philosophy and functionality, Reo is the only one that enables propagation of synchrony though composition, mutual exclusion through connectors, and combination of synchrony and asynchrony [78, 73, 77].

1.1 Quantitative analysis of systems

In recent years, there has been an increasing interest in studying the behavior of software systems from a quantitative perspective. Consider a service-based system running in a call center that matches calling clients with the appropriate representatives that can provide them with the specialized customer service that they need. Challenges that the center might face include minimizing the number of customers waiting to be matched at any point (while not having to increase their number of employees and servers too much) and improving the quality of the matching service. The relevance of being able to propose solutions for such challenges cannot be underestimated, since resources are neither infinite nor free. In addition, the answers to *quantitative* questions have to be adapted according to the context: different services have different constraints. For instance, in the context of safety critical and time critical applications (like airplane and automobile control systems), if a request is waiting for more than a few seconds there could be disastrous consequences, whereas in other applications, such as a ticket booking website, a few seconds will not have too much of a negative impact.

As mentioned above, distributed services are platform independent and, therefore, heterogeneous, in the sense that, for instance, they are written in different programming languages. In such a setting, even if the QoS properties of every individual service and connector are known, it is far from trivial to build a model for and make statements about the end-to-end QoS of a composed system. For this purpose, over the past few decades, several stochastic methods, such as Stochastic Petri Nets (SPN) [79, 65] and Stochastic Process Algebra (SPA) [63, 49, 45], have been suggested in various application areas. SPN are useful for the analysis of computer systems since they allow the system operations to be precisely described by means of a graph which then translates into a Markovian model used to obtain performance estimates. Due to its graphical representation, it can easily be understood. In addition, the derivation of the Markovian model and its solution can be automated and transparent to the users. However, as typical of state-based models, they suffer from the state-explosion problem, and often, for a large SPN model, exact solutions cannot be computed. In addition, SPN essentially deal with asynchronous events and, hence, the synchrony of events is not propagated through composition [4]. SPA, on the other hand, offers a compositional specification framework. A complicated system can be modeled by first modeling its sub-systems and then the interaction between them. The main disadvantage of SPA is the lack of expressiveness of the timing distributions that can be used in the modeling: only negative exponential distributions are allowed.

In this thesis we focus on Reo, a channel-based coordination language which provides a flexible and expressive model for compositional construction of connectors that coordinate distributed services over networks. Reo has been around for many years now and much research has been done in order to turn it into an expressive, modular and usable language. One of the main streams in this research concerns formal semantic models for Reo. There have been several proposals: a coalgebraic

model [9], colouring tables [30] which are used in the animation tool of Reo connectors in Extensible Coordination Tools (ECT) [35], and several automata models, particularly suitable for verification. Among the proposed automata models, each of which offers different expressiveness and modeling advantages, we mention Constraint Automata (CA) [12], Intentional Automata (IA) [31] and Reo automata [19]. CA are a basic and compact automaton model, which unfortunately does not support context dependency directly. Context dependency expresses behavior that depends on both the positive and negative availability of I/O requests on the boundary ports of a connector. To overcome this limitation of CA, IA and Reo automata were recently proposed. The Reo automata model is compact, quite close in spirit to the CA model, whereas the IA model is more verbose. In this thesis, we provide *quantitative* extensions of both IA and Reo automata.

First steps have been taken to extend Reo in order to accommodate QoS aspects of a system. In [5], Quantitative Reo and Quantitative Constraint Automata (QCA) were introduced. The QCA model integrates the QoS aspects of components/services and connectors that comprise an application to yield the QoS properties of that application, ignoring the impact of the environment on its performance, such as throughput and delays. QCA provide a useful model for service selection and composition [61], but, because it ignores the interaction with the environment, it does not provide a faithful model for the end-to-end QoS of a system. The latter can crucially depend not only on the internal details of a system, but also on how it is used in an environment, as determined, for instance, by the frequencies and distributions of the arrivals of I/O requests. Such stochastic aspects are not investigated in [5].

1.2 Thesis overview and contributions

The main aim of this thesis is to provide an expressive model wherein the specification of the overall end-to-end QoS of a composed service in a distributed environment can be carried out compositionally. We use as basis of our model Reo, which we extend with the power to specify stochastic aspects of a system. We provide two formal semantic models for this extension of Reo, based on the IA and Reo automata models mentioned above. Furthermore, in order to enable practical analysis of the end-to-end QoS of a system, we provide translation methods from the specification models into stochastic models (Markov Chains and Interactive Markov Chains). We have implemented all the methods presented in this thesis as plug-ins for the ECT tools [8] and, using them, we have modeled and analyzed a real application, the ASK system [83].

In Chapter 2 we mention the basic preliminaries of Reo and its semantics models. In addition, Stochastic Reo, a stochastic extension of Reo, is introduced, in which it is possible to specify the end-to-end QoS of a system. Stochastic Reo constitutes the only original contribution of this chapter and it is based on the paper:

[6] Farhad Arbab, Tom Chothia, Rob van der Mei, Sun Meng, Young-Joo Moon, and Chrétien Verhoef. From Coordination to Stochastic Models of QoS. In *COORDINATION*, volume 5521 of *Lecture Notes in Computer Science*, pages 268–287. Springer, 2009

In Chapter 3 we introduce Quantitative Intentional Automata (QIA), as a semantic model for Stochastic Reo. QIA extend the semantics of Reo by representing Reo channels and their channel ends separately and admitting annotation on them to describe data-flows through those channels and I/O request arrivals at the channel ends as stochastic events. In addition, QIA can be considered as an intermediate model for translation into stochastic models, in particular Continuous-Time Markov Chains (CTMCs), for stochastic analysis. The translation method from Stochastic Reo into CTMCs via QIA is also introduced in this chapter. This chapter is based on the following paper:

[6] Farhad Arbab, Tom Chothia, Rob van der Mei, Sun Meng, Young-Joo Moon, and Chrétien Verhoef. From Coordination to Stochastic Models of QoS. In *COORDINATION*, volume 5521 of *Lecture Notes in Computer Science*, pages 268–287. Springer, 2009

QIA can be seen as an extension of IA. In the above paper, the structure and basic definitions of QIA are different from the ones we now present in Chapter 3 since the reference for IA [31] was not available when the above paper was written. For the sake of consistency and coherence, we have completely rewritten the above paper to keep the definitions closer to the IA definitions.

In Chapter 4 we introduce Stochastic Reo Automata as an alternative semantic model for Stochastic Reo. In general, QIA have a large number of states, mainly due to the separate representation of I/O request arrivals and data-flows. Stochastic Reo Automata were designed to provide a more compact semantic model for Stochastic Reo. More importantly, Stochastic Reo Automata also enable an easy formal proof for their compositionality, which is lacking in the case of QIA. For general QoS aspects, Stochastic Reo Automata were extended with reward information to accommodate concerns such as CPU computation time and memory space. As an alternative model to QIA, Stochastic Reo Automata are also used to generate corresponding CTMCs. In addition, in this chapter, we discuss why Interactive Markov Chains (IMCs) [43] are not an appropriate semantic model for Stochastic Reo, and show the translation from Stochastic Reo into IMCs via Stochastic Reo Automata. This chapter is based on the following papers:

[68] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. A Compositional Semantics for Stochastic Reo Connectors. In *FOCLASA*, volume 30 of *EPTCS*, pages 93–107, 2010

[67] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. A Compositional Model to Reason about end-to-end QoS in Stochastic Reo Connectors. To apper in *Science of Computer Programming*, 2011

In Chapter 5 we describe the Reo2MC tool which is available as a plug-in for the ECT. Reo2MC is a fully automated tool which is able to automatically derive the QIA semantics of Reo models and their corresponding CTMCs. In addition, it provides bridges to existing third-party tools for stochastic analysis, such as PRISM[1] [57, 48], Maple, and MATLAB, by generating the input files for those tools. We also explain the usage of the Reo2MC tool. This chapter is based on the following paper:

[8] Farhad Arbab, Sun Meng, Young-Joo Moon, Marta Z. Kwiatkowska, and Hongyang Qu. Reo2MC: a tool chain for performance analysis of coordination models. In *ESEC/SIGSOFT FSE*, pages 287–288. ACM, 2009

In Chapter 6 we show a case study using the ASK system [83], an industrial software developed by the Dutch company Almende [1], and marketed by their daughter company ASK Community Systems [10]. The ASK system is a communication software product that acts as a mediator between service consumers and service providers. We model the ASK system using Stochastic Reo, and then translate the model into a CTMC in order to analyze it using PRISM. The rates used in this model were obtained by applying statistical analysis techniques on the raw values that we obtained from the real logs of an actual running ASK system. Since the translation target model is a CTMC, only exponential distributions are allowed as rates in the modeling. However, not all the distributions we obtained from the statistical analysis were exponential. In the case of properties involving rates that follow a non-exponential distribution, we also show in this chapter how to use the Reo simulator to obtain insights in the behavior of the system. This chapter is based on the following paper:

[66] Young-Joo Moon, Farhad Arbab, Alexandra Silva, Andries Stam, and Chrétien Verhoef. Stochastic Reo: a Case Study. Accepted for publication in TTSS 2011

1.2.1 Contributions

We summarize in the table below the main contributions of this thesis and the chapters where they can be found.

Stochastic Reo: a compositional model for specifying composite systems, where non-functional (QoS) aspects and the influence of the environment on their performance are taken into account.	Chapter 2
Quantitative intentional automata (QIA): an operational semantic model for Stochastic Reo	Chapter 3
Methods to translate QIA into CTMC	Chapter 3, Section 3.3

[1] http://www.prismmodelchecker.org/

Stochastic Reo Automata (SRA): an alternative compact semantic model for Stochastic Reo	Chapter 4
Methods to translate SRA into CTMC and IMC	Chapter 4, Sections 4.4 and 4.5
Formal proof of compositionality of SRA	Chapter 4, Section 4.2.1
Extension of SRA to specify more general QoS (reward information)	Chapter 4, Section 4.3
Reo2MC: a tool for the analysis of Stochastic Reo models	Chapter 5
Case study of a real commercial system, the ASK system, using the Reo2MC tool and the Reo simulator	Chapter 6

Chapter 2

Models for component coordination

In this section, we recall the basics of the Reo coordination language and its semantic models. We also present Stochastic Reo, an extension of Reo, which enables the modeling of QoS properties. In addition, we introduce the basic definitions of some stochastic models, in particular Markov Chains and Interactive Markov Chains which we will use later as target models for the translation from Stochastic Reo for performance analysis. We conclude this chapter with a brief discussion on related work.

2.1 Reo language

Reo is a channel-based coordination model wherein so-called *connectors* are used to coordinate (i.e., control the interaction among) components or services exogenously (from outside of those components and services). In Reo, complex connectors are compositionally built out of primitive channels. Channels are atomic connectors with exactly two ends. An end can be either a *source* or a *sink* end. Source ends accept data into, and sink ends dispense data out of their respective channels. Reo allows channels to be undirected, i.e., to have two source or two sink ends.

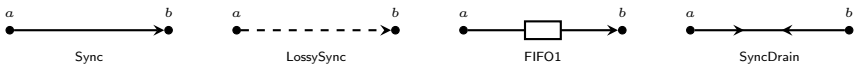


Figure 2.1: Some basic Reo channels

Figure 2.1 shows the graphical representations of some basic channel types. The Sync channel is a directed, unbuffered channel that synchronously reads data items from its source end and writes them to its sink end. The LossySync channel behaves similarly, except that it does not block if the party at the sink end is not ready to receive data. Instead, it just loses the data item. The FIFO1 is an asynchronous channel with a buffer of size one. The SyncDrain channel differs from the other channels in

that it has two source ends (and no sink end). If there is data available at both ends, this channel consumes (and loses) both data items synchronously.

Channels can be joined together using nodes. A node can have one of three types: source, sink or mixed node, depending on whether all ends that coincide on the node are source ends, sink ends or a combination of both. Source and sink nodes, called *boundary nodes*, form the boundary of a connector, allowing interaction with its environment. We assume that at most one request can wait for the acceptance at a boundary node. Source nodes act as synchronous replicators, and sink nodes as mergers. A mixed node combines both behaviors by atomically consuming a data item from one of its sink ends and replicating it to all of its source ends.

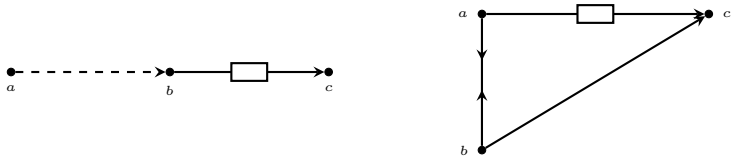


Figure 2.2: LossyFIFO1 and Ordering circuit

For example, the connectors shown in Figure 2.2 are a (overflow) LossyFIFO1 and an alternator. The LossyFIFO1 reads a data item from a, buffers it in a FIFO1 and writes to c. This connector loses data items at a if and only if the FIFO1 buffer is already full. The alternator imposes an ordering on the data from its input nodes a and b to its output node c. The SyncDrain channel enforces that data flow through a and b only synchronously. The empty buffer together with the propagation of synchrony through the three nodes guarantee that the data item obtained from b is delivered to c while the data item obtained from a is stored in the FIFO1 buffer. After this, the buffer of the FIFO1 is full and propagation of exclusion from a through the SyncDrain channel to b guarantees that data cannot flow in through either a or b, but c can dispense the data stored in the FIFO1 buffer, which makes it empty again. Assume three independent processes (that follow no communication protocol and each of which knows nothing about the others) place I/O requests on nodes a, b, and c, each according to its own internal timing. By delaying the reply to their requests, when necessary, this circuit guarantees that successive read operations at c obtain the values produced by the successive write operations at b and a alternately.

2.2 Stochastic Reo

Stochastic Reo is an extension of Reo where channels are annotated with stochastic values denoting distributions of their relevant data-flow events and arrival of I/O request at the channel ends. We refer to these distributions as processing delay rates and arrival rates of I/O requests, respectively. Such stochastic values are non-negative real values and describe the probability of a certain value (or interval) of a discrete (or

continuous) random variable. Figure 2.3 shows some primitive channels of Stochastic Reo that correspond to the primitives of Reo in Figure 2.1. In this figure and throughout, for simplicity, we do not show node names, but these names can be inferred from the names of their respective arrival rates: for instance, 'γa' refers to the node 'a'.

It should be noted that such an annotation does not affect the functionalities of Reo connectors, thus, when the annotations of rates are neglected, the mapping operational semantics between Reo and Stochastic Reo is quite straightforward, i.e., one-to-one mapping.

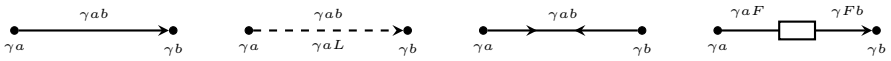


Figure 2.3: Some basic Stochastic Reo channels

A processing delay rate represents the duration that a channel takes to perform a certain activity such as transporting a data item. For instance, a LossySync has two associated variables γab and γaL for the stochastic delay rates of, respectively, successful data-flow from node a to node b, and losing the data item at node a when a read request is absent at node b. In a FIFO1, γaF means the delay for data-flow from its source node a into the buffer, and γFb means the delay for sending the data from the buffer to the sink b. Similarly, γab of a Sync (and a SyncDrain, respectively) indicates the delay for data-flow from its source node a to its sink node b (and losing data at both ends, respectively).

Arrival rates describe the time between consecutive arrivals of I/O requests at source and sink nodes of Reo channels. For instance, γa and γb in Figure 2.3 represent the associated arrival rates of write/take requests at nodes a and b. As mentioned earlier, at most one request can wait at a boundary node for acceptance. That is, if a boundary node is occupied by a pending request, then the node is blocked and consequently all further arrivals at that node are lost.

Stochastic Reo supports the same compositional framework of joins of connectors as in Reo. Most of the technical details of this *join* operation are identical to that of Reo. The nodes in Stochastic Reo have certain QoS information on them, hence joining nodes must accommodate QoS composition.

Since arrival rates on nodes model their interaction with the environment only, mixed nodes have no associated arrival rates. This is justified by the fact that a mixed node delivers data items instantaneously to the source end(s) of its connected channel(s). Thus, when joining a source with a sink node into a mixed node, their arrival rates are discarded[1].

[1]For simplicity, we assume that the activity of ideal nodes incur no delay. Any real implementation of a node, of course, induces some processing delay rate. However, such a real node can be modeled as a composition of an ideal node with a Sync channel that manifests the processing delay rate. Thus, we can even associate delay distributions with Stochastic Reo nodes and automatically translate such nodes into "Sync plus ideal node" constructs. We ignore this issue in the rest of this thesis.

The activities of a Reo connector consist of I/O request arrivals at boundary nodes, synchronization in mixed nodes, and data-flows through primitive channels. Adding time information to a connector gives rise to the causality of such activities. That is, for a given Reo connector, first I/O requests must arrive at the boundary nodes of a connector, second synchronization occurs, and finally data-flows happen. For instance, in Figure 2.4, first I/O requests arrive at a and d; second the synchronization on the mixed node b or c, selected by merger d, occurs; finally a data item is delivered from the source node a to the sink node d via the mixed node b or c.

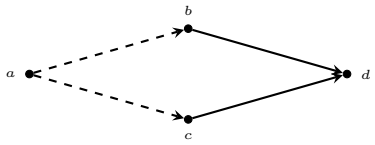


Figure 2.4: Example for the causality of a Reo connector

In order to describe the processing delay rates of a primitive channel explicitly, we name the rate by the combination of a pair of (source, sink) nodes and the buffer of the channel. For example, γab for the Sync channel and γaF for the FIFO1 channel in Figure 2.3. As mentioned in Section 2.1, a source node and a sink node act as a replicator and a non-deterministic merger, respectively, and each activity, such as replicating data to its source nodes or selecting a sink node, has its own stochastic value, the reference of which can be represented using their source and sink nodes. However, for simplicity, we do not describe the names of source and sink nodes of a replicator and a merger explicitly when the nodes are not boundary nodes. In these cases, the processing delay rates for the selection or the replication by, respectively, a merger or a replicator are not distinguishably described. Thus, we name the internal nodes of a replicator or a merger by naming after the initial name of the replicator or the merger with index. For example, merger d in Figure 2.4 has three different nodes: two source nodes and one sink node. Let the source node transmitting data from node b, the other source node, and the sink node be, respectively, d_1, d_2, and d, whereas

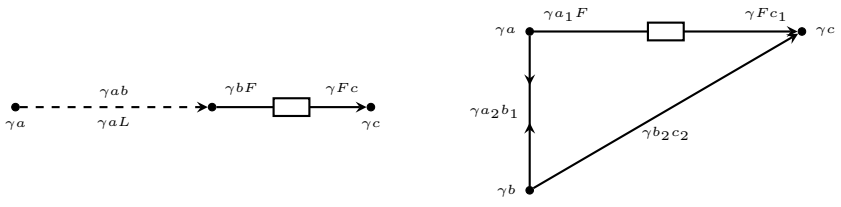


Figure 2.5: LossyFIFO1 and ordering circuit in Stochastic Reo

the first two of those distinctive names are omitted here. Then, the processing delay rates of merger d are described as $\gamma d_1 d$ and $\gamma d_2 d$ which refer to the rates for the selection of data from node b and c, respectively.

Figure 2.5 shows the LossyFIFO1 and the ordering circuit in Stochastic Reo with their stochastic values. (Compare Figure 2.2)

2.3 Semantic models for Reo

2.3.1 Constraint Automata

Constraint Automata (CA) were introduced in [12] as a formalism to capture the operational semantics of Reo, based on timed data streams, which constitute the foundation of the coalgebraic semantics of Reo [9].

We assume a finite set Σ of nodes, and denote by $Data$ a fixed, non-empty set of data that can be sent and received through these nodes via channels. CA use a symbolic representation of data assignments by data constraints, which are propositional formulas built from the atoms "$d_a \in P$", "$d_a = d_b$" and "$d_a = d$" using standard Boolean operators. Here, $a, b \in \Sigma$, d_a is a symbol for the observed data item at node a, $d \in Data$, and $P \subseteq Data$. $DC(N)$ denotes the set of data constraints can refer to the observed data items d_a at node a for $a \in N$ where $N \subseteq \Sigma$. Logical implication induces a partial order $\leq$ on DC: $g \leq g'$ iff $g \Rightarrow g'$.

A CA over the data domain $Data$ is a tuple $\mathcal{A} = (S, S_0, \Sigma, \rightarrow)$ where S is a set of states, also called configurations, $\emptyset \neq S_0 \subseteq S$ is the set of its initial states, Σ is a finite set of nodes, $\rightarrow$ is a finite subset of $\bigcup_{\emptyset \subset N \in 2^\Sigma} S \times \{N\} \times DC(N) \times S$, called the transition relation. A transition fires if it observes data items in its respective ports/nodes of the component that satisfy the data constraint of the transition, and this firing may consequently change the state of the automaton.

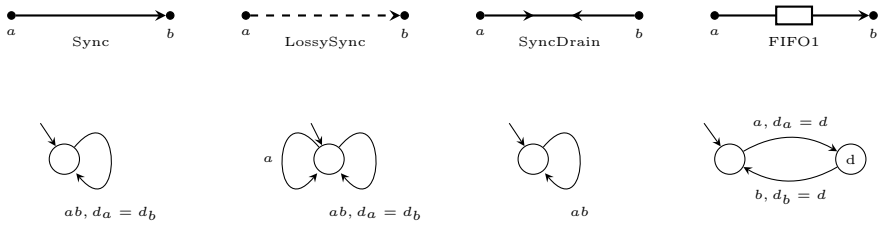


Figure 2.6: Constraint Automata for basic Reo channels of Figure 2.1

Figure 2.6 shows the CA for the primitive Reo channels in Figure 2.1. In this figure and the remainder of this thesis, the initial states are indicated with an extra incoming arrows. For simplicity, we assume the data constraints of all transitions are implicitly *true* (which simply imposes no constraints on the contents of the data-flows) and omit them to avoid clutter. In addition, we use a simplified notation for the set of nodes in

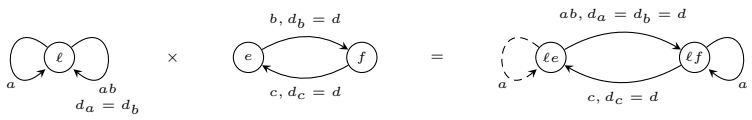
the labels of transitions by deleting the curly brackets { and } and commas between the set elements. For a full treatment of data constraints in CA, see [12].

As the counterpart for the join operation in Reo, the product of two CA $\mathcal{A}_1 = (S_1, S_{1,0}, \Sigma_1, \rightarrow_1)$ and $\mathcal{A}_2 = (S_2, S_{2,0}, \Sigma_2, \rightarrow_2)$ is defined as a constraint automaton $\mathcal{A}_1 \bowtie \mathcal{A}_2 \equiv (S_1 \times S_2, S_{1,0} \times S_{2,0}, \Sigma_1 \cup \Sigma_2, \rightarrow)$ where $\rightarrow$ is given by the following rules:

- If $s_1 \xrightarrow{N_1, g_1}_1 s_1'$, $s_2 \xrightarrow{N_2, g_2}_2 s_2'$ and $N_1 \cap \Sigma_2 = N_2 \cap \Sigma_1$, then $\langle s_1, s_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle s_1', s_2' \rangle$.
- If $s_1 \xrightarrow{N_1, g_1}_1 s_1'$ and $N_1 \cap \Sigma_2 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N_1, g_1} \langle s_1', s_2 \rangle$.
- If $s_2 \xrightarrow{N_2, g_2}_2 s_2'$ and $N_2 \cap \Sigma_1 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N_2, g_2} \langle s_1, s_2' \rangle$.

Context-dependency

The context-dependency of a Reo connector is not captured by CA. For example, recall the LossyFIFO1 example in Figure 2.2. The corresponding CA for the LossyFIFO1 is built by the product of a Sync channel ab and a FIFO1 channel bc as shown below. For simplicity, here and in the remainder of this chapter, the representations of the configurations are simplified by omitting commas between composed configurations and round brackets '(' and ')' surrounding the composed configurations.



The dashed transition from the source state ℓe is unintended because it implies that a data item is lost at node a even though the buffer is empty and able to take a data item from node a.

2.3.2 Intentional Automata

Intentional Automata (IA) [31, 32] are another semantic model for Reo, where the arrivals of I/O requests and the actual communication are described separately. Based on such characteristics, IA are useful to represent certain behavior that depends on the presence or absence of pending I/O requests in its environment/context. Thus, it can be used to specify context-dependent connectors [2] which CA cannot capture.

In general, a connector has a range of possible outputs for the same inputs from its environment. To model such a connector, throughout this thesis IA are considered to be non-deterministic even if the non-determinism is not explicitly mentioned.

Definition 2.3.1 (Intentional Automaton [31]). *An* Intentional Automaton *is a tuple* (Q, Σ, δ) *with a set of states (internal configurations)* Q*, a set of nodes* Σ*, and a transition relation* $\delta : Q \rightarrow \mathcal{P}(\mathcal{F} \times Q)^{\mathcal{R}}$ *where*

- $\mathcal{R} = \mathcal{P}(\Sigma)$ *is a set for the arrival of I/O requests, a so-called request-set, and*
- $\mathcal{F} = \mathcal{P}(\Sigma)$ *is a set for the actual communication, a so-called firing-set.*

This transition relation associates a function $\delta_q : \mathcal{R} \to \mathcal{P}(\mathcal{F} \times Q)$ *with every state* $q \in Q$*, defined by* $\delta_q(R) = \delta(q)(R)$*.* ■

Note that $\mathcal{P}(S)$ is the collection of all subsets of any set S, i.e., $\mathcal{P}(S) = 2^S$. A transition in an IA model (Q, Σ, δ) is represented as $q \xrightarrow{R|F} q'$ where $R,\ F \in \mathcal{P}(\Sigma)$ which is interpreted as $(F, q') \in \delta_q(R)$. Based on this definition, Figure 2.7 shows the IA for a Sync channel. For readability, here and in the remainder of this chapter, we simplify the representation of labels on transitions by omitting curly brackets for the sets of R and F and the commas between the elements in R and F.

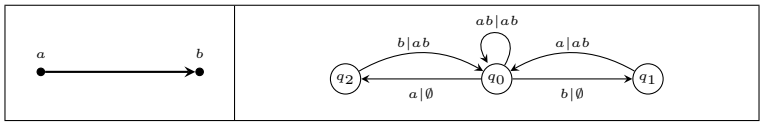


Figure 2.7: IA for a Sync channel

However, the IA only considers internal configurations of connectors. This is not enough to fully specify the behavior of Reo connectors since the behavior of a connector does not only involve its internal configuration, but also the external configuration of the system interacting with its environment. For this purpose, IA have been extended by states in $S \subseteq Q \times \mathcal{P}(\Sigma)$ where Q is the set of internal configurations of a connector and Σ is the set of nodes. Such an extension allows us to infer important invariants for the evaluation steps (transitions) of the extended IA model of a Reo connector [31, Chapter 5]:

1. a node can fire only if it either has already a pending request, or receives a request in this step;
2. when it receives a request, a node either fires the request in this step or the request becomes pending;
3. a node with a pending request, either fires it in this step or it remains pending;
4. a node has a pending request after an evaluation step only if the node receives a request and does not fire it in this step, or a request was already pending and does not fire in this step;
5. a node with a pending request is unavailable to receive requests;
6. a node that fires cannot become/remain pending.

The following formulas show these invariants formally; each formula corresponds to the invariant with the same number. For the evaluation step of the extended IA of a connector $(q, P) \xrightarrow{R|F} (q', P')$, it holds that

1. $F \subseteq R \cup P$
2. $R \subseteq F \cup P'$
3. $P \subseteq F \cup P'$
4. $P' \subseteq R \cup P$
5. $P \cap R = \emptyset$
6. $F \cap P' = \emptyset$

Here and in the remainder of this thesis, we consider the extended IA that satisfy the above invariants.

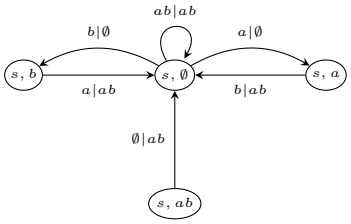
Compared to CA, the extended IA models have more states since IA consider both internal and external configurations, whereas CA only consider internal configurations. For a concise specification of the configurations of the extended IA, a listing, called an *abstract configuration table*, is used.

Definition 2.3.2 (Abstract configuration table [31]). *Given a set of internal configurations S and a set of nodes Σ, an* abstract configuration table *over S and Σ, denoted by $\theta(S, \Sigma)$, is a table such that:*

- *for each $s \in S$, there is one column labeled by s;*
- *for each $R \subseteq \Sigma$, there is one row labeled by R;*
- *at each cell of the table at the intersection of row R with column s we have a set, denoted $\theta\langle s, R\rangle$, such that $\theta\langle s, R\rangle \subseteq \mathcal{P}(\Sigma) \times (S \times \mathcal{P}(\Sigma))$, and for all $\langle F, (s', P')\rangle \in \theta\langle s, R\rangle$, we have $R = F \cup P'$ and $F \cap P' = \emptyset$.*

■

For example, Figure 2.8 shows the extended IA for a Sync channel ab and its configuration table. For readability, here and in the remainder of this chapter, we simplify the representation of the configurations by omitting brackets '()' and '{}' for, respectively, the overall configurations and the external configuration. Moreover, we delete commas between the elements in the external configuration.



	a	b
	s	
$\emptyset$	$\langle\emptyset, (s, \emptyset)\rangle$	
$\{a\}$	$\langle\emptyset, (s, \{a\})\rangle$	
$\{b\}$	$\langle\emptyset, (s, \{b\})\rangle$	
$\{a, b\}$	$\langle\{a, b\}, (s, \emptyset)\rangle$	

Figure 2.8: Extended IA for Sync ab and its configuration table θ_{Sync}

Such an abstract configuration table defines the extended IA model for a Reo connector and is, generally, more compact than its automaton model. Thus, an abstract

configuration table is used to apply other operations to its corresponding automaton model, for example, the product of the extended IA corresponding to a Reo connector is defined with abstract configuration tables (see below). The extended IA model of an abstract configuration table for a connector C is denoted by $[\![\theta_C(S,\Sigma)]\!]_R$ where S is a set of configuration and Σ is a set of nodes.

Operations

For the compositional semantics of a *join* operation in a Reo connector, the configuration tables of automata models are used. The advantage of this method, instead of using the operation of automata composition, is that it has lower computational cost, since in general, abstract configuration tables are smaller than automata models.

Definition 2.3.3 (Product of abstract configuration tables [31]). *Given two abstract configuration tables $\theta\langle S_1,\Sigma_1\rangle$ and $\theta\langle S_2,\Sigma_2\rangle$, their product abstract configuration table is*

$$\theta\langle S_1,\Sigma_1\rangle \times_T \theta\langle S_2,\Sigma_2\rangle = \theta\langle S_1\times S_2, \Sigma_1\cup\Sigma_2\rangle$$

where each cell of the table is given by: for every $R\in\mathcal{P}(\Sigma_1\cup\Sigma_2)$ and $R_i\in\mathcal{P}(\Sigma_i)$ with $i\in\{1,2\}$

$$\begin{aligned}
&\theta\langle(s_1,s_2),R\rangle = \\
&\{\ \langle F,((s_1',s_2'),P')\rangle \ \mid\ R=R_1\cup R_2,\ F=F_1\cup F_2,\ P'=P_1'\cup P_2',\\
&\qquad\qquad F_1\cap\Sigma_2=F_2\cap\Sigma_1,\ \langle F_i,(s_i',P_i')\rangle\in\theta\langle s_i,R_i\rangle,\ i=1,2\ \}\\
&\cup\{\ \langle F_1,((s_1',s_2),P')\rangle \ \mid\\
&\qquad F_1\cap\Sigma_2=\emptyset,\ R=R_1\cup R_2,\ P'=P_1'\cup R_2,\ \langle F_1,(s_1',P_1')\rangle\in\theta\langle s_1,R_1\rangle\ \}\\
&\cup\{\ \langle F_2,((s_1,s_2'),P')\rangle \ \mid\\
&\qquad F_2\cap\Sigma_1=\emptyset,\ R=R_1\cup R_2,\ P'=R_1\cup P_2',\ \langle F_2,(s_2',P_2)\rangle\in\theta\langle s_2,R_2\rangle\ \}
\end{aligned}$$

■

Note that, here and the rest of this section, $\times_T$ is used to represent the product of two abstract configuration tables, as defined in [31, Chapter 5].

The notion of equivalence $\simeq^2$ is used as a bisimilarity, defined below.

Definition 2.3.4 (Bisimulation of IA [31]). *Given two IA $\mathcal{A}_1=(Q_1,\Sigma_1,\delta_1)$ and $\mathcal{A}_2=(Q_2,\Sigma_2,\delta_2)$, a relation $\mathcal{Z}\subseteq Q_1\times Q_2$ is called a bisimulation if for $q_1\in Q_1$ and $q_2\in Q_2$, $(q_1,q_2)\in\mathcal{Z}$, then*

- $q_1 \xrightarrow{R|F}_{\delta_1} q_1'$ *implies there is a* $q_2'\in Q_2$ *such that* $q_2 \xrightarrow{R|F}_{\delta_2} q_2'$ *with* $(q_1',q_2')\in\mathcal{Z}$
- $q_2 \xrightarrow{R|F}_{\delta_2} q_2'$ *implies there is a* $q_1'\in Q_1$ *such that* $q_1 \xrightarrow{R|F}_{\delta_1} q_1'$ *with* $(q_1',q_2')\in\mathcal{Z}$

[2] In this thesis, we mention IA and Reo Automata as preliminaries. For a bisimilarity relation, the same notation $\sim$ is used for both automata models in their original literatures (IA in [31] and Reo Automata in [19]). To distinguish these two relations, in this thesis, $\simeq$ is used for the bisimilarity of IA, and $\sim$ is used for Reo Automata.

■

Two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are *bisimilar*, written $q_1 \simeq q_2$, if there exists a bisimulation relation that contains the pair (q_1, q_2). Furthermore, two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *bisimilar*, written $\mathcal{A}_1 \simeq \mathcal{A}_2$, if there exists a bisimulation relation such that every state of one automaton is related to some state of the other automaton.

Theorem 2.3.5. *[31] Given two abstract configuration tables $\theta\langle S_1, \Sigma_1\rangle$ and $\theta\langle S_2, \Sigma_2\rangle$,*

$$[\![\theta\langle S_1, \Sigma_1\rangle]\!]_R \times_I [\![\theta\langle S_2, \Sigma_2\rangle]\!]_R \simeq [\![\theta\langle S_1, \Sigma_1\rangle \times_T \theta\langle S_2, \Sigma_2\rangle]\!]_R$$

Note that $\times_I$ is used to represent the product of the extended IA models, as defined in [31, Chapter 5]. The proof of Theorem 2.3.5 is shown in [31, Chapter 5].

A hiding operation is also defined for IA on abstract configuration tables.

Definition 2.3.6 (Hiding on abstract configuration tables [31]). *Consider an abstract configuration table $\theta\langle S, \Sigma\rangle$ and a node $h \in \Sigma$. We define*

$$\exists_T[h]\theta\langle S, \Sigma\rangle = \theta[h]\langle S, \Sigma \setminus \{h\}\rangle$$

where

$$\theta[h]\langle s, R\rangle = \begin{cases} \{\langle F \setminus \{h\}, q\rangle \mid \langle F, q\rangle \in \theta\langle s, R \cup \{h\}\rangle,\ h \in F\} & \text{if non-empty} \\ \theta\langle s, R\rangle & \text{otherwise} \end{cases}$$

■

In addition, the extended IA model context-dependent connectors. For instance, the LossyFIFO1 example mentioned above is given below with the correct semantics, where a data item is lost only if the buffer is full, i.e., a loop with $a|a$ occurs in configuration ℓf.

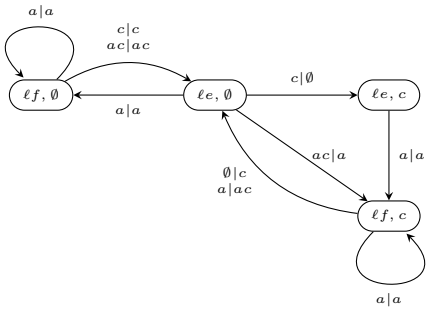


Figure 2.9: Extended IA for a LossyFIFO1 connector in Figure 2.2

2.3.3 Reo Automata

In this section, we recall Reo Automata [19], another semantic model for Reo. This model also provides a compositional operational semantics and the correct semantics for the context-dependent Reo connectors. Intuitively, a Reo Automaton is a non-deterministic automaton whose transitions have labels of the form $g|f$, where f a set of nodes that fire synchronously, and g is a *guard* (boolean condition) that represents the presence or the absence of I/O requests at nodes, i.e., the pending status of the nodes. A transition can be taken only when its guard g is true.

Compared to IA, Reo Automata provide the formal proof of their compositionality [19]. Moreover, Reo Automata are simpler and more compact, retaining the power of correctly encoding context-dependency of Reo connectors.

We recall some facts about Boolean algebras. Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a set of symbols that denote the names of connector nodes, $\overline{\sigma}$ be the negation of σ, and $\mathcal{B}_\Sigma$ be the free Boolean algebra generated by the grammar:

$$g ::= \sigma \in \Sigma \mid \top \mid \bot \mid g \vee g \mid g \wedge g \mid \overline{g}$$

We refer to the elements of the above grammar as *guards* and in their representation we frequently omit $\wedge$ and write $g_1 g_2$ instead of $g_1 \wedge g_2$. Given two guards $g_1, g_2 \in \mathcal{B}_\Sigma$, we define a (natural) order $\leq$ as $g_1 \leq g_2 \iff g_1 \wedge g_2 = g_1$. The intended interpretation of $\leq$ is logical implication: g_1 implies g_2. An *atom* of $\mathcal{B}_\Sigma$ is a guard $a_1 \ldots a_k$ such that $a_i \in \Sigma \cup \overline{\Sigma}$ with $\overline{\Sigma} = \{\overline{\sigma}_i \mid \sigma_i \in \Sigma\}$, $1 \leq i \leq k$. We can think of an atom as a truth assignment. We denote atoms by Greek letters $\alpha, \beta, \ldots$ and the set of all atoms of $\mathcal{B}_\Sigma$ by $\mathbf{At}_\Sigma$. Given $S \subseteq \Sigma$, we define $\widehat{S} \in \mathcal{B}_\Sigma$ as the conjunction of all elements of S. For instance, for $S = \{a, b, c\}$ we have $\widehat{S} \equiv abc$.

Definition 2.3.7 (Reo automaton [19]). *A* Reo Automaton *is a triple* (Σ, Q, δ) *where* Σ *is the set of nodes,* Q *is the set of states,* $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ *is the finite transition relation such that for each* $\langle q, g, f, q' \rangle \in \delta$*, which is represented as* $q \xrightarrow{g|f} q' \in \delta$*:*

(1) $g \leq \widehat{f}$ *(reactivity)*

(2) $\forall g \leq g' \leq \widehat{f} \cdot \forall \alpha \leq g' \cdot \exists q \xrightarrow{g''|f} q' \in \delta \cdot \ \alpha \leq g''$ *(uniformity)*

■

In Reo Automata, for simplicity we abstract data constraints [12] and assume they are *true*.

Intuitively, a transition $q \xrightarrow{g|f} q'$ in an automaton corresponding to a Reo connector conveys the following notion: if the connector is in state q and the boundary requests present at the moment, encoded by an atom α that is the conjunction of all possible requests presence, are such that $\alpha \leq g$, then the nodes f fire and the connector evolves to state q'. Each transition labeled by $g|f$ satisfies two criteria: (i) *reactivity* — data flow only through those nodes where a request is pending, capturing Reo's interaction model; and (ii) *uniformity* — which captures two properties: (a) the request set

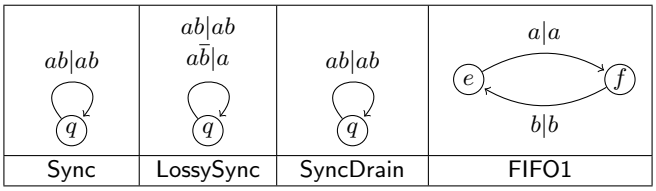


Figure 2.10: Automata for basic Reo channels of Figure 2.1

corresponding precisely to the firing set is sufficient to cause firing, and (b) removing additional unfired requests from a transition will not affect the (firing) behavior of the connector [19]. In compliance with these criteria, for a firing f, its guard g considers the presence of the least sufficient requests.

In Figure 2.10 we depict the Reo Automata for the basic channel types listed in Figure 2.1. Note that here and in the remainder of this thesis, given transition $q \xrightarrow{g|f} q'$, if there is more than one transition from a state q to the same state q' we often just draw one arrow and separate their labels by commas, and every guard in a transition label in the automata is a conjunction of literals in Σ. Moreover, it is always possible to transform any guard g into this form, by taking its disjunctive normal form (DNF) $g_1 \vee \ldots \vee g_k$ and splitting the transition $g|f$ into the several $g_i|f$, for $i = 1, \ldots, k$. Given a transition relation δ we call $norm(\delta)$ the normalized transition relation obtained from δ by putting all of its guards in DNF and splitting the transitions as explained above.

Composing Reo connectors

We now model at the automata level the composition of Reo connectors. We define two operations: product, which puts two connectors in parallel, and synchronization, which models the plugging of two nodes. Thus, the product and synchronization operations can be used to obtain the automaton of a Reo connector by composing the automata of its primitive connectors. Later in this section we formally show the compositionality of these operations.

We first define the product operation for Reo Automata. This definition differs from the classical definition of (synchronous) product for automata: our automata have disjoint alphabets and they can either take steps together or independently. In the latter case the composite transition in the product automaton explicitly encodes that one of the two automata cannot perform a step in the current state, using the following notion:

Definition 2.3.8. *[19] Given a Reo Automaton $\mathcal{A} = (\Sigma, Q, \delta)$ and $q \in Q$ we define*

$$q^\sharp = \neg \bigvee \{\ g \mid q \xrightarrow{g|f} q' \in \delta\ \}.$$

■

This captures precisely the condition under which $\mathcal{A}$ cannot fire in state q.

Definition 2.3.9 (Product of Reo Automata [19]). *Given two Reo Automata* $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ *and* $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ *such that* $\Sigma_1 \cap \Sigma_2 = \emptyset$*, we define the* product *of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *as* $\mathcal{A}_1 \times \mathcal{A}_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta)$ *where* δ *consists of:*

$$\begin{aligned}
&\{(q,p) \xrightarrow{gg'|ff'} (q',p') \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \xrightarrow{g'|f'} p' \in \delta_2\} \\
\cup\ &\{(q,p) \xrightarrow{gp^\sharp|f} (q',p) \mid q \xrightarrow{g|f} q' \in \delta_1 \wedge p \in Q_2\} \\
\cup\ &\{(q,p) \xrightarrow{gq^\sharp|f} (q,p') \mid p \xrightarrow{g|f} p' \in \delta_2 \wedge q \in Q_1\}
\end{aligned}$$

■

Here and throughout, we use ff' as a shorthand for $f \cup f'$. The first term in the union, above, applies when both automata fire in parallel. The other terms apply when one automaton fires and the other is unable to (indicated by $p^\sharp$ and $q^\sharp$, respectively). Note that the product operation is closed for Reo Automata, since according to [19], the product result preserves the properties of Reo automata, i.e., reactivity and uniformity in Definition 2.3.7. Figure 2.11 shows an example of the product of two automata.

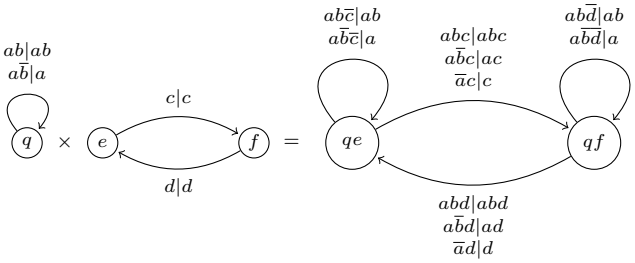


Figure 2.11: Product of LossySync and FIFO1

We now define a synchronization operation that corresponds to joining two nodes in a Reo connector. When synchronizing two nodes a and b (which are then made internal), only the transitions where either both a and b or neither a nor b fire are kept in the resulting automaton, i.e., $a \in f \Leftrightarrow b \in f$ — this is what it means for a and b to synchronize. Moreover, we keep only those transitions whose guards encode that ports a and b are not blocked. That is, transitions labeled by $g|f$ where $g \not\leq \overline{a}\overline{b}$. This condition roughly corresponds to the notion of an internal node acting like a *self-contained pumping station* [2], which implies that an internal node cannot store data nor actively block behavior.

Definition 2.3.10 (Synchronization [19]). *Given a Reo Automaton* $\mathcal{A} = (\Sigma, Q, \delta)$*, we define the* synchronization *for* $a, b \in \Sigma$ *as* $\partial_{a,b}\mathcal{A} = (\Sigma, Q, \delta')$ *where*

$$\delta' = \{q \xrightarrow{g\backslash_{ab}|f\backslash\{a,b\}} q' \mid q \xrightarrow{g|f} q' \in norm(\delta) \text{ s.t. } g \not\leq \overline{a}\overline{b} \text{ and } a \in f \Leftrightarrow b \in f\}$$

■

Here and throughout, $g\backslash_{ab}$ is the guard obtained from g by deleting all occurrences of a and b. It is worth noting that synchronization preserves reactivity and uniformity.

Synchronizing nodes b and c of the product automaton in Figure 2.11 yields the automaton depicted in Figure 2.12[3], which provides the semantics for the LossyFIFO1 example.

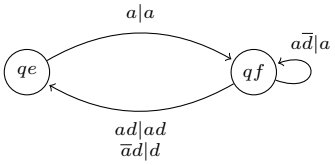


Figure 2.12: Reo Automaton for LossyFIFO1

Compositionality

Given two Reo Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the disjoint alphabet sets Σ_1 and Σ_2, $\{a_1, \ldots, a_k\} \subseteq \Sigma_1$ and $\{b_1, \ldots, b_k\} \subseteq \Sigma_2$ we construct $\partial_{a_1,b_1}\partial_{a_2,b_2}\cdots\partial_{a_k,b_k}(\mathcal{A}_1 \times \mathcal{A}_2)$ as the automaton corresponding to a connector where node a_i of the first connector is connected to node b_i of the second connector, for all $i \in \{1, \ldots, k\}$. Note that the 'plugging' order does not matter because ∂ can be applied in any order and it interacts well with product. These properties are captured in the following lemma.

Lemma 2.3.11. *[19] For the Reo Automata $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$:*

1. $\partial_{a,b}\partial_{c,d}\mathcal{A}_1 = \partial_{c,d}\partial_{a,b}\mathcal{A}_1$, *if* $a, b, c, d \in \Sigma_1$.
2. $(\partial_{a,b}\mathcal{A}_1) \times \mathcal{A}_2 \sim \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$, *if* $a, b \notin \Sigma_2$ $\Sigma_1 \cap \Sigma_2 = \emptyset$.

♦

The notion of equivalence $\sim$ used above is bisimilarity, defined as follows.

Definition 2.3.12 (Bisimulation [19]). *Given the Reo Automata $\mathcal{A}_1 = (\Sigma, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, \delta_2)$, we call $R \subseteq Q_1 \times Q_2$ a* bisimulation *iff for all $(q_1, q_2) \in R$:*

If $q_1 \xrightarrow{g|f} q_1' \in \delta_1$ and $\alpha \in \mathcal{B}_\Sigma$, $\alpha \leq g$, then there exists a transition $q_2 \xrightarrow{g'|f} q_2' \in \delta_2$ such that $\alpha \leq g'$ and $(q_1', q_2') \in R$ and vice-versa. ■

[3]For simplicity, we abstract away data-constrains on firings by assuming them true. Thus, the composition result of a LossySync and a FIFO1 channels, i.e., an overflow LossyFIFO1 circuit, becomes indistinguishable from the automaton for a shift LossyFIFO1 [12] circuit. However, by reviving data-constraints we can distinguish the automata for these two circuits.

We say that two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are bisimilar if there exists a bisimulation relation containing the pair (q_1, q_2) and we write $q_1 \sim q_2$. Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there exists a bisimulation relation such that every state of one automaton is related to some state of the other automaton.

2.4 Markov Chains

Stochastic processes are used for modeling random phenomena as transition systems with probability distributions for the outgoing transitions of a state. Markov Chains (MCs) are a special case of such stochastic processes, which satisfy

1. *discrete state space* which implies that their state space is countable and
2. *Markov property* which implies that the state change from a current state depends on only the current state, not on the history, i.e., the sequence of visited states.

Such state change in MCs can be considered with or without taking into account the time instance when the change occurs. In case that the state change is independent of the time instance, MCs are said to be *homogeneous*. The time homogeneity in stochastic processes gives us the freedom for a certain event to occur at any time instance. In the other case, it is called *inhomogeneous*, which gives much flexibility for specifying system behavior.

In addition, the Markov property requires that the waiting time (i.e., sojourn time) satisfies *memoryless property*: at time instance t, the remaining time before leaving a state is independent of the time already spent in that state.

According to the time domains, MCs are categorized into two classes: Discrete-Time Markov Chains (DTMCs) and Continuous-Time Markov Chains (CTMCs). To satisfy the memoryless property in respective time domains, the geometric distribution and the exponential distributions are necessary for DTMC and CTMC, respectively.

With these conditions, MCs can be seen as relatively simple stochastic processes. Nonetheless, MCs are frequently used to model various probabilistic systems. Moreover, its simplicity yields efficient algorithms [85] for numerical analysis.

Here and in the remainder of this thesis, we deal only with homogeneous MCs, especially homogeneous CTMCs, even though we do not mention the homogeneity of MCs explicitly.

Continuous-Time Markov Chains

A Continuous-Time Markov Chain (CTMC) is a discrete-state Markov process with continuous time domain, $\{X(t)|t \geq 0\}$, which can be used to model and analyze random system behavior. $X(t) \in S$ denotes the state in a given state space S at time t. Let $\mathbf{P}\{X(t) = i\}$ be the probability that the process is in state i at time t. The stochastic process $X(t)$ is a homogeneous CTMC if, for ordered times $t_0 < \cdots <$

$t_n < (t_n + \Delta t)$, the conditional probability of staying in any state j satisfies:

$$\mathbf{P}\{X(t_n + \Delta t) = j \mid X(t_n) = i_n, X(t_{n-1}) = i_{n-1}, \cdots, X(t_0) = i_0\} = \mathbf{P}\{X(t_n + \Delta t) = j \mid X(t_n) = i_n\}.$$

Briefly, the probability that the process is in *future* state j depends on only the *current* state i_n, not the past states.

The sojourn time in any state of a CTMC model must be *exponentially distributed* since the exponential distributions are the only class that satisfies the memoryless property in continuous time domain. Below we list the properties of the exponential distributions that are relevant to our work.

- An exponential distribution $P\{delay \leq t\} = 1 - e^{-\lambda t}$ is characterized by a positive real value λ, the so-called *rate* of the distribution. Its mean duration is $1/\lambda$ time units.
- While satisfying the memoryless property, the remaining delay after some time t_0 has elapsed is also exponentially distributed:

$$P\{delay \leq t + t_0 \mid delay > t_0\} \;=\; P\{delay \leq t\}$$

- Exponential distributions are closed under minimum which is the sum of the rates:

$$P\{min(delay_1, delay_2)\} = 1 - e^{-(\lambda_1 + \lambda_2)t}$$

  where λ_1 and λ_2 are the rates of the distributions $delay_1$ and $delay_2$, respectively.
- The probability that $delay_1$ with the rate λ_1 is smaller than $delay_2$ with the rate λ_2 is

$$P\{delay_1 < delay_2\} = \frac{\lambda_1}{\lambda_1 + \lambda_2}$$

- In the continuous-time domain, the probability that two delays elapse at the same time is *zero*.

Such properties of exponential distributions state that the probability to stay in a state decreases as time elapses, i.e., a transition emanating from a certain state will be triggered eventually. When a certain state has more than one possible leaving transitions, the transition will be triggered proportional to its rate.

2.5 Interactive Markov Chains

Interactive Markov Chains (IMCs) [43] are a stochastic model to specify reactive systems. In IMCs, timing information and actions are represented separately. Timing information is described by *Markovian transitions*, and actions are described by *interactive transitions*. Roughly speaking, IMCs are a combination of Labeled Transition Systems (LTSs) and CTMCs.

An IMC is formally described as a tuple $(S, Act, \rightarrow, \Rightarrow, s_0)$ where S is a finite set of states; Act is a set of actions; s_0 is an initial state in S; $\rightarrow$ and $\Rightarrow$ are two types of transition relations:

- $\rightarrow \subseteq S \times Act \times S$ for *interactive* transitions and
- $\Rightarrow \subseteq S \times \mathbb{R}^+ \times S$ for *Markovian* transitions.

Thus, an IMC is an LTS if $\Rightarrow = \emptyset$ and $\rightarrow \neq \emptyset$, and is a CTMC if $\Rightarrow \neq \emptyset$ and $\rightarrow = \emptyset$.

Compared to other stochastic models such as CTMCs, the main strength of IMCs is their compositionality. Thus, one can generate a complex IMC as the composition of relevant simple IMCs, which enables compositional specification of complex systems.

Definition 2.5.1. ***(Product of IMCs [43])*** *Given two IMCs* $\mathcal{I}_1 = (S_1, Act_1, \rightarrow_1, \Rightarrow_1, s_{(1,0)})$ *and* $\mathcal{I}_2 = (S_2, Act_2, \rightarrow_2, \Rightarrow_2, s_{(2,0)})$*, the composition of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *with respect to a set* A *of actions is defined as* $\mathcal{I}_1 \times \mathcal{I}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \Rightarrow, s_{(1,0)} \times s_{(2,0)})$ *where* $\rightarrow$ *and* $\Rightarrow$ *are defined as:*

$$
\begin{array}{rcll}
\rightarrow & = & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2') \mid \alpha \in A,\ s_1 \xrightarrow{\alpha}_1 s_1' \ \wedge\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
 & \cup & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2) \mid \alpha \notin A,\ s_2 \in S_2,\ s_1 \xrightarrow{\alpha}_1 s_1'\} \\
 & \cup & \{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2') \mid \alpha \notin A,\ s_1 \in S_1,\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
\Rightarrow & = & \{(s_1, s_2) \overset{\lambda}{\Rightarrow} (s_1', s_2) \mid s_2 \in S_2\ ,\ s_1 \overset{\lambda}{\Rightarrow}_1 s_1'\} \\
 & \cup & \{(s_1, s_2) \overset{\lambda}{\Rightarrow} (s_1, s_2') \mid s_1 \in S_1\ ,\ s_2 \overset{\lambda}{\Rightarrow}_2 s_2'\}
\end{array}
$$

■

The product of interactive transitions is similar to ordinary automata product, which includes interleaving and synchronized compositions of interactive transitions. The product of Markovian transitions consists of only interleaved transitions.

Compared to CTMCs, IMCs can represent not only exponential distributions, but also non-exponential distributions, especially phase-type distributions. The analysis of IMCs is supported by tools such as the Construction and Analysis of Distributed Processes (CADP) [40]. CADP verifies the functional correctness of the specification of system behavior and also minimizes IMCs efficiently [39]. Moreover, IMCs can be used in various other applications, such as Dynamic Fault Trees (DFTs), Architectural Analysis and Design Language (AADL), and so on [44].

2.6 Related work

2.6.1 Other coordination languages

Orc [64] is a theory of *orchestration* of *sites* which are considered as basic services distributed over a network. In Orc, each connection between sites takes place highly asynchronously and performs only once. While performing the connection, the orchestrator (Orc expression) initiates its connections dynamically. Such dynamics enables

to deal with failures in sites well. Compared to Orc, the connection in Reo is static, as it is based on the assumption that components communicate continuously. Recently, the research on the dynamic reconfiguration of Reo connectors has been initiated in [53]. In addition, Reo is highly synchronous, thus, it can specify the propagation of synchrony and mutual exclusion through Reo connectors. More detailed comparison is provided in [78].

Linda [41] is the first coordination language that describes the communication between different processes by exchanging data. In Linda, data objects are referred to as *tuples*, and communicating data takes place in a *shared tuple-space*. Communication actions in the shared tuple-space can occur atomically, and interactions occur in an interleaved way. That is, Linda does not handle the propagation of synchrony which is supported by Reo.

BIP [15] (an acronym of *Behavior*, *Interaction*, and *Priority*) is a methodology for modeling heterogeneous real-time components and their composition. The composition in BIP happens in three different layers, viz. that of behavior, interaction, and priority. The lower layer, an atomic component, describes its behavior; the intermediate layer specifies possible interactions between atomic components; the upper layer presents the priority relation to select amongst possible interactions. Compared to Reo, the priority relation in BIP is the main difference. This priority is used to explicitly consider the scheduling the connection between components. Whereas, in Reo, the scheduling/selection aspects is decided non-deterministically randomly by each merger.

These coordination languages have been proposed to model the composition of distributed system over a network. Each of them has its own features to specify some situations in the composition. However, Reo is the only coordination language that supports global synchronization (the propagation of synchrony), mutual exclusion through connectors, and the combination of synchrony and asynchrony.

2.6.2 Continuous-Time Constraint Automata

Continuous-Time Constraint Automata (CCA) [13] are a stochastic extension of CA that support reasoning about QoS aspects such as expected response times. CCA are close to IMCs in that they distinguish between interactive transitions and Markovian transitions:

- *interactive transitions* $p \xrightarrow{N,g} q$ as an ordinary transition in CA and
  - hidden transitions if $N = \emptyset$
  - visible transitions otherwise
- *Markovian transitions* $p \xrightarrow{\lambda} q$ where $\lambda \in \mathbb{R}^+$, called the rates of distributions.

In CCA, data-flows in connectors are represented by interactive transitions since the synchrony and the asynchrony of data-flows can be captured by the ordinary CA transitions. Processing data in components is represented by Markovian transitions

since processing data in each component is independent of the processing in the others, and each processing occurs concurrently.

CCA can be used to specify the interaction of components and connectors that connect the components, as well as to reason about some QoS aspects of the connectors such as the average processing time of I/O requests in a certain component. Moreover, CCA support both *non-deterministic choice* and *probabilistic choice*. When a current state has one or more outgoing hidden (invisible interactive) transitions, one of the outgoing interactive transitions from the state is chosen non-deterministically. When there is no outgoing hidden transitions from a current state, then one outgoing Markovian transition from the current state is chosen probabilistically and fires.

The stochastic extension in CCA focuses on internal behavior of a connector, but it does not take into account the interaction with the environment, i.e., the arrivals of I/O requests at the boundary nodes of a connector are not considered as stochastic processes. Reasoning about the end-to-end QoS of systems requires incorporation of this external behavior. In addition, CCA do not capture the context-dependency of a Reo connector since interactive transitions in CCA merely follow CA transitions that do not formalize the context-dependency of Reo connectors. Compared to such CCA, the specification models in this thesis, Quantitative Intentional Automata and Stochastic Reo Automata (See Chapter 3 and Chapter 4, respectively), not only specify the end-to-end QoS of a Reo connector, but also capture context-dependent behavior.

2.6.3 Stochastic Process Algebra

Process Algebra (PA) [63, 49, 11] is a compositional specification formalism of algebraic nature for concurrent systems. It describes interactions, communications, and synchronizations between processes in a system. PA provides a compositional approach, where a system is modeled by a collection of subsystems called *agents* that execute atomic *actions*. These actions describe communications between agents and sequential behavior that may run concurrently.

Stochastic Process Algebra (SPA) [45] is a stochastic extension of PA, which integrates Process Algebra theory and stochastic processes. SPA is described by three parts: *actions* that model the system activities, *algebraic operators* that compose the subsystem specifications, and *synchronization discipline*. An action in SPA consists of an action type a and its exponential rate λ, i.e., $\langle a.\lambda \rangle$. Several algebraic operators are shown below:

name	expression	denotation
prefix	$\langle a.\lambda \rangle.E$	After action a with a rate λ, the agent becomes E.
abstraction	E/L	The actions in L are hidden.
relabeling	$E[a_1/a_0, \dots]$	The label a_1 is renamed a_0.
choice	$E_1 + E_2$	The agent behaves either E_1 or E_2.
parallel composition	$E_1 \| \| E_2$	The agents E_1 and E_2 proceed in parallel.

There are several synchronized solution disciplines for the rate of the synchronized (shared) actions, and different solutions yield various SPA formalisms such as Performance Evaluation Process Algebra (PEPA) [46, 47], Extended Markovian Process Algebra (EMPA) [17, 16]. In PEPA, it is assumed that each agent has a *bounded capacity* to carry out activities of any particular type, determined by the rate that is the sum of the rates of each action enabled in that agent. That is, an agent cannot exceed its boundary capacity, thus the rate of a synchronized action is the minimum of the rates of the agents involved. In EMPA, it is assumed that in a synchronization, at most one participant in the synchronization has an explicit representation for the rate of the resulting (synchronized) action.

SPA has the following benefits:

- to support a compositional specification, i.e., given a complicated system, modeling its sub-systems and the interaction between the sub-systems
- clear structure and semantics
- model reuse and maintaining a library of models

The limitation of SPA is the lack of expressiveness with respect to its timing distribution: only negative exponential distributions can be used. To make the SPA more general, some work has been carried out by associating general distributions with the actions of a model [52].

The operational semantic model of SPA is defined by means of a labeled transition model. Because of interleaving, the semantic model of SPA suffers from the state explosion problem. Research has been carried out to mitigate this problem in [26, 62, 42].

SPA describes '*how*' each process behaves, whereas, (Stochastic) Reo directly describes '*what*' communication protocols connect and '*how*' they coordinate the processes in a system, in terms of primitive channels and their composition. Therefore, (Stochastic) Reo explicitly models the pure coordination and communication protocols including the impact of real communication networks on software systems and their interactions.

2.6.4 Stochastic Petri Nets

Petri Nets (PNs) [74, 79] are graphical and mathematical models that describe system behavior with concurrency, asynchrony, and synchrony. As a graphical model, PN is similar to flow charts, block diagrams, and networks. As a mathematical model, it is used to set up state equations, algebraic equations, and so on.

Stochastic Petri Nets (SPNs) [65, 87, 60] are a stochastic extension of PN, by associating an exponentially distributed firing time with each transition in a PN. The reachability set of an SPN model is identical to the one of its underlying PN model, thus, the structural properties obtained for PN, such as liveness, boundness, conservativeness, repetitiveness, consistency, and controllability, are still valid for SPNs.

The countability of the markings and the memoryless property of exponential distributions allow an isomorphism between SPN models and CTMC models. Thus, the CTMC model corresponding to an SPN is obtained by constructing the reachability graph of the SPN model and by labeling its arcs with the firing rates of each transition that changes markings.

Such an SPN is a useful tool for the analysis of computer systems since it allows the system operations to be described precisely by means of a graph that translates into a Markovian model useful for obtaining performance estimates. Due to its graphical representation, an SPN can be easily understood. In addition, the derivation of the MC model and its solution can be made automatic, and transparent to the users.

However, as for state-based models, they in general suffer from the state-explosion problem, the graphical representation of an SPN causes fast increasing complexity in their numerical solution as the system size increases. Thus, a large SPN model is often used for simulation. In addition, a PN essentially deals with asynchronous events and does not propagate the synchrony of events, thus, its compositionality is not clear in general [4].

The topology of connectors in (Stochastic) Reo is inherently dynamic, and it accommodates mobility as described in [56]. Moreover, (Stochastic) Reo supports a liberal notion of channels, which allows to express synchrony and asynchrony. Reo is more general than data-flow models and PNs [4], which can be viewed as specialized channel-based models that incorporate certain built-in primitive coordination constructs.

2.6.5 Stochastic Automata Networks

A Stochastic Automata Network (SAN) [86, 37] specifies a system consisting of a number of individual Stochastic Automata. Each Stochastic Automaton runs independently or synchronously with the others. The rates on the transitions of a SAN are either constants or functions:

- constants, i.e., non-negative real numbers
- functions from the global state space to non-negative real numbers

Normally an automaton makes use of both kinds of transitions for modeling.

In general, events in each automaton are categorized into two different types of independent and synchronized events. In the case of independent events in a SAN, the effect of the constants or functional transitions is local, thus, all the information relevant to the transitions in a Stochastic Automaton is handled in that automaton with the assumption that the automaton has a knowledge of the global state space.

In the case of synchronized events, the effect of the transitions is global by altering the state of a number of Stochastic Automata.

SAN is used for performance modeling related to parallel distributed systems. Parallel and distributed systems can be seen as collections of components that interact with each other. Thus, each component corresponds to an individual Stochastic Automaton and the overall system corresponds to a collection of such automata.

However, SAN is a state-base model, where potentially the state explosion problem arises. To mitigate this problem, techniques to minimize the number of states have been suggested. For this purpose, in SAN, it is possible to make use of symmetries as well as lumping and various superpositioning of automata [27, 82]. In addition, SAN does not store nor generate the (global) state transition matrix. Instead of that, it is represented by a number of small matrices relevant for each Stochastic Automaton. Thus, a SAN approach has minimal memory requirements.

Compared to (Stochastic) Reo, the interactions in SAN are rather limited for patterns like synchronizing events. The representation of synchronized events requires an appropriate transition label that consists of a transition probability and an alternative probability. A transition probability must be unique for the synchronized events; an alternative probability is different for each individual automaton involved in the synchronized event [75].

Chapter 3

Quantitative Intentional Automata

3.1 Introduction

In Service-oriented Computing (SOC), services distributed over a network are composed according to the requirements of service consumers. Services are platform – and network – independent applications that support rapid, low-cost, loosely-coupled composition. Services run on the hardware of their own providers, in different containers, separated by firewalls and other ownership and trust barriers. Their composition requires additional mechanisms (e.g., process work-flow engines, connectors, and glue code) to impose some form of coordination (i.e., orchestration and/or choreography). Even if the quality of service (QoS) properties of every individual service and connector are known, it is far from trivial to build a model for and make statements about the end-to-end QoS of a composed system.

In CA, Reo Automata, and IA, mentioned in Chapter 2, the end-to-end QoS is not considered along with the specification of system behavior. In order to specify and reason about the end-to-end QoS of system behavior, Stochastic Reo was also introduced in Chapter 2. As the name reveals, Stochastic Reo is a stochastic extension of Reo and preserves the flexibility and the expressiveness of Reo for compositional construction of connectors. The aim of this chapter is to introduce a semantic model for Stochastic Reo.

This chapter consists of two parts. the first part introduces a semantic model for Stochastic Reo, *Quantitative Intentional Automata (QIA)* [6]. Actually, this semantic model is a stochastic extension of Intentional Automata (IA) [31]. We show the mapping between primitive Stochastic Reo channels and their corresponding QIA, as well as other operations such as the product.

The second part shows the translation from QIA into homogeneous CTMCs which are simple stochastic processes, widely used with efficient algorithms [85] for stochastic analysis.

3.2 Quantitative Intentional Automata

In this section, we introduce the notion of Quantitative Intentional Automata (QIA) which is designed as a stochastic extension of IA to provide an operational semantics for Stochastic Reo. Existing semantic models for Reo include IA[1] and CA. Whereas CA transitions describe system configuration changes, transitions of IA (and QIA) describe both the changes of system configuration as well as the changes of pending I/O requests. In CA, configurations are stored in the states, and processes causing state changes are shown in transition labels as a set of nodes where data are observed. Similarly, in IA (and QIA), the configurations of a system and the pending I/O requests are stored in the states. A data-flow or a firing through nodes causes changes in the system configuration, and arrivals of I/O requests at the nodes change the configurations of the pending I/O requests. These two different types of changes are distinguishably represented. (See Definition 3.2.1.)

Definition 3.2.1 (Quantitative Intentional Automaton). *A Quantitative Intentional Automaton is a tuple* $(Q, I, \Sigma, \rightarrow, \mathbf{r})$ *where*

- $Q \subseteq L \times 2^{\Sigma}$ *is a finite set of states, where*
  - L *is a finite set of system configurations.*
  - 2^{Σ} *is a set of pending node sets, each element in* 2^{Σ} *describes the pending status in the current state.*
- $I \subseteq Q$ *is a set of initial states.*
- Σ *is a finite set of nodes.*
- $\rightarrow \subseteq Q \times 2^{\Sigma} \times 2^{\Sigma} \times 2^{\Theta} \times Q$ *is the transition relation where* $\Theta \subseteq 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^{+}$ *such that for any* $I, O \subseteq \Sigma$ *and* $I \cap O = \emptyset$*, each* $(I, O, r) \in \Theta$ *corresponds to a data-flow where* I *is a set of mixed and/or input nodes;* O *is a set of output and/or mixed nodes; and* r *is a processing delay rate for the data-flow described by* I *and* O*. We require that*
  - *for any two 3-tuples* (I_1, O_1, r_1), $(I_2, O_2, r_2) \in \Theta$ *such that* $I_1 = I_2 \wedge O_1 = O_2$*, it holds that* $r_1 = r_2$*, and*
  - *for a transition* $s \xrightarrow{R,F,D} s' \in \rightarrow$ *with* $D = \{(I_1, O_1, r_1), \ldots, (I_n, O_n, r_n)\}$, $F \setminus (I \cap O) = (I \cup O) \setminus (I \cap O)$ *where* $I = \bigcup_{1 \leq i \leq n} I_i$ *and* $O = \bigcup_{1 \leq i \leq n} O_i$.
- $\mathbf{r} : \Sigma \rightarrow \mathbb{R}^{+}$ *is a function that associates with each node its arrival rate.*

■

[1] IA is a general semantic model for component connectors. For a Reo connector, some invariants [31, Chapter 5] are required. Here and in the remainder of this chapter, IA are considered to be the extended version of IA, with the invariants, even though this will not be explicitly mentioned.

Note that for simplicity, here and throughout, we assume that all data constraints for transitions are *true* and thus abstract away the data constraints without loss of generality. In fact, the data constraints are used only for the nodes that fire, and they can be obtained from the transition with the same firing nodes in the corresponding CA of a QIA. In case of I/O request arrivals, there are no specific constraints on incoming data items, thus, the data constraints for I/O request arrivals are always *true*. In [31], for IA, a function $Q \to \mathcal{P}(2^{\Sigma} \times 2^{\Theta} \times Q)^{2^{\Sigma}}$ is used as an alternative.

In the QIA model for a Stochastic Reo connector, a transition $\langle q_1, R, F, D, q_2 \rangle$ is represented as $q_1 \xrightarrow{R,F,D} q_2$ where:

- R is the set of nodes that interact with the environment of the nodes;
- F is the set of nodes that fire and are released by the data-flows of the transition;
- $D \subseteq \Theta$ is the set of 3-tuples $\theta = (I, O, r)$ that correspond to individual data-flows in primitive Reo channels; the first two elements in θ depict the structural information of the relevant channel ends in the data-flow corresponding to θ, e.g., input and output nodes for the data-flow; the last element shows the processing delay rate of the data-flow.

In case a transition corresponds to only request-arrivals, $D = \emptyset$. The arrival rates for I/O requests are given by the function $\mathbf{r}$. For instance, a QIA transition

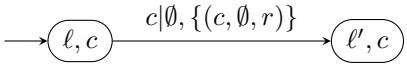
$$\langle \ell, \{a, b\} \rangle \xrightarrow{\emptyset,\{a,b\},\{(\{a\},\{b\},\gamma ab)\}} \langle \ell', \emptyset \rangle$$

encodes that a data-flow occurs from source node a to sink node b with the processing delay rate γab. Each element of a 3-tuple $\theta \in \Theta$ is accessed, respectively, by the projection functions $i : \Theta \to 2^{\Sigma}$, $o : \Theta \to 2^{\Sigma}$, and $v : \Theta \to \mathbb{R}^{+}$. Note that for readability, here and in the rest of this chapter, we use simplified representations for all sets used to describe QIA, such as R, F, and D in a QIA transition. The curly brackets { and } for these sets are deleted, and the elements in the sets are arranged without commas. In addition, R and F are distinguished by a vertical bar '|', i.e., the above transition is represented as

$$\langle \ell, ab \rangle \xrightarrow{\emptyset|ab,\{(a,b,\gamma ab)\}} \langle \ell', \emptyset \rangle.$$

3.2.1 Invariants

Consider the following automaton, with two states.



This automaton satisfies Definition 3.2.1. However, it does not capture a behavior that corresponds to the semantics for a Reo connector. The reason is that node c is

already pending (as indicated by the presence of c in the configuration of the source state of the transition) and blocked to further request arrivals, therefore, it cannot interact with the environment as indicated by the new request arrival at node c on the transition. The correct behavior of a Reo connector is subject to the same invariants mentioned for IA in Section 2.3.2. We recall these invariants. For an IA transition $\langle q, P\rangle \xrightarrow{R|F,D} \langle q', P'\rangle$, it is required that:

1. $F \subseteq R \cup P$
2. $R \subseteq F \cup P'$
3. $P \subseteq F \cup P'$
4. $P' \subseteq R \cup P$
5. $P \cap R = \emptyset$
6. $F \cap P' = \emptyset$

These invariants are also used for a QIA transition $\langle \ell, P\rangle \xrightarrow{R|F,D} \langle \ell', P'\rangle$, since in QIA, the function $\mathbf{r}$ and the set of 3-tuples D in transition labels do not affect the structure on the transitions of QIA, which are decided by F.

The intuitive meaning of these invariants is explained in Section 2.3.2. Based on Definition 3.2.1 and these invariants, the appropriate QIA, as a semantic model for Stochastic Reo, corresponding to the primitive Stochastic Reo channels are presented in Figure 3.1. Note that here and the remainder of this chapter, for simplicity, when a set of 3-tuples is empty, i.e., transitions correspond to request-arrivals, we abstract it away. That is, only firing transitions include a set of relevant 3-tuples. The function $\mathbf{r}$ is shown as tables in Figure 3.1.

3.2.2 QIA composition

As mentioned in Section 2.2, a Stochastic Reo connector is obtained by composing primitive channels. Similarly, the QIA corresponding to a connector is also obtained by the composition of the QIA of its respective primitive channels. This composition is carried out in two operations:

1. **product** that plugs two automata together, considering synchronization and interleaving of the transitions from each automaton, and
2. **synchronization** that makes mixed nodes internal and filters firing transitions, taking into account the context-dependency of a Reo connector.

The QIA product is similar to the IA product in [31, Chapter 5]. However, this IA product does not account for the status of pending requests. In addition, we need to define how the QIA product handles the extended elements of the function $\mathbf{r}$ and the sets of 3-tuples in the transition labels of QIA. The function $\mathbf{r}$ associates arrival rates with the nodes in the set of nodes only, thus, it does not influence the structure of QIA. The set of 3-tuples in transition labels depends on firings. However, the converse is not true, i.e., the firing is not decided by the set of 3-tuples. In general, the product operation decides if firings are either composed together or interleave according to the synchronization constraints of the firings. Therefore, the function $\mathbf{r}$ and the set of 3-tuples do not affect the product definition, which as usual (e.g., CA and IA) primarily depends on firings. We define the QIA product considering synchrony of firings as follows:

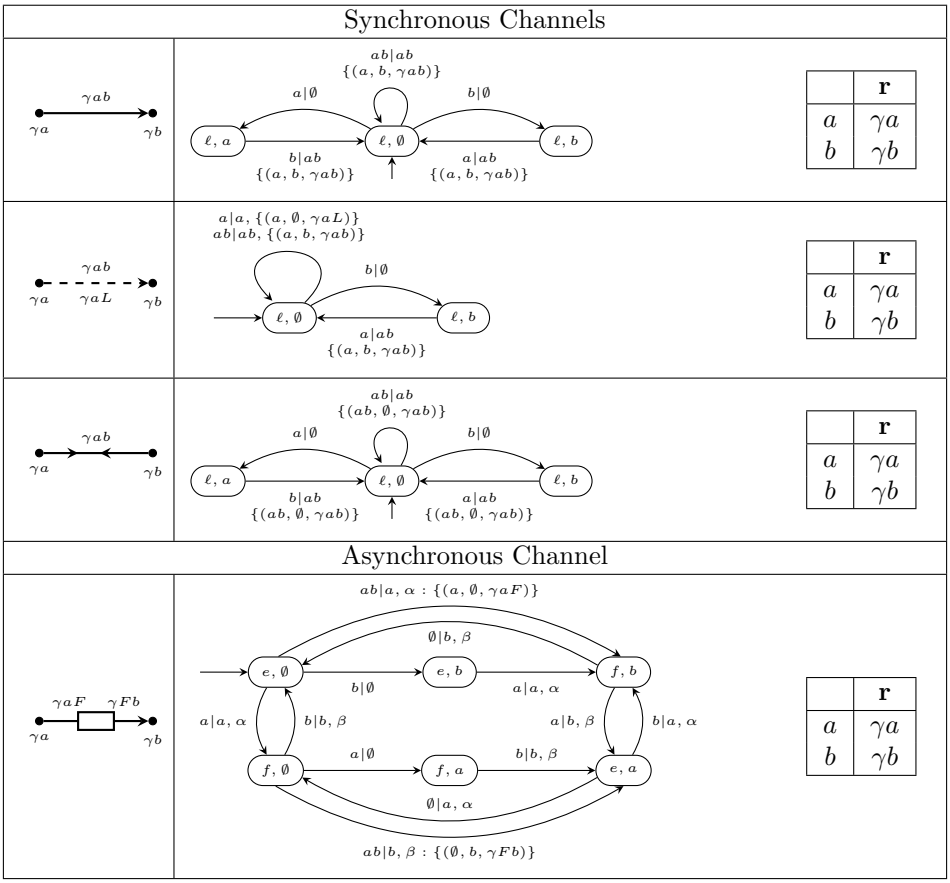


Figure 3.1: QIA for channels of Figure 2.3

Definition 3.2.2 (Product of QIA). *Given two QIA* $\mathcal{A} = (Q_1, I_1, \Sigma_1, \rightarrow_1, \mathbf{r}_1)$ *and* $\mathcal{B} = (Q_2, I_2, \Sigma_2, \rightarrow_2, \mathbf{r}_2)$*, their product is defined as* $\mathcal{A} \bowtie \mathcal{B} = (Q_1 \times Q_2, I_1 \times I_2, \Sigma_1 \cup \Sigma_2, \rightarrow, \mathbf{r}_1 \cup \mathbf{r}_2)$ *where* $\rightarrow$ *is given by following rules:*

1. $$\frac{\langle \ell_1, P_1\rangle \xrightarrow{R_1,F_1,D_1}_1 \langle \ell_1', P_1'\rangle \qquad F_1 \cap \Sigma_2 = \emptyset \qquad \forall \langle \ell_2, P_2\rangle \in Q_2 \text{ s.t. } P_2 \cap R_1 = \emptyset}{\langle (\ell_1, \ell_2), P_1 \cup P_2\rangle \xrightarrow{R_1,F_1,D_1} \langle (\ell_1', \ell_2), P_1' \cup P_2\rangle}$$

2. $$\frac{\langle \ell_2, P_2\rangle \xrightarrow{R_2,F_2,D_2}_2 \langle \ell_2', P_2'\rangle \qquad F_2 \cap \Sigma_1 = \emptyset \qquad \forall \langle \ell_1, P_1\rangle \in Q_1 \text{ s.t. } P_1 \cap R_2 = \emptyset}{\langle (\ell_1, l_2), P_1 \cup P_2\rangle \xrightarrow{R_2,F_2,D_2} \langle (\ell_1, \ell_2'), P_1 \cup P_2'\rangle}$$

3. $$\frac{\langle \ell_1, P_1\rangle \xrightarrow{R_1,F_1,D_1}_1 \langle \ell_1', P_1'\rangle \quad \langle \ell_2, P_2\rangle \xrightarrow{R_2,F_2,D_2}_2 \langle \ell_2', P_2'\rangle \quad F_1 \cap \Sigma_2 = F_2 \cap \Sigma_1 \ \wedge \ R_1 \cap P_2 = \emptyset = R_2 \cap P_1}{\langle (\ell_1, \ell_2), P_1 \cup P_2\rangle \xrightarrow{R_1 \cup R_2, F_1 \cup F_2, D_1 \cup D_2} \langle (\ell_1', \ell_2'), P_1' \cup P_2'\rangle}$$

■

Request-arrivals and data-flows are representative activities of Reo connectors. Request-arrivals are independent of synchrony or asynchrony of connector behavior, thus, the product result of transitions for request-arrivals interleave. On the other hand, data-flows are influenced by the synchrony or asynchrony of the behavior. Therefore, the product operation needs to consider the set of firing nodes, for example, $F_1 \cap \Sigma_2 = \emptyset$, $F_2 \cap \Sigma_1 = \emptyset$, and $F_1 \cap \Sigma_2 = F_2 \cap \Sigma_1$ in the definition. In addition, the consideration of request-arrivals (R) and existing pending status (P) is required to satisfy the aforementioned invariants. For example, consider the following two transitions:

1. $\langle \ell_1, a \rangle \xrightarrow{b|ab,D_1} \langle \ell_1', \emptyset \rangle$ with $\Sigma_1 = \{a, b\}$ 2. $\langle \ell_2, b \rangle \xrightarrow{c|bc,D_2} \langle \ell_2', \emptyset \rangle$ with $\Sigma_2 = \{b, c\}$

Taking into account only the synchrony, e.g, $\{a, b\} \cap \Sigma_2 = \{b, c\} \cap \Sigma_1$, the product result of transitions 1 and 2 is

$$\langle (\ell_1, \ell_2), ab \rangle \xrightarrow{bc|abc,D_1 \cup D_2} \langle (\ell_1', \ell_2'), \emptyset \rangle$$

However, this violates invariant 5 of $P \cap R = \emptyset$, i.e., $\{a, b\} \cap \{b, c\} = \{b\}$. Therefore, in the product operation, $P_1 \cap R_2 = \emptyset$, $P_2 \cap R_1 = \emptyset$, or $P_1 \cap R_2 = \emptyset = P_2 \cap R_1$ must be considered.

The product result of the set D is the union of 3-tuple sets from each automaton. In order to keep QIA generally useful and compositional, and their product commutative, we avoid fixing the precise formal meaning of distributions of synchronized transitions composed in a product; instead, we represent the "processing delay rate" of their composite transition in the product automaton as the union of the processing delay rates of the synchronizing transitions of the two automata. How exactly these rates combine to yield the composite rate of the transition depends on the different properties of the distributions and their time ranges. For example, in the continuous-time case, no two events can occur at the same time; and the exponential distributions are not closed under taking maximum. In Section 3.3, we show how to translate a QIA to a CTMC using the union of the rates of the exponential distributions in the continuous-time case.

As an example, Figure 3.2 shows the product of a LossySync channel ab and a Sync channel bc. For simplicity, we represent the set of 3-tuples in the labels of transitions only by their names and give them in the table below.

Note that the resulting automaton includes unintended transitions such as $\langle \ell, c \rangle \xrightarrow{a|a,\alpha} \langle \ell, c \rangle$ and $\langle \ell, bc \rangle \xrightarrow{a|a,\alpha} \langle \ell, bc \rangle$ which imply losing data items at node a which violate context-dependency of the LossyFIFO1 connector. These unintended transitions are generated since the product operation does not consider mixed nodes as internal nodes that cannot interact with the outside. For this reason, we define a synchronization operation that makes mixed nodes internal and filters firing transitions taking into account the context-dependency of a Reo connector.

Definition 3.2.3 (Synchronization). *Given a QIA $\mathcal{A} = (Q, I, \Sigma, \rightarrow, \mathbf{r})$, synchronization of a mixed node $h \in \Sigma$, denoted by $synch[h](\mathcal{A})$, is equal to $(Q, I, \Sigma, \rightarrow', \mathbf{r}')$*

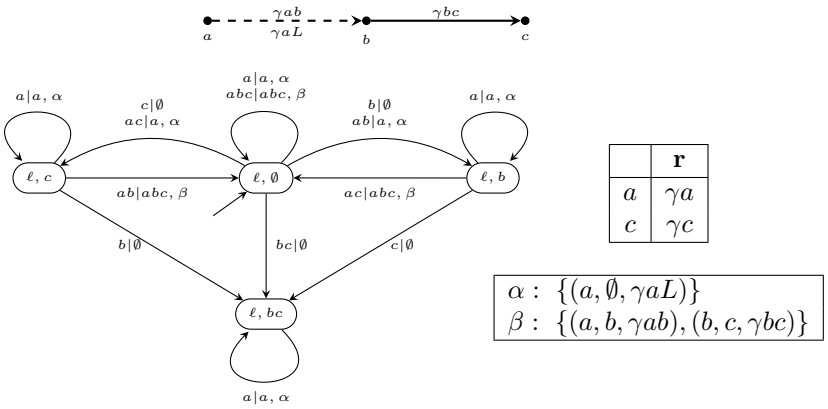


Figure 3.2: Product of a LossySync channel ab and a Sync channel bc

where

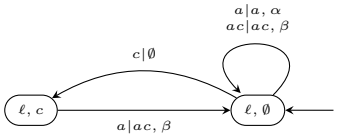
$$
\begin{array}{lcll}
\multicolumn{4}{l}{\mathbf{r}' \textit{ is } \mathbf{r} \textit{ restricted to the domain } \Sigma \setminus \{h\},\ r'(h) \textit{ is } \infty, \textit{ and}} \\
\rightarrow' & = & \{\langle \ell, P\rangle \xrightarrow{R,F,D} \langle \ell', P'\rangle \mid \langle \ell, P\rangle \xrightarrow{R \uplus \{h\}, F \uplus \{h\}, D} \langle \ell', P'\rangle\} & 1) \\
 & \cup & \{\langle \ell, P\rangle \xrightarrow{R,F,D} \langle \ell', P'\rangle \mid h \notin R \ \wedge\ h \notin F \ \wedge & \\
 & & \quad \nexists\ \langle \ell, P\rangle \xrightarrow{R',F',D'} \langle \ell'', P''\rangle \textit{ s.t. } R' = R \cup \{h\}\} & 2)
\end{array}
$$

where $\uplus : \Sigma \times \Sigma \to \Sigma$ *is a union restricted to disjoint sets.* ■

As an internal node, a mixed node does not interact with the environment and is always ready to dispense data-items, i.e., a mixed nodes deliver data items that it receives immediately. Thus, the synchronization operation restricts function $\mathbf{r}$ to only boundary nodes.

The product operation of QIA does not take into account context-dependency of connecting channels. The synchronization operation allows in an automaton (possibly resulting from the product of two automata) only firing transitions that respect the interaction with new environment. For example, consider the connector in Figure 3.2. In a LossySync channel ab, losing data at node a occurs only when node b is not pending. After the product with a Sync channel bc, node b is always pending, and losing data occurs only when node c is not pending. However, the state $\langle \ell, c\rangle$ in the product result has two firings for losing data at node a and dispensing data from node a to node c via node b. The synchronization checks such situation and deletes unintended firings, i.e., it allows firings that 1) consider pending and firing at the mixed nodes as an immediate atomic activity or 2) are independent of mixed nodes. The QIA synchronization operation is analogous to the hiding operation [31, Chapter 4] for IA. The result of the synchronization operation on the product result

in Figure 3.2 is shown in Figure 3.3, which is the same as the original QIA for a LossySync channel.



$\alpha: \{(a, \emptyset, \gamma aL)\}$
$\beta: \{(a, b, \gamma ab), (b, c, \gamma bc)\}$

Figure 3.3: Synchronization result on QIA in Figure 3.2

The semantics of plugging two Reo connectors together on a common node h is represented in QIA by first considering the product of their QIA and then applying the synchronization operation, i.e. $synch[h](\mathcal{A}_1 \bowtie \mathcal{A}_2)$. Thus, the product and the synchronization operations can be used to obtain, in a compositional way, the QIA of a connector built out of the primitive channels that comprise the connector. Given two QIA $\mathcal{A}_1$ and $\mathcal{A}_2$ with their node sets Σ_1 and Σ_2, respectively, sharing the common nodes $\Sigma_1 \cap \Sigma_2 = \{h_1, h_2, \ldots, h_k\}$, $synch[h_1](synch[h_2] \cdots (synch[h_k](\mathcal{A}_1 \bowtie \mathcal{A}_2)))$ represents the automaton corresponding to a connector. Note that the "plugging" order does not matter as synchronization interacts well with product.

Example 3.2.4. As a more complex example of the QIA composition, we apply the product and synchronization operations to the LossyFIFO1 example in Figure 2.5. The product result $\mathcal{A} = (Q, I, \Sigma, \rightarrow, \mathbf{r})$ of a LossySync channel ab and a FIFO1 channel bc is too big to draw and not readable. Instead of showing the whole figure of $\mathcal{A}$, we show $\rightarrow_{prod}$, the transitions that will be considered by the synchronization operation:

$$
\begin{aligned}
\rightarrow_{prod} = \{ \quad & \langle \ell e, \emptyset \rangle \xrightarrow{c|\emptyset} \langle \ell e, c \rangle, && \checkmark \\
& \langle \ell e, \emptyset \rangle \xrightarrow{\mathbf{ac}|\mathbf{a}} \langle \ell e, c \rangle, && \times \\
& \langle \ell e, \emptyset \rangle \xrightarrow{\mathbf{ac}\mathrm{b}|\mathbf{ab}} \langle \ell f, c \rangle, && \checkmark \\
& \langle \ell e, \emptyset \rangle \xrightarrow{\mathbf{a}|\mathbf{a}} \langle \ell e, \emptyset \rangle, && \times \\
& \langle \ell e, \emptyset \rangle \xrightarrow{\mathbf{ab}|\mathbf{ab}} \langle \ell f, \emptyset \rangle, && \checkmark \\
& \langle \ell f, \emptyset \rangle \xrightarrow{a|a} \langle \ell f, \emptyset \rangle, && \checkmark \\
& \langle \ell f, \emptyset \rangle \xrightarrow{c|c} \langle \ell e, \emptyset \rangle, && \checkmark \\
& \langle \ell f, \emptyset \rangle \xrightarrow{ac|ac} \langle \ell e, \emptyset \rangle, && \checkmark \\
& \langle \ell e, c \rangle \xrightarrow{\mathbf{a}|\mathbf{a}} \langle \ell e, c \rangle, && \times \\
& \langle \ell e, c \rangle \xrightarrow{\mathbf{ab}|\mathbf{ab}} \langle \ell f, c \rangle, && \checkmark \\
& \langle \ell f, c \rangle \xrightarrow{a|a} \langle \ell f, c \rangle, && \checkmark \\
& \langle \ell f, c \rangle \xrightarrow{\emptyset|c} \langle \ell e, \emptyset \rangle, && \checkmark \\
& \langle \ell f, c \rangle \xrightarrow{a|ac} \langle \ell e, \emptyset \rangle && \checkmark \quad \}
\end{aligned}
$$

Note that the system configuration ℓe and ℓf are the abbreviation of (ℓ, e) and (ℓ, f) where ℓ represents the system configuration of a LossySync channel ab, and e and f represent the configurations of, respectively, an empty and a full buffer of the FIFO1 channel bc. Formally, these system configurations must be written as $\langle(\ell, e), P\rangle$ where $P \subseteq \{a, b, c\}$, but for simplicity, we use the aforementioned abbreviations. The transitions with the bold labels represent the filtered transitions by the synchronization. The reason of this filtering is that the nodes in bold in these transitions are dependent on mixed nodes, thus, they must fire together with the mixed nodes. This dependency is shown by the presence of their counterparts that fire with the same nodes(represented in **bold**) as well as the mixed nodes (represented in roman, next to the nodes in bold). Each counterpart follows its relevant filtered transition above and belongs to the transition set 1) in Definition 3.2.3. The transitions with the black labels represent the independent firings of the mixed nodes and belong to the transition set 2) in Definition 3.2.3.

In this example, the three transitions $\langle \ell e, \emptyset\rangle \xrightarrow{a|a} \langle \ell e, \emptyset\rangle$, $\langle \ell e, \emptyset\rangle \xrightarrow{ac|a} \langle \ell e, c\rangle$, and $\langle \ell e, c\rangle \xrightarrow{a|a} \langle \ell e, c\rangle$, which have only the labels in bold, are deleted by the synchronization because they have counterparts that fire the pending mixed node b since node b is always ready to dispense data items as an internal node. The counterparts of these transitions are, respectively, $\langle \ell e, \emptyset\rangle \xrightarrow{ab|ab} \langle \ell f, \emptyset\rangle$, $\langle \ell e, \emptyset\rangle \xrightarrow{abc|ab} \langle \ell f, c\rangle$, and $\langle \ell e, c\rangle \xrightarrow{ab|ab} \langle \ell f, c\rangle$, firing mixed node b is represented in red. The synchronization result of the product result $\mathcal{A}$ is shown in Figure 3.4.

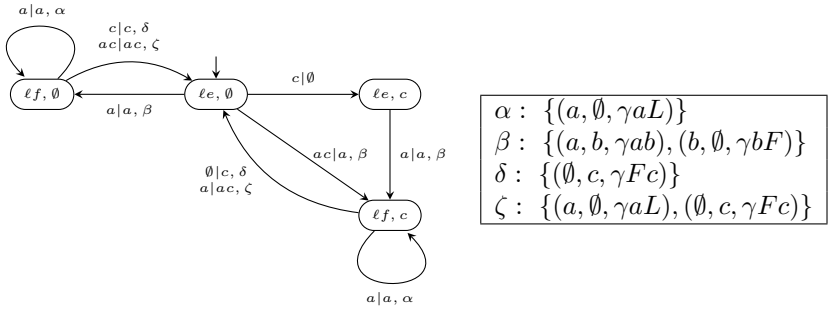


Figure 3.4: Corresponding QIA for LossyFIFO1 in Figure 2.5

◊

3.3 Translation into a stochastic model

In this section, we show how to translate QIA into a homogeneous CTMC model for stochastic analysis. In general, CTMCs are not compositional and large even for a

small system. That is, it is difficult to model CTMCs for complex systems directly. In our approach, modeling compositional behavior is carried out by QIA, and then corresponding CTMCs are derived from QIA which is considered as an intermediate model for this translation. Even though the resulting CTMCs are still big to handle, we can easily obtain CTMCs for complex systems via QIA. A homogeneous CTMC is a stochastic process with (1) discrete state space, (2) Markov property, (3) memoryless property, and (4) homogeneity in the continuous-time domain [43]. These properties yield efficient methodologies for numerical analysis. Note that here and in the remainder of this chapter, CTMCs are homogeneous even though it is not explicitly mentioned.

In the continuous-time domain, the exponential distribution is the only one that satisfies the memoryless property. Therefore, for the translation, we assume that the rates of request-arrivals and data-flows are exponentially distributed. However, note that such restriction did not appear in the QIA model. Other types of distributions can appear in the label of QIA, but then the target model has to be other than CTMCs.

A CTMC model derived from a QIA is a pair (S, δ) where $S = S_A \cup S_M$ is the set of states. S_A represents the configurations of the system derived from its QIA including the pending status of I/O requests; S_M is the set of states that result from the micro-step division of synchronized actions (see below). $\delta = \delta_{Arr} \cup \delta_{Proc} \subseteq S \times \mathbb{R}^+ \times S$, explained below, is the set of transitions, each labeled with a stochastic value specifying the arrival or the processing delay rate of the transition. δ_{Arr} and δ_{Proc} are defined in Section 3.3.4 and Section 3.3.3, respectively.

A state in QIA models a configuration of the connector, including the presence of the I/O requests pending on its boundary nodes, if any. Request-arrivals change system configuration only by changing the pending status of their respective boundary nodes. Data-flows corresponding to a transition in QIA change the system configurations, and release the pending I/O requests on their involved boundary nodes. In addition, data-flows depicted in a single transition illustrate multiple synchronized firings. In the following, we show how to deal with such request-arrivals and data-flows in an appropriate way for the translation from QIA into CTMCs.

In a CTMC model, the probability that two events (e.g., the arrival of an I/O request, the transfer of a data item, a processing step, etc.) happen at the same time is *zero*: only a single event occurs at a time. In compliance with this requirement, for a QIA $\mathcal{A} = (L \times 2^{\Sigma}, I, \Sigma, \rightarrow, \mathbf{r})$ and a set of boundary nodes Σ', we define its set of request-arrival transitions, δ_{Arr}, in several steps. The set S_A and the preliminary set[2] of request-arrival transitions of the CTMC derived from $\mathcal{A}$ are defined as:

[2]In the process of generating CTMCs, some macro-step events (e.g., synchronized data-flows) are divided into several micro-step events. After that, independent events (e.g., request-arrivals) are considered as preemptive events between any two micro-step event. Before this division, we need to specify the transitions for respective synchronized data-flows. For this purpose, S_A is obtained to describe source and target states of these transitions. The preliminary set of request-arrivals includes the transitions that connect the states in S_A, each of which corresponds to all possible request-arrivals at every connector configurations.

$$\begin{aligned} S_A &= \{\langle q, P\rangle \mid q \in L,\ P \subseteq \Sigma'\} \\ \delta'_{Arr} &= \{\langle q, P\rangle \xrightarrow{v} \langle q, P \cup \{d\}\rangle \mid \langle q, P\rangle,\ \langle q, P \cup \{d\}\rangle \in S_A,\ v = \mathbf{r}(d)\} \end{aligned}$$

The set δ'_{Arr} is used in Section 3.3.4 to define the δ_{Arr} component of δ.

As an example of obtaining S_A and δ'_{Arr}, recall the QIA for the LossyFIFO1 circuit in Figure 3.4. It has two system configuration of states ℓe and ℓf, and its boundary nodes set Σ' is $\{a, c\}$. Therefore:

$$\begin{aligned} S_A &= \{\ \langle \ell e, \emptyset\rangle,\ \langle \ell e, a\rangle,\ \langle \ell e, c\rangle,\ \langle \ell e, ac\rangle,\ \langle \ell f, \emptyset\rangle,\ \langle \ell f, a\rangle,\ \langle \ell f, c\rangle, \langle \ell f, ac\rangle\ \} \\ \delta'_{Arr} &= \{\ \langle \ell e, \emptyset\rangle \xrightarrow{\gamma a} \langle \ell e, a\rangle,\ \langle \ell e, \emptyset\rangle \xrightarrow{\gamma c} \langle \ell e, c\rangle,\ \langle \ell e, a\rangle \xrightarrow{\gamma c} \langle \ell e, ac\rangle, \\ &\quad \langle \ell e, c\rangle \xrightarrow{\gamma a} \langle \ell e, ac\rangle,\ \langle \ell f, \emptyset\rangle \xrightarrow{\gamma a} \langle \ell f, a\rangle,\ \langle \ell f, \emptyset\rangle \xrightarrow{\gamma c} \langle \ell f, c\rangle, \\ &\quad \langle \ell f, a\rangle \xrightarrow{\gamma c} \langle \ell f, ac\rangle,\ \langle \ell e, c\rangle \xrightarrow{\gamma a} \langle \ell f, ac\rangle\ \} \end{aligned}$$

The diagrams of S_A and δ'_{Arr} are presented in Figure 3.5.

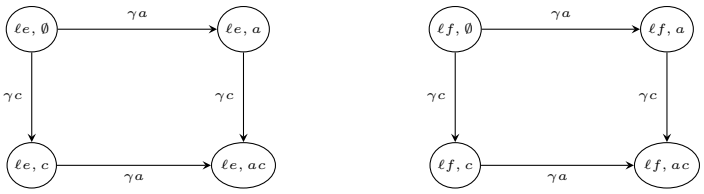


Figure 3.5: State diagram for request-arrivals

3.3.1 Micro-step transitions

The CTMC transitions associated with data-flows are more complicated because groups of synchronized data-flows are modeled as a single transition[3] in QIA, abstracting away their precise occurrence order. Therefore, we need to divide such synchronized data-flows into so-called micro-step transitions[4], respecting the connection information, i.e., the topology of a Reo connector, through which the data-flow occurs.

The connection information can be recovered from the 3-tuples in the label on each firing transition in a QIA, since the first and the second elements of a 3-tuple describe, respectively, the input and the output nodes involved in the data-flow of its transition, and the data-flow in the transition occurs from its input to its output nodes.

For example, the transition from state $\langle \ell e, \emptyset\rangle$ to state $\langle \ell f, \emptyset\rangle$ in the QIA of the LossyFIFO1 example in Figure 3.4 has $\{(a, b, \gamma ab), (b, \emptyset, \gamma bF)\}$ as its set of 3-tuples. The connection information inferred from this set states that the data-flows occur

[3]Note that here and in the remainder of this section, we skip to explicitly mention that a QIA transition $s \xrightarrow{R|F,D} s'$ satisfies $F \neq \emptyset\ \wedge\ D \neq \emptyset$ for its synchronized data-flows.

[4]This division delineates synchronized data-flows, not each data-flow itself.

from a to the buffer through b. The transition is, thus, divided into two consecutive micro-step transitions $(a, b, \gamma ab)$ and $(b, \emptyset, \gamma bF)$.

Such data-flow information on each firing transition in a QIA is formalized by a *delay-sequence* defined by the following grammar:

$$\Lambda \ni \lambda ::= \epsilon \ \mid \ \theta \ \mid \ \lambda|\lambda \ \mid \ \lambda;\lambda$$

where ϵ is the empty sequence, and θ is a 3-tuple (I, O, r) for a primitive Reo channel. $\lambda|\lambda$ denotes parallel composition, and $\lambda;\lambda$ denotes sequential composition. The empty sequence ϵ is an identity element for $|$ and ;, $|$ is commutative, associative, and idempotent, ; is associative and distributes over $|$. Most of properties of these compositional operators are intuitive, except for the distributivity of ;. The delay-sequence λ extracted by the Algorithm 3.3.1 is in the format $\lambda = \lambda_1|\lambda_2|\ldots|\lambda_n$. Consider $\lambda = \lambda_1|\lambda_2 = (\theta_1;\theta_2)|(\theta_3;\theta_2)$[5]. Distributivity, that is, the property $(\theta_1;\theta_2)|(\theta_3;\theta_2) = (\theta_1|\theta_3);\theta_2$ is justified by the fact that θ_2 is the delay of the same action and the other actions θ_1 and θ_3 in the composed delays $(\theta_1;\theta_2)$ and $(\theta_3;\theta_2)$ need to finish before the action corresponding to θ_2 occurs. We use this distributivity law to generate compacter delay-sequences from the delay-sequences extracted in Section 3.3.2. For example, recall the delay-sequence $\lambda = (\theta_1|\theta_2)|(\theta_3;\theta_2)$. Then, λ becomes $(\theta_1|\theta_3);\theta_2$ and it still preserves the sequential precedence of θ_1 and θ_3 over θ_2 and shows the undetermined order between θ_1 and θ_3.

3.3.2 Extracting a delay-sequence

The delay-sequence corresponding to a set of 3-tuples associated with a transition in a QIA is obtained by Algorithm 3.3.1. Note that if the parameter of the function **Ext** is a singleton, then $\mathbf{Ext}(\{\theta\}) = \theta$ since $i(\theta) \cap o(\theta) = \emptyset$.

Intuitively, the **Ext** function delineates the set of activities that – at the level of a QIA – must happen synchronously/atomically, into its corresponding delay-sequences. If a certain data-flow associated with a 3-tuple θ_1 explicitly precedes another one θ_2, then θ_1 is sequenced before θ_2, i.e., encoded as $\theta_1;\theta_2$. Otherwise, they can occur in any order, encoded as $\theta_1|\theta_2$.

Applying Algorithm 3.3.1 to the LossyFIFO1 example in Figure 3.4 yields the following result shown in Figure 3.6, where the delineated results appear in the table.

The parameter D of Algorithm 3.3.1 is a finite set of 3-tuples, and *Init*, *Post* and *toGo*, subsets of D, are also finite. Moreover, *Post* becomes eventually $\emptyset$ since *toGo* decreases during the procedure. Thus, we can conclude that Algorithm 3.3.1 always terminates.

A resulting delay-sequence S extracted by Algorithm 3.3.1 is generated by the parallel composition of λ_θ. The order of selecting θ from the set *Init* is not deterministic, thus, the resulting delay-sequence for the same input can be syntactically different,

[5]In general, the operators inside '()' have the highest order. Here and in the remainder of this thesis, we also follow this standard order without explicit mention.

Algorithm 3.3.1: Extraction of a delay-sequence out of a set Θ of 3-tuples

Ext(D) where D in $p \xrightarrow{R|F,D} q$
  $S = \epsilon$
  $toGo = D$
  $Init := \{\theta \in D \mid i(\theta) \cap o(\theta') = \emptyset \text{ for all } \theta' \in D\}$
  for $\theta \in Init$ **do**
    $\lambda_\theta := \theta$
    $Pre := \{\theta\}$
    $toGo := toGo \setminus Pre$
    $Post = \{\theta \in toGo \mid \exists \theta' \in Pre \ s.t. \ o(\theta') \cap i(\theta) \neq \emptyset\}$
    while $Post \neq \emptyset$ **do**
      $\lambda' := (\theta_1 | \cdots | \theta_k)$ where $Post = \{\theta_1, \cdots, \theta_k\}$
      $\lambda_\theta := \lambda_\theta; \lambda'$
      $Pre := Post$
      $toGo := toGo \setminus Pre$
      $Post := \{\theta \in toGo \mid \exists \theta' \in Pre \ s.t. \ o(\theta') \cap i(\theta) \neq \emptyset\}$
    end while
    $S := S|\lambda_\theta$
  end for
  return S

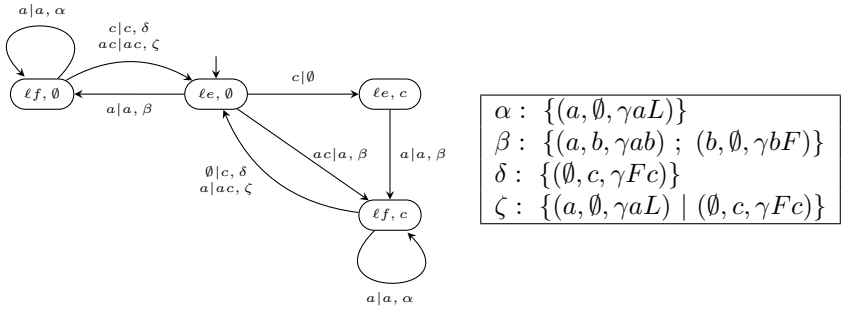


Figure 3.6: Applying Algorithm 3.3.1 to QIA in Figure 3.4

for example, $\lambda_\theta|\lambda_{\theta'}$ and $\lambda_{\theta'}|\lambda_\theta$ with $Init = \{\theta, \theta'\}$. However, the parallel composition operator | is commutative, thus, the composition order of | does not matter.

3.3.3 Dividing macro-step transitions with a delay-sequence

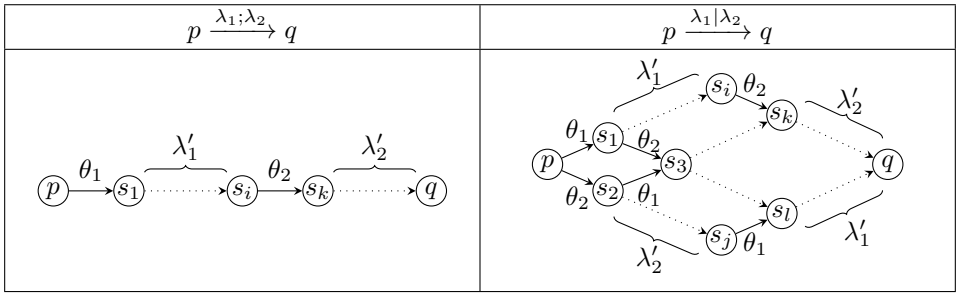
We now show how to derive the transitions in the CTMC model from the QIA transitions. In QIA, data-flows and request-arrivals can be put on a single transition,

whereas, in CTMCs, these two must be considered separately. For this purpose, we explore cases in which sole data-flows are possible. Recall the invariants of QIA in Section 3.2.1. A firing can occur when all its relevant nodes are pending. For a QIA transition $\langle q, P\rangle \xrightarrow{R|F,D} \langle q', P'\rangle$, the firing can occur if $F \subseteq P \cup R$. In compliance with this consideration, we derive the CTMC transitions in two steps:

1. For each QIA transition $\langle q, P\rangle \xrightarrow{R|F,D} \langle q', P'\rangle \in \rightarrow$ such that $F \neq \emptyset \ \wedge \ D \neq \emptyset$, we derive transitions $\langle q, P''\rangle \xrightarrow{\lambda} \langle q', P'' \setminus F\rangle$ where P'' is a node set that includes all the firing nodes of each QIA transition; λ is the delay-sequence associated with the set of 3-tuples D in the label on the transition. This set of derived transitions is defined below as δ_{Macro}.

2. We divide a transition in δ_{Macro} labeled by λ into a combination of micro-step transitions, each of which corresponds to a single event.

The following figure briefly illustrates the procedure mentioned above, for the two transitions $p \xrightarrow{\lambda_1;\lambda_2} q$ and $p \xrightarrow{\lambda_1|\lambda_2} q$ where $\lambda_1 = \theta_1; \lambda_1'$ and $\lambda_2 = \theta_2; \lambda_2'$:



A sequential delay-sequence $\lambda_1; \lambda_2$ allows for the events corresponding to λ_1 to occur before the ones corresponding to λ_2. For a parallel delay-sequence $\lambda_1|\lambda_2$, events corresponding to λ_1 and λ_2 occur interleaving each other, while they preserve their respective order of occurrence in λ_1 and λ_2. All indexed states s_n are included in S_M which consists of the states derived from the division of the synchronized data-flows into micro-step transitions. The formal description of dealing with these two delay-sequences is presented in the definition of a *div* function below, in which handling the respective delay-sequences correspond to the second and the third conditions of the *div* function.

Given a QIA $(Q, I, \Sigma, \rightarrow, \mathbf{r})$ and its boundary nodes set Σ', a macro-step transition relation for the synchronized data-flows is defined as:

$$\delta_{Macro} = \{\langle p, P''\rangle \xrightarrow{\lambda} \langle q, P'' \setminus F\rangle \mid \langle p, P\rangle \xrightarrow{R|F,D} \langle q, P'\rangle \in \rightarrow,\ F \subseteq P'' \subseteq \Sigma',\ \lambda = \mathbf{Ext}(D)\}$$

As an example of obtaining a macro-step transition relation, consider the transition $\langle \ell e, \emptyset\rangle \xrightarrow{a|a,\beta} \langle \ell f, \emptyset\rangle$ with $\beta = (a, b, \gamma ab)\ ;\ (b, \emptyset, \gamma bF)$ in Figure 3.6. Given the firing

set $\{a\}$ and the boundary nodes set $\Sigma' = \{a, c\}$, P'' is $\{a\}$ or $\{a, c\}$, this generates the macro-step transitions $\langle \ell e, a\rangle \xrightarrow{\beta} \langle \ell f, \emptyset\rangle$ and $\langle \ell e, ac\rangle \xrightarrow{\beta} \langle \ell f, c\rangle$. Figure 3.7 shows a state diagram derived from the QIA of the LossyFIFO1 example in Figure 3.4 with the set of macro-step transitions δ_{Macro} and the preliminary set of request-arrival transitions δ'_{Arr}, which are represented as dashed transitions.

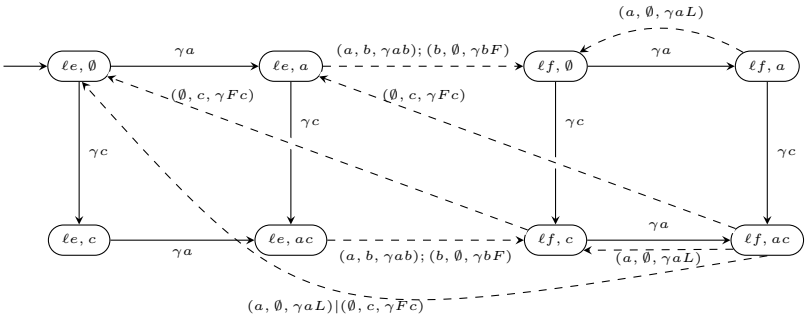


Figure 3.7: State diagram derived from the QIA in Figure 3.4 with δ_{Macro} and δ'_{Arr}

We now explicate a macro-step transition with a number of micro-step transitions, each of which corresponds to a single data-flow. This refinement yields auxiliary states between the source and the target states of the macro-step transition. Let $\langle p, P\rangle$ be a source state for a data-flow corresponding to a 3-tuple θ. The generated auxiliary states are defined as $\langle p_\theta, P \setminus nodes(\theta)\rangle$ where p_θ is just a label denoting that the data-flow corresponding to θ has occurred, and the function $nodes : \Lambda \to 2^\Sigma$ is defined for the delay-sequence level as:

$$nodes(\lambda) = \begin{cases} i(\theta) \cup o(\theta) & \text{if } \lambda = \theta \\ nodes(\lambda_1) \cup nodes(\lambda_2) & \text{if } \lambda = \lambda_1; \lambda_2 \ \vee \lambda = \lambda_1 | \lambda_2 \end{cases}$$

The set of such auxiliary states is obtained as $S_M = states(\langle p, P\rangle \xrightarrow{\lambda} \langle q, P'\rangle)$ where

$$states(\langle p, P\rangle \xrightarrow{\lambda} \langle q, P'\rangle) = \begin{cases} \{\langle p, P\rangle, \langle q, P'\rangle\} & \text{if } \lambda = \theta \\ \bigcup states(m) \ \forall m \in div(\langle p, P\rangle \xrightarrow{\lambda} \langle q, P'\rangle) & \text{otherwise} \end{cases}$$

The function $div : \delta_{Macro} \to 2^{\delta_{Macro}}$ is defined as:

$$
div(\langle p, P\rangle \xrightarrow{\lambda} \langle q, P'\rangle) = \\
\begin{cases}
\{\langle p, P\rangle \xrightarrow{\theta} \langle q, P'\rangle\} & \text{if } \lambda = \theta \ \wedge \ \nexists \langle p, P\rangle \xrightarrow{\theta} \langle p', P'\rangle \in \delta_{Macro} \\
div(\langle p, P\rangle \xrightarrow{\lambda_1} \langle p_{\lambda_1}, P''\rangle) \ \cup \ div(\langle p_{\lambda_1}, P''\rangle \xrightarrow{\lambda_2} \langle q, P'\rangle) & \\
\qquad\qquad \text{if } \lambda = \lambda_1;\lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_1) & \\
\{m_1 \bowtie m_2 \mid m_i \in div(\langle p, P\rangle \xrightarrow{\lambda_i} \langle p_{\lambda_i}, P''\rangle),\ i \in \{1,2\}\} & \\
\qquad\qquad \text{if } \lambda = \lambda_1|\lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_i) & \\
\emptyset & \text{otherwise}
\end{cases}
$$

where the function $\bowtie$: $\delta_{Macro} \times \delta_{Macro} \to 2^{\delta_{Macro}}$ computes all interleaving compositions of the two transitions as follows. For a transition $(p, R) \xrightarrow{\lambda_1|\lambda_2} (q, R') \in \delta_{Macro}$, $(p, R) \xrightarrow{\lambda_1} (p_{\lambda_1}, R \setminus nodes(\lambda_1))$ and $(p, R) \xrightarrow{\lambda_2} (p_{\lambda_2}, R \setminus nodes(\lambda_2))$ correspond to, respectively, m_1 and m_2 of the third condition in the definition of the div function. While m_1 and m_2 are handled by the div function recursively, some auxiliary states, i.e., $states(m_1)$ and $states(m_2)$, are generated. In the interleaving composition, m_1 can occur at any states that are generated by $states(m_2)$, and vice-versa. This interleaving composition of m_1 and m_2 is represented as:

$$
\begin{aligned}
m_1 \bowtie m_2 = \ \{\ & div\big((p_1, R_1) \xrightarrow{\lambda_2} (p_{(1,\lambda_2)}, R \setminus nodes(\lambda_2))\big), \\
& div\big((p_2, R_2) \xrightarrow{\lambda_1} (p_{(2,\lambda_1)}, R \setminus nodes(\lambda_1))\big) \ \mid \\
& \qquad (p_1, R_1) \in states(m_1) \text{ and } (p_2, R_2) \in states(m_2)\ \}
\end{aligned}
$$

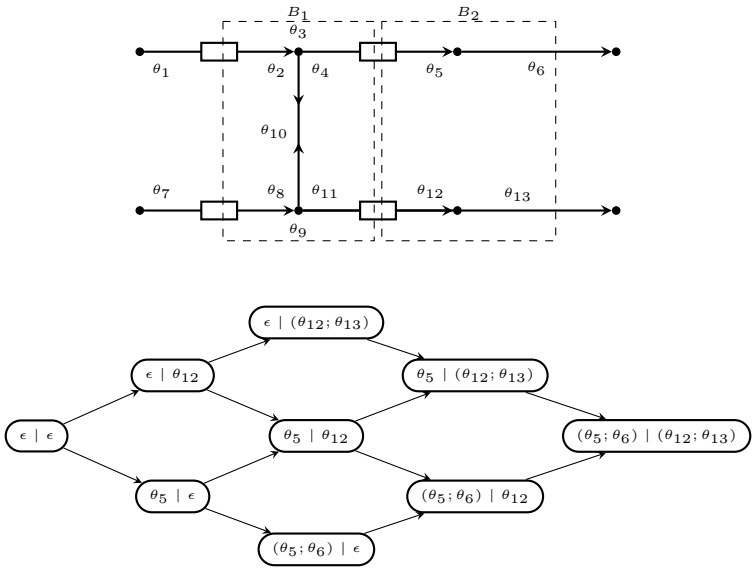
The following example shows the application of the function div to a non-trivial delay-sequence, which contains a combination of sequential and parallel compositions.

Example 3.3.1. Consider the Stochastic Reo connector shown below. Every indexed θ is a rate for its respective processing activity, e.g., θ_2 is the rate at which the top-left FIFO1 dispenses data through its sink end; θ_3 is the rate at which the node replicates its incoming data, etc. Data-flows contained in boxed regions marked as B_1 and B_2 appear in δ_{Macro}, derived from the QIA of this circuit, as two transitions with the delay-sequences of λ_1 and λ_2 where:

- from B_1: $\lambda_1 = ((\theta_2;\theta_3)|(\theta_8;\theta_9))\ ;\ (\theta_4|\theta_{10}|\theta_{11})$
- from B_2: $\lambda_2 = (\theta_5;\theta_6)\ |\ (\theta_{12};\theta_{13})$

To derive a CTMC, λ_1 and λ_2 must be divided into micro-step transitions. We exemplify a few of these divisions. For λ_1, the division of $(\theta_4|\theta_{10}|\theta_{11})$ is trivial since it contains only simple parallel composition. This division result is then appended to the division result of $(\theta_2;\theta_3)|(\theta_8;\theta_9)$, which has the same structure as that of λ_2. Thus, we show below the division result of λ_2 only.

In the following CTMC fragment, to depict which events have occurred up to a current state, the name of each state consists of the delays of all the events that have



occurred up to that state. The delay for a newly occurring event is appended at the end of its respective segment in the current state name.

This example shows that when a delay-sequence λ is generated by parallel composition, the events in one of the sub-delay-sequences of λ occur independently of the events in other sub-delay-sequences. Still events preserve their occurrence order within the sub-delay-sequence that they belong to. ◊

The division into micro-step transitions ensures that each such transition has a single 3-tuple in its label. As mentioned above, this 3-tuple includes the structural information and the processing delay rate of its relevant data-flow, but in CTMCs, only the processing delay rate is used. Thus, the extraction of processing delay rates from the micro-step transitions are defined as:

$$\delta_{Proc} = \{\langle p, P\rangle \xrightarrow{v(\theta)} \langle p', P'\rangle \mid \langle p, P\rangle \xrightarrow{\theta} \langle p', P'\rangle \in div(t) \text{ for all } t \in \delta_{Macro}\}$$

Figure 3.8 shows applying the division method and extracting rates from the LossyFIFO1 example in Figure 3.7. In Figure 3.8, the elements in S_M appear in gray, and the micro-step transitions by the division method are represented as dashed transitions.

3.3.4 Preemptive request-arrivals

Synchronized data-flows in QIA are considered atomic, thus other events cannot interfere with them. However, splitting these data-flows allows non-interfering events to

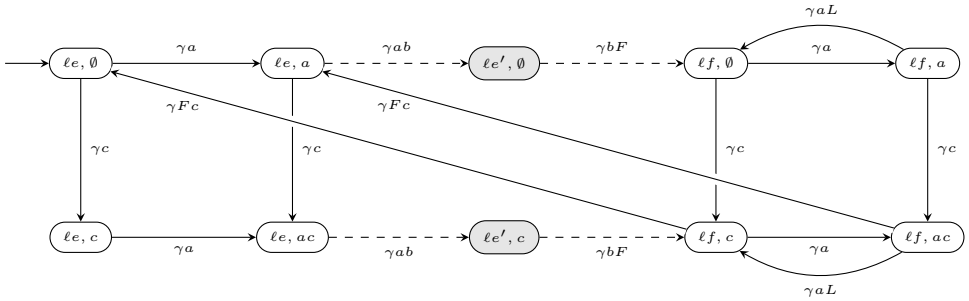


Figure 3.8: Division result of Figure 3.7

interleave with their micro-steps, disregarding the strict sense of their atomicity. For example, a certain boundary node unrelated to a group of synchronized data-flows can accept a data item between any two micro-steps. Since we want to allow such interleaving, we must explicitly add such request-arrivals. With a set of micro-step states S_M, its full set of request-arrival transitions, including its preliminary request-arrival set δ'_{Arr}, is defined as:

$$\delta_{Arr} = \delta'_{Arr} \cup \{\langle p, P\rangle \xrightarrow{\mathbf{r}(d)} \langle p, P \cup \{d\}\rangle \mid \langle p, P\rangle,\ \langle p, P \cup \{d\}\rangle \in S_M, d \in \Sigma,\ d \notin P\}$$

Figure 3.9 shows the consideration of all possible preemptive request-arrivals for the division result in Figure 3.8, and the preemptive request-arrival is represented as a dashed transition. Thus, Figure 3.9 is the CTMC model $(S_A \cup S_M, \delta_{Arr} \cup \delta_{Proc})$ derived from the LossyFIFO1 example in Figure 2.5 via its QIA in Figure 3.4.

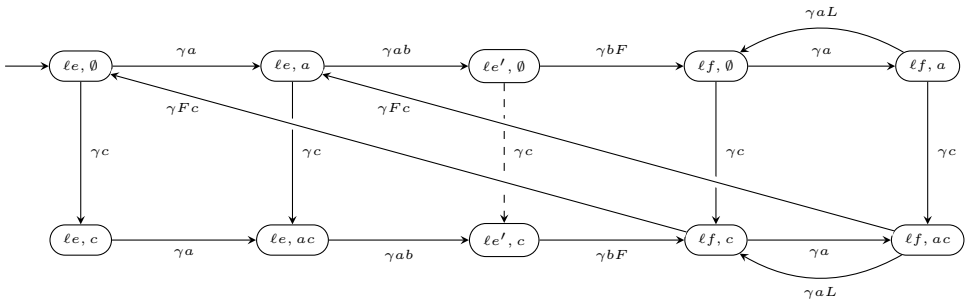


Figure 3.9: Derived CTMC of LossyFIFO1

3.4 Discussion

In this chapter, we introduced QIA as a semantic model for Stochastic Reo. This model specifies the behavior of a connector that coordinates services distributed over a network, along with its end-to-end QoS, specified as stochastic rates. QIA are an extension of IA, thus, QIA also consider both I/O request arrivals at channel ends and data-flows through channels separately. In contrast to IA, request arrivals and data-flows in QIA are considered stochastic activities. Considering the interaction with the environment of a connector, i.e., I/O request arrivals at channel ends, as a stochastic activity, QIA can specify and reason about end-to-end QoS aspects of system behavior. As IA capture the context-dependency of connectors, QIA, an extension of IA, also capture the context-dependency of connectors. As a complex connector is built out of the primitive Stochastic Reo channels, the QIA model corresponding to a complex connector is also obtained by composing the QIA models corresponding to the primitive Stochastic Reo channels that comprise the connector.

QIA are considered an intermediate model for translation into stochastic models, in particular CTMCs, for stochastic analysis. CTMCs are frequently used stochastic processes with some restrictions, such as discrete state space and Markov property. These restrictions (features) provide efficient algorithms for their numerical analysis [85]. For this purpose, we have shown the translation from Stochastic Reo into CTMCs in this chapter. Based on this method, a tool has been implemented in the Extensible Coordination Tools (ECT) [35], whose implementation details will be shown in Chapter 5. The CTMCs derived from Stochastic Reo via the QIA semantic model can be used for analysis of the stochastic behavior of Reo connectors.

In general, QIA are large models in terms of the number of states and transitions, because their configurations include not only data-flows, but also the interaction with the environment, in contrast to CA which consider the configurations of data-flows only. Thus, QIA quickly become too large to handle. Moreover, as a semantic model for Stochastic Reo, QIA must support the compositional semantics of a Stochastic Reo connector. However, the proof of the compositionality of QIA is far from trivial. Consequently, we designed a more compact and tractable semantic model, called Stochastic Reo Automata, which we present in Chapter 4.

Chapter 4

Stochastic Reo Automata

4.1 Introduction

In the previous chapter, we introduced Quantitative Intentional Automata (QIA), a compositional semantic model for Stochastic Reo. QIA specify the behavior of connectors and enable reasoning about their end-to-end QoS. However, QIA explicitly describe all I/O interaction with the environment which is abstracted away in other (non-stochastic) semantic models such as Constraint Automata (CA) and Reo Automata. An explicit description of all interaction with the environment produces many states and transitions. Having a large state diagram, QIA are not easy to handle.

In this chapter, we introduce Stochastic Reo Automata [68] as an alternative semantic model for Stochastic Reo. Not only a Stochastic Reo Automaton is compact and tractable, but it also retains the features of QIA for representing context-dependency and reasoning about end-to-end QoS. Moreover, in order to reason about general end-to-end QoS, a Stochastic Reo Automaton is extended with reward information to deal with Stochastic Reo with rewards.

This chapter consists of four parts. In the first part, Stochastic Reo Automata are introduced as an alternative semantic model for Stochastic Reo. In fact, this model is a stochastic extension of Reo Automata. We introduce that the mapping between primitive Reo channels and their corresponding Stochastic Reo Automata, as well as the composition operation for Stochastic Reo Automata.

The second part shows the extended version of Stochastic Reo Automata for general end-to-end QoS properties. In this extension, the general QoS aspects are considered as reward information, which is associated with stochastic activities such as I/O request arrivals at channel ends and data-flows through channels. We also show the mapping of Stochastic Reo to Stochastic Reo Automata with the concern for the reward information.

The third part shows the translation from Stochastic Reo Automata into homogeneous CTMCs. In Chapter 3, we have shown the translation from QIA into CTMCs. This translation is partially similar to the translation from Stochastic Reo Automata into CTMCs. To avoid duplication, we skip some procedures that we reuse from the

earlier translation method. In addition, we present the translation from the Stochastic Reo Automata extended with reward information into CTMCs with state reward.

In the fourth part, we discuss to what extent Interactive Markov Chains (IMCs) can serve as another semantic model for Stochastic Reo. As shown in Section 2.5, the main strength of IMCs is their compositionality. In this section, we show that in our treatment the compositionality of IMCs is not adequate to specify the behavior of Stochastic Reo.

4.2 Stochastic Reo Automata

Stochastic Reo Automata constitute an alternative semantic model for Stochastic Reo to the model explained in Chapter 3. Compared to QIA, each Stochastic Reo Automaton has a disjoint set of node names. For example, two QIA $\mathcal{A}_1$ and $\mathcal{A}_2$ with node sets $\Sigma_{\mathcal{A}_1}$ and $\Sigma_{\mathcal{A}_2}$, respectively, are synchronized at nodes in $\Sigma_{\mathcal{A}_1} \cap \Sigma_{\mathcal{A}_2}$, whereas two Stochastic Reo Automata $\mathcal{B}_1$ and $\mathcal{B}_2$ are assumed that the two automata have disjoint node sets and can either take a step together or independently. Thus, naming the nodes of Stochastic Reo for a Stochastic Reo Automaton is slightly different from that for QIA. For instance, compared to Figure 2.5, Figure 4.1 shows the difference for the primitive channels of a LossySync and a FIFO1 and their composition result, i.e., the joined nodes in Figure 4.1 do not use a common name.

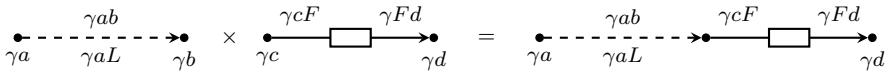


Figure 4.1: Stochastic LossyFIFO1 connector

As a more complex Stochastic Reo connector, Figure 4.2 shows a *discriminator* which takes the first arriving input value and produces it as its output. It also ensures that an input value arrives on every other input node before the next round.

4.2.1 Stochastic Reo Automata

In this section, we provide a compositional semantics for Stochastic Reo connectors, as an extension of the Reo Automata of Section 2.3.3 with functions that assign stochastic values for data-flows and I/O request arrivals.

Definition 4.2.1 (Stochastic Reo Automata). *A Stochastic Reo Automaton is a triple* $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ *with a Reo Automaton* $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ *according to Definition 2.3.7 and*

- $\mathbf{r} : \Sigma \to \mathbb{R}^+$ *is a function that associates with each node its arrival rate.*
- $\mathbf{t} : \delta_{\mathcal{A}} \to 2^{\Theta}$ *is a function that associates with a transition a subset of* $\Theta = 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^+$ *such that for any* $I, O \subseteq \Sigma$ *and* $I \cap O = \emptyset$*, each* $(I, O, r) \in \Theta$

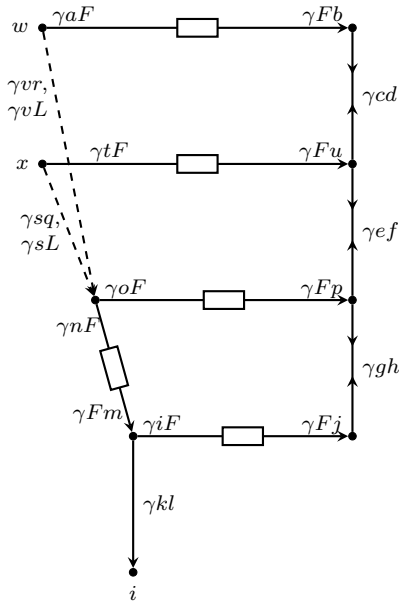


Figure 4.2: Stochastic Discriminator with two inputs

corresponds to a data-flow where I is a set of mixed and/or input nodes; O is a set of output and/or mixed nodes; and r is a processing delay rate for the data-flow described by I and O. We require that

- *for any two 3-tuples (I_1, O_1, r_1), $(I_2, O_2, r_2) \in \Theta$ such that $I_1 = I_2 \wedge O_1 = O_2$, it holds that $r_1 = r_2$, and*
- *for a transition $s \xrightarrow{g|f} s' \in \delta_{\mathcal{A}}$ with $\mathbf{t}(s \xrightarrow{g|f} s') = \{(I_1, O_1, r_1), (I_2, O_2, r_2), \ldots, (I_n, O_n, r_n)\}$, $f \setminus (I \cap O) = (I \cup O) \setminus (I \cap O)$ where $I = \bigcup_{1 \leq i \leq n} I_i$ and $O = \bigcup_{1 \leq i \leq n} O_i$.*

■

The Stochastic Reo Automata corresponding to the primitive Stochastic Reo channels in Figure 2.3 are defined by the functions $\mathbf{r}$ and $\mathbf{t}$ shown in Table 4.1. Note that the function $\mathbf{t}$ is encoded in the labels of the transitions of the automata, and the function $\mathbf{r}$ is shown inside the tables. For simplicity, here and in the remainder of this chapter, we simplify the representation of the 3-tuple (I, O, r), which is assigned by the function $\mathbf{t}$, by omitting the curly brackets for I and O and the commas between the elements in I and O.

An element of $\theta \in \Theta$ is accessed by projection functions $i : \Theta \to 2^{\Sigma}$, $o : \Theta \to 2^{\Sigma}$ and $v : \Theta \to \mathbb{R}^{+}$; $i(\theta)$ and $o(\theta)$ return the respective input and output nodes of

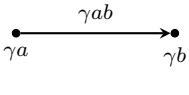
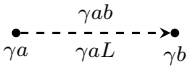
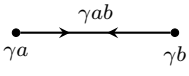
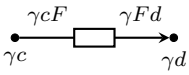
Synchronous Channels		
	$ab\|ab,\ \{(a,b,\gamma ab)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r} \\ \hline a & \gamma a \\ b & \gamma b \\ \hline \end{array}$
	$ab\|ab,\ \{(a,b,\gamma ab)\}$ $a\bar{b}\|a,\ \{(a,\emptyset,\gamma aL)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r} \\ \hline a & \gamma a \\ b & \gamma b \\ \hline \end{array}$
	$ab\|ab,\ \{(ab,\emptyset,\gamma ab)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r} \\ \hline a & \gamma a \\ b & \gamma b \\ \hline \end{array}$
Asynchronous Channel		
	$c\|c,\ \{(c,\emptyset,\gamma cF)\}$  $d\|d,\ \{(\emptyset,d,\gamma Fd)\}$	$\begin{array}{\|c\|c\|}\hline & \mathbf{r} \\ \hline c & \gamma c \\ d & \gamma d \\ \hline \end{array}$

Table 4.1: Stochastic Reo Automaton for some basic Stochastic Reo channels

a data-flow, and $v(\theta)$ returns the delay rate of the data-flow through nodes in $i(\theta)$ and $o(\theta)$.

As mentioned in Section 2.3.3, Reo Automata provide a compositional semantics for Reo connectors. As an extension of Reo Automata, Stochastic Reo Automata also present the composition of Stochastic Reo connectors at the automata level. For this purpose, we define the two operations of product and synchronization that are used to obtain an automaton of a Stochastic Reo connector by composing the automata of its primitive connectors. The compositionality of these operations is formally proved later in this section.

Definition 4.2.2 (Product). *Given two Stochastic Reo Automata* $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ *and* $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ *with* $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_{\mathcal{A}_1})$ *and* $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_{\mathcal{A}_2})$*, their product* $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ *is defined as* $(\mathcal{A}_1 \times \mathcal{A}_2, \mathbf{r}_1 \cup \mathbf{r}_2, \mathbf{t})$ *where*

$$
\begin{array}{lcll}
\mathbf{t}((q,p) \xrightarrow{gg'|ff'} (q',p')) & = & \mathbf{t}_1(q \xrightarrow{g|f} q') \cup \mathbf{t}_2(p \xrightarrow{g'|f'} p') & \\
 & & \multicolumn{2}{l}{\textit{where } q \xrightarrow{g|f} q' \in \delta_1 \wedge p \xrightarrow{g'|f'} p' \in \delta_2} \\
\mathbf{t}((q,p) \xrightarrow{gp^\sharp|f} (q',p)) & = & \mathbf{t}_1(q \xrightarrow{g|f} q') & \textit{where } q \xrightarrow{g|f} q' \in \delta_1 \wedge p \in Q_2 \\
\mathbf{t}((q,p) \xrightarrow{gq^\sharp|f} (q,p')) & = & \mathbf{t}_2(p \xrightarrow{g|f} p') & \textit{where } p \xrightarrow{g|f} p' \in \delta_2 \wedge q \in Q_1
\end{array}
$$

■

Note that we use $\times$ to denote both the product of Reo Automata and the product of Stochastic Reo Automata.

The set of 3-tuples that $\mathbf{t}$ associates with a transition m combines the delay rates involved in all data-flows synchronized by the transition m. For this combining, we might use a representative value for the synchronized data-flows, for example, the maximum of the delay rates. However, deciding the representative rate is not always desirable, and moreover, it can cause restriction to modeling random behavior of a system. In order to keep Stochastic Reo Automata generally useful and compositional, and their product commutative, we avoid fixing the precise formal meaning of distribution rates of synchronized transitions composed in a product. Instead, we represent the "delay rate" of a composite transition in the product automaton as the union of the delay rates of the synchronizing transitions of the two automata. The way these rates are combined to yield the composite rate of the transition depends on the different properties of the distributions and their time domains. For example, in the continuous-time case, no two events can occur at the same time; and we might choose the maximum one among the rates of the synchronized data-flows as their representative rate, but the exponential distributions are not closed under taking maximum. In Section 4.4, we show how to translate a Stochastic Reo Automaton to a CTMC using the union of the rates of the exponential distributions in the continuous-time case.

Definition 4.2.3 (Synchronization). *Given a Stochastic Reo Automaton* $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ *with* $\mathcal{A} = (\Sigma, Q, \delta)$*, the synchronization operation for nodes* a *and* b *is defined as* $\partial_{a,b}(\mathcal{A}, \mathbf{r}, \mathbf{t}) = (\partial_{a,b}\mathcal{A}, \mathbf{r}', \mathbf{t}')$ *where*

- $\mathbf{r}'$ *is* $\mathbf{r}$ *restricted to the domain* $\Sigma \setminus \{a, b\}$.
- $\mathbf{t}'$ *is defined as:*

$$\mathbf{t}'(q \xrightarrow{g\setminus_{ab}|f\setminus\{a,b\}} q') = \{(A', B', r) \mid (A, B, r) \in \mathbf{t}(q \xrightarrow{g|f} q'), \\ A' = sync(A, \{a, b\}) \wedge \ B' = sync(B, \{a, b\}) \ \}$$

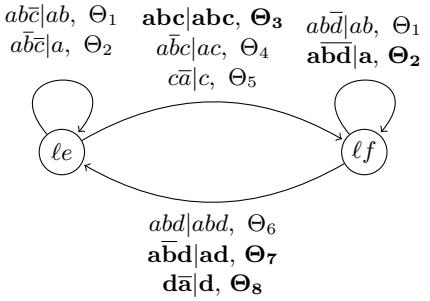
where $sync : 2^\Sigma \times 2^\Sigma \to 2^\Sigma$ *gathers nodes joined by synchronization, and is defined as:*

$$sync(A, B) = \begin{cases} A \cup B & \text{if } A \cap B \neq \emptyset \\ A & \text{otherwise} \end{cases}$$

■

Note that we use the symbol $\partial_{a,b}$ to denote both the synchronization of Reo Automata and the synchronization of Stochastic Reo Automata. The number of nodes joined by a synchronization is always two, and these joined nodes are gathered in a one set. The sets of joined nodes in multiple synchronization steps are disjoint. That is, given two different synchronizations $\partial_{a,b}$ and $\partial_{c,d}$ on a Stochastic Reo automaton, $\{a, b\} \cap \{c, d\} = \emptyset$.

Example 4.2.4. We now revisit the LossyFIFO1 example. Its semantics is given by the triple $(\mathcal{A}_{LossyFIFO1}, \mathbf{r}, \mathbf{t})$, where $\mathcal{A}_{LossyFIFO1}$ is the automaton depicted in Figure 2.11 and $\mathbf{r}$ is defined as $\mathbf{r} = \{a \mapsto \gamma a, b \mapsto \gamma b, c \mapsto \gamma c, d \mapsto \gamma d\}$. For $\mathbf{t}$, we first compute $\mathbf{t}_{LossySync \times FIFO1}$:



Θ_1 :	$\{(a, b, \gamma ab)\}$
Θ_2 :	$\{(a, \emptyset, \gamma aL)\}$
Θ_3 :	$\{(a, b, \gamma ab), (c, \emptyset, \gamma cF)\}$
Θ_4 :	$\{(a, \emptyset, \gamma aL), (c, \emptyset, \gamma cF)\}$
Θ_5 :	$\{(c, \emptyset, \gamma cF)\}$
Θ_6 :	$\{(a, b, \gamma ab), (\emptyset, d, \gamma Fd)\}$
Θ_7 :	$\{(a, \emptyset, \gamma aL), (\emptyset, d, \gamma Fd)\}$
Θ_8 :	$\{(\emptyset, d, \gamma Fd)\}$

Above, the labels that correspond to the transitions that will be kept after synchronization appear in **bold**. Thus, the result of joining nodes by synchronization, is shown in Figure 4.3 with $\mathbf{r}' = \{a \mapsto \gamma a, d \mapsto \gamma d\}$ which is restricted to its boundary nodes.

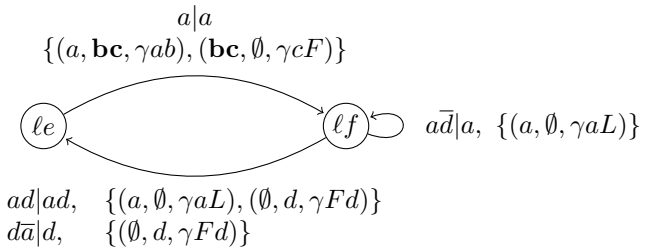


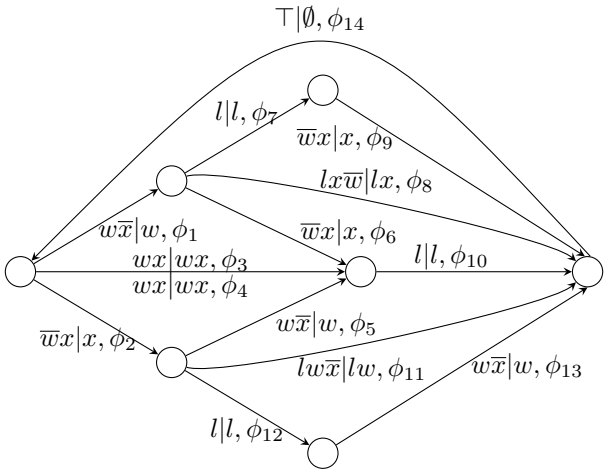
Figure 4.3: Stochastic Reo Automaton for LossyFIFO1

Note that the node names that appear in **bold** represent the synchronization of nodes b and c. For simplicity, here and in the remainder of this chapter, we use abbreviated representation of states, for example, the states ℓe and ℓf in the above automaton are abbreviations for (ℓ, e) and (ℓ, f). ◊

In this way, we can carry on stochastic information, i.e., arrival rates and processing delay rates that pertain to its QoS, in the semantic model of Reo circuits, given as Reo Automata.

As a more complex example of such composition, Figure 4.4 shows a Stochastic Reo Automaton for the discriminator in Figure 4.2.

Definition 4.2.1 shows that our extension of Reo Automata deals with such stochastic information separately, apart from the underlying Reo Automaton. Thus, our extended model retains the properties of Reo Automata, i.e., the compositionality result presented in Section 2.3.3 can be extended to Stochastic Reo Automata.



$$
\begin{array}{lcll}
\phi_1 & = & \{ & (w, av, \gamma avw),\ (av, r, \gamma vr),\ (r, no, \gamma nor), \\
 & & & \quad (no, \emptyset, \gamma oF),\ (no, \emptyset, \gamma nF),\ (av, \emptyset, \gamma aF)\ \} \\
\phi_2 & = & \{ & (x, st, \gamma stx),\ (st, \emptyset, \gamma tF),\ (st, q, \gamma sq), \\
 & & & (q, no, \gamma noq),\ (no, \emptyset, \gamma oF),\ (no, \emptyset, \gamma nF)\ \} \\
\phi_3 & = & & \phi_2 \cup \phi_5 \\
\phi_4 & = & & \phi_1 \cup \phi_6 \\
\phi_5 & = & \{ & (w, av, \gamma avw),\ (av, \emptyset, \gamma vL),\ (av, \emptyset, \gamma aF)\ \} \\
\phi_6 & = & \{ & (x, st, \gamma stx)\ , (st, \emptyset, \gamma tF),\ (st, \emptyset, \gamma sL)\ \} \\
\phi_7 & = & \{ & (\emptyset, m, \gamma Fm),\ (m, ik, \gamma ikm),\ (ik, l, \gamma kl), \\
 & & & (ik, \emptyset, \gamma iF)\ \} \\
\phi_8 & = & & \phi_7 \cup \phi_6 \\
\phi_9 & = & & \phi_6 \\
\phi_{10} & = & & \phi_{12}\ =\ \phi_7 \\
\phi_{11} & = & & \phi_7 \cup \phi_5 \\
\phi_{13} & = & & \phi_5 \\
\phi_{14} & = & \{ & (\emptyset, bc, \gamma Fb),\ (bcde, \emptyset, \gamma cd),\ (\emptyset, u, \gamma Fu) \\
 & & & (u, de, \gamma deu),\ (\emptyset, p, \gamma Fp),\ (defg, \emptyset, \gamma ef), \\
 & & & (p, fg, \gamma fgp),\ (fghj, \emptyset, \gamma gh), (\emptyset, hj, \gamma Fj)\ \}
\end{array}
$$

Figure 4.4: Stochastic Reo Automaton for discriminator in Figure 4.2

Given two Stochastic Reo Automata $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ and $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ with $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$ over the disjoint alphabets Σ_1 and Σ_2, and two subsets $\{a_1, \ldots, a_k\} \subseteq \Sigma_1$ and $\{b_1, \ldots, b_k\} \subseteq \Sigma_2$, we construct $\partial_{a_1,b_1}\partial_{a_2,b_2}\cdots\partial_{a_k,b_k}(\mathcal{A}_1 \times \mathcal{A}_2)$ as the automaton corresponding to a connector where node a_i of the first connector is connected to node b_i of the second connector, for all $i \in \{1, \ldots, k\}$. Note that the

'plugging' order does not matter because ∂ can be applied in any order and it interacts well with the product. These properties are captured in the following lemma.

Lemma 4.2.5 (Compositionality). *Given two disjoint Stochastic Reo Automata* $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$ *and* $(\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$ *with* $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ *and* $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$,

1. $\partial_{a,b}\partial_{c,d}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) = \partial_{c,d}\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)$, *if* $a, b, c, d \in \Sigma_1$
2. $(\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1)) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2) \sim \partial_{a,b}((\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2))$, *if* $a, b \notin \Sigma_2$

Here $(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \sim (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)$, *where* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are automata over the same alphabet, if and only if* $\mathcal{A}_1 \sim \mathcal{A}_2$, $\mathbf{r}_1 = \mathbf{r}_2$, *and* $\mathbf{t}_1 = \mathbf{t}_2$. ♦

Proof. For the first proposition, let

- $\partial_{a,b}\partial_{c,d}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) = (\partial_{a,b}\partial_{c,d}\mathcal{A}_1, \mathbf{r}'_1, \mathbf{t}'_1)$ and
- $\partial_{c,d}\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) = (\partial_{c,d}\partial_{a,b}\mathcal{A}_1, \mathbf{r}''_1, \mathbf{t}''_1)$

By [19, Lemma 4.13] which is the analogue result for Reo Automata, we know that $\partial_{a,b}\partial_{c,d}\mathcal{A}_1 = \partial_{c,d}\partial_{a,b}\mathcal{A}_1$. Using basic set theory, we also have that

$$\begin{array}{rcl} \mathbf{r}'_1 & = & \mathbf{r} \mid (\Sigma \setminus \{a,b\}) \setminus \{c,d\} \\ & = & \mathbf{r} \mid (\Sigma \setminus \{c,d\}) \setminus \{a,b\} \\ & = & \mathbf{r}''_1 \end{array}$$

where for $x \subseteq \Sigma$, $\mathbf{r}|x$ is the restriction of $\mathbf{r}$ to x.

Before moving to the fact that $\mathbf{t}'_1 = \mathbf{t}''_1$, we show that the order of applying the synchronization is irrelevant for the synchronization result, i.e., given three node sets A, $\{a,b\}$, and $\{c,d\}$, and the synchronization function in Definition 4.2.3,

$$sync(sync(A, \{a,b\}), \{c,d\}) = sync(sync(A, \{c,d\}), \{a,b\})$$

because, given three node sets A, B, and C with $B \cap C = \emptyset$,

$$sync(sync(A,B),C) = \begin{cases} A \cup B \cup C & \text{if } A \cap B \neq \emptyset \ \wedge \ A \cap C \neq \emptyset \\ A \cup B & \text{if } A \cap B \neq \emptyset \ \wedge \ A \cap C = \emptyset \\ A \cup C & \text{if } A \cap B = \emptyset \ \wedge \ A \cap C \neq \emptyset \\ A & \text{otherwise} \end{cases}$$

and set union $\cup$ is commutative. Now

$$\begin{array}{rl} & \mathbf{t}'_1(q \xrightarrow{g \setminus abcd \mid (f \setminus \{a,b\}) \setminus \{c,d\}} q') \\ = & \{(A', B', r) \mid (A, B, r) \in \mathbf{t}_1(q \xrightarrow{g|f} q'), \\ & \qquad A'' = sync(A, \{a,b\}) \ \wedge \ B'' = sync(B, \{a,b\}) \ \wedge \\ & \qquad A' = sync(A'', \{c,d\}) \ \wedge \ B' = sync(B'', \{c,d\})\} \\ = & \{(A', B', r) \mid (A, B, r) \in \mathbf{t}_1(q \xrightarrow{g|f} q'), \\ & \qquad A'' = sync(A, \{c,d\}) \ \wedge \ B'' = sync(B, \{c,d\}) \ \wedge \\ & \qquad A' = sync(A'', \{a,b\}) \ \wedge \ B' = sync(B'', \{a,b\})\} \\ = & \mathbf{t_1}''(q \xrightarrow{g \setminus cdab \mid (f \setminus \{c,d\}) \setminus \{a,b\}} q') \end{array}$$

For the second proposition, let

- $\partial_{a,b}(\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_1) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2) = (\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2, \mathbf{r}, \mathbf{t})$ and
- $\partial_{a,b}((\mathcal{A}_1, \mathbf{r}_1, \mathbf{t}_2) \times (\mathcal{A}_2, \mathbf{r}_2, \mathbf{t}_2)) = (\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2), \mathbf{r}', \mathbf{t}')$

By [19, Lemma 4.13], we know that $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2 \sim \partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$ if $a, b \notin \Sigma_2$. It remains to prove that $\mathbf{r} = \mathbf{r}'$ and $\mathbf{t} = \mathbf{t}'$.

For the first part, we easily calculate:

$$\mathbf{r}(p) \quad = \quad \begin{cases} \mathbf{r}_1(p) & \text{if } \ p \in \Sigma_1 \setminus \{a,b\} \\ \mathbf{r}_2(p) & \text{if } \ p \in \Sigma_2 \end{cases} \quad = \quad \mathbf{r}'(p)$$

For the second part, consider transitions $(q_1, q_2) \xrightarrow{(g_1 \setminus ab) g_2 | (f_1 \setminus \{a,b\}) f_2} (p_1, p_2)$ in $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2$ and $(q_1, q_2) \xrightarrow{(g_1 g_2) \setminus ab | (f_1 f_2) \setminus \{a,b\}} (p_1, p_2)$ in $\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$ with $g_i \in \mathcal{B}_{\Sigma_i}$ and $f_i \in 2^{\Sigma_i}$ for $i = 1, 2$, which includes joined nodes a and b. Then

$$\begin{array}{lll}
& \mathbf{t}((q_1, q_2) \xrightarrow{(g_1 \setminus ab) g_2 | (f_1 \setminus \{a,b\}) f_2} (p_1, p_2)) & \\
= & \{(A', B', r) \mid (A, B, r) \in \mathbf{t}_1(q_1 \xrightarrow{g_1 | f_1} p_1), & \\
& \qquad\qquad A' = sync(A, \{a,b\}) \ \wedge \ B' = sync(B, \{a,b\})\} & \\
\cup & \{(A, B, r) \mid (A, B, r) \in \mathbf{t}_2(q_2 \xrightarrow{g_2 | f_2} p_2)\} & \\
= & \{(A', B', r) \mid (A, B, r) \in \mathbf{t}_1(q_1 \xrightarrow{g_1 | f_1} p_1) \ \cup \ \mathbf{t}_2(q_2 \xrightarrow{g_2 | f_2} p_2), & \\
& \qquad\qquad A' = sync(A, \{a,b\}) \ \wedge \ B' = sync(B, \{a,b\})\} & \\
= & \mathbf{t}'((q_1, q_2) \xrightarrow{(g_1 g_2) \setminus ab | (f_1 f_2) \setminus \{a,b\}} (p_1, p_2)) &
\end{array}$$

Since $sync(C, D) = C$ if $C \cap D = \emptyset$, the above result holds without a need to consider if $ab \leq g_1$ or $\{a, b\} \subseteq f_1$. This also implies that $\mathbf{t} = \mathbf{t}'$ holds for transitions $(q, p) \xrightarrow{g_1 | f_1} (q', p)$ and $(q, p) \xrightarrow{g_2 | f_2} (q, p')$, which do not include joined nodes, in $\partial_{a,b}(\mathcal{A}_1) \times \mathcal{A}_2$ (equivalently, in $\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2)$). □

4.3 Reward model

The end-to-end QoS aspects considered in Stochastic Reo typically involve timing information. Stochastic Reo is, however, general enough to include other types of quantitative information and, in this section, we consider rewards to model, for instance, CPU computation time, memory space, etc. Rewards are assigned to request-arrivals or data-flows in Stochastic Reo. Assigning a reward is done in a similar fashion to annotating an activity with a stochastic rate. This similarity is leading in the following section which discusses the extension of Stochastic Reo Automata with reward information.

4.3.1 Stochastic Reo with reward information

To specify the rewards of the behavior of Reo connectors, we have extended Stochastic Reo by associating reward information to the stochastic activities of request-arrivals at channel ends and data-flows through channels. Intuitively, the reward information indicates the amount of resources required or released (gained or lost) for carrying out the relevant stochastic activities.

In our extension, reward information is not confined to specific types. Moreover, multiple types of rewards can be associated with a single stochastic activity. The type of each reward is labeled to its reward value, for instance, *memory space: 3*. Formally, the reward information is an element of $(\mathit{Types} \times \mathbb{R})^*$ where *Types* is a set of reward types. For simplicity, here and throughout of this thesis, we do not explicitly mention reward types and assume that they are implied by the positions of the values in each sequence. Thus, we use $\mathbb{R}^*$ instead of $(\mathit{Types} \times \mathbb{R})^*$, where $\mathbb{R}^*$ is a sequence of real numbers, which we shall call a *reward sequence*. Let π be a reward sequence. The ith element of π is denoted by $\pi(i)$ and the length of π, by $|\pi|$. Implicitly, each $\pi(i)$ is associated with a certain type of reward.

Figure 4.5 shows some primitive Stochastic Reo channels with stochastic rates and reward sequences for stochastic activities. We associate a pair of a reward sequence and a stochastic rate with each of their relevant stochastic activities, represented in the format (*rate* | *reward sequence*).

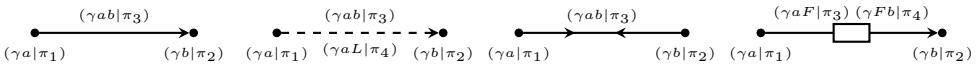


Figure 4.5: Some basic Stochastic Reo channels with rewards

For efficient reasoning about the rewards, we assume that all reward sequences of the same connector have the same length. For instance, for π_1, π_2, π_3 in the Sync channel in Figure 4.5, $|\pi_1| = |\pi_2| = |\pi_3|$. In addition, the reward values positioned at the same index in the reward sequences from the same connector must have the same type. For example, in the context of the FIFO1 channel in Figure 4.5: Let the reward types of π_3 associated with the data-flow from node a to the buffer be [*memory space*, *computation time*], then for the reward sequence π_4 associated with the data-flow from the buffer to node b, $|\pi_4| = 2$ and the order of reward types in π_4 must also be [*memory space*, *computation time*].

Stochastic Reo extended with reward information retains the compositionality of Stochastic Reo. As mentioned in Section 4.2, we assume that pumping data at mixed nodes is an immediate activity, thus, the rates of mixed nodes are considered as ∞. In the case of rewards, even if pumping data does not consume any time it still requires some rewards/resource to carry out this activity. We need to define how to determine reward sequences for pumping data at mixed nodes. For this purpose, we recall next the definition of a constraint semiring (c-semiring, for short).

Definition 4.3.1. ***(Constraint semiring [18])*** *A constraint semiring is a structure* $(C, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ *where* C *is a set,* $\mathbf{0}, \mathbf{1} \in C$*, and* $\oplus$ *and* $\otimes$ *are binary operations on* C *such that:*

- $\oplus$ *is commutative, associative, idempotent;* $\mathbf{0}$ *is its unit element, and* $\mathbf{1}$ *is its absorbing element.*
- $\otimes$ *is commutative, associative and distributes over* $\oplus$*;* $\mathbf{1}$ *is its unit element, and* $\mathbf{0}$ *is its absorbing element.*

■

It should be noted that the operation $\oplus$ induces a partial order $\trianglelefteq$ on C, which is defined by $c \trianglelefteq c'$ iff $c \oplus c' = c'$.

The c-semiring structure is appropriate when only one calculation is possible over its domain. In the case of Reo connectors, we need two different types of calculations: a sequential calculation through the connector for its overall reward and a parallel calculation for joining nodes into a mixed node. For these two different types of calculations, another algebraic structure, called Q-algebra, is used. We recall the definition of Q-algebra; here Q refers to the *QoS* or *quantitative* values.

Definition 4.3.2. ***(Q-algebra [29])*** *A Q-algebra is a structure* $\mathcal{R} = (C, \oplus, \otimes, \oplus, \mathbf{0}, \mathbf{1})$ *such that* $\mathcal{R}_{\otimes} = (C, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ *and* $\mathcal{R}_{\oplus} = (C, \oplus, \oplus, \mathbf{0}, \mathbf{1})$ *are both c-semirings.* C *is called* the domain of $\mathcal{R}$. ■

The set C is a set of reward values, and the operations $\otimes$ and $\oplus$ calculate rewards whose relevant activities occur sequentially and in parallel, respectively. That is, given $c_1,\ c_2 \in C$, $c_1 \otimes c_2$ is the composed reward of when c_2 follows c_1, and $c_1 \oplus c_2$ is the composed reward of when c_1 and c_2 occur concurrently. For instance, the Q-algebra for the shortest computation time can be given as $(\mathbb{R}^+ \cup \{\infty\}, min, +, +, \infty, 0)$.

The product of two Q-algebra is defined as:

Definition 4.3.3. ***(Product [29])*** *For two Q-algebras* $\mathcal{R}_1 = (C_1, \oplus_1, \otimes_1, \oplus_1, \mathbf{0}_1, \mathbf{1}_1)$ *and* $\mathcal{R}_2 = (C_2, \oplus_2, \otimes_2, \oplus_2, \mathbf{0}_2, \mathbf{1}_2)$*, their product is* $\mathcal{R}_1 \diamond \mathcal{R}_2 = \mathcal{R} = (C, \oplus, \otimes, \oplus, \mathbf{0}, \mathbf{1})$ *where*

- $C = C_1 \times C_2$
- $(c_1, c_2) \oplus (c_1', c_2') = (c_1 \oplus_1 c_1',\ c_2 \oplus_2 c_2')$
- $(c_1, c_2) \otimes (c_1', c_2') = (c_1 \otimes_1 c_1',\ c_2 \otimes_2 c_2')$
- $(c_1, c_2) \oplus (c_1', c_2') = (c_1 \oplus_1 c_1',\ c_2 \oplus_2 c_2')$
- $\mathbf{0} = (\mathbf{0}_1, \mathbf{0}_2)$
- $\mathbf{1} = (\mathbf{1}_1, \mathbf{1}_2)$

■

To deal with different types of rewards, we label them, as mentioned above, as elements of $(Types \times \mathbb{R})^*$, and use *a labeled Q-algebra*, which is defined as follows.

Definition 4.3.4. ***(A labeled Q-algebra [29])*** *For each $1 \leq i \leq n$, let $\mathcal{R}_i = (C_i, \oplus_i, \otimes_i, \oplus\!\!\!\cdot_i, \mathbf{0}_i, \mathbf{1}_i)$ be a Q-algebra. Associating distinct label l_i with each $\mathcal{R}_i$, denoted by $(l_i : \mathcal{R}_i)$, such that $l_i \neq l_j$ if $i \neq j$, the product of $\mathcal{R}_i$ is a labeled Q-algebra $\mathcal{R} = (C, \oplus, \otimes, \oplus\!\!\!\cdot, \mathbf{0}, \mathbf{1})$ if*

- $C = (\{l_1\} \times C_1) \times \cdots \times (\{l_n\} \times C_n)$
- $(l_1 : c_1, \ldots, l_n : c_n) \oplus (l_1 : c'_1, \ldots, l_n : c'_n) = (l_1 : (c_1 \oplus_1 c'_1), \ldots, l_n : (c_n \oplus_n c'_n))$
- $(l_1 : c_1, \ldots, l_n : c_n) \otimes (l_1 : c'_1, \ldots, l_n : c'_n) = (l_1 : (c_1 \otimes_1 c'_1), \ldots, l_n : (c_n \otimes_n c'_n))$
- $(l_1 : c_1, \ldots, l_n : c_n) \oplus\!\!\!\cdot (l_1 : c'_1, \ldots, l_n : c'_n) = (l_1 : (c_1 \oplus\!\!\!\cdot_1 c'_1), \ldots, l_n : (c_n \oplus\!\!\!\cdot_n c'_n))$
- $\mathbf{0} = (l_1 : \mathbf{0}_1, \ldots, l_n : \mathbf{0}_n)$
- $\mathbf{1} = (l_1 : \mathbf{1}_1, \ldots, l_n : \mathbf{1}_n)$

■

The product operation on Q-algebras in Definition 4.3.3 can be applied to labeled Q-algebras only if the order of the labels of two Q-algebras are identical.

Now we show how to comprise/obtain the reward for mixed nodes. When joining a sink node and a source node into a mixed node, the resulting reward for the mixed node is obtained using $\oplus\!\!\!\cdot$ on two rewards of each node since respective activities for each node occur in parallel. That is, let a mixed node be composed out of a sink node a and a source node b whose respective rewards are π_a and π_b, then the resulting reward for the mixed node is $\pi_a \oplus\!\!\!\cdot \pi_b$.

We now consider a merger and a replicator with reward information. As mentioned in Section 2.1, a merger selects its source node and dispenses a data item from the selected source node to its sink node. Consider the merger in Figure 4.6[1]. This merger has 3 source nodes a, b, and c and a sink node d whose respective rewards are π_a, π_b, π_c, and π_d, then the comprised reward for this merger is $(\pi_a \oplus \pi_b \oplus \pi_c) \otimes \pi_d$.

(a) merger (b) replicator

Figure 4.6: Magnified merger and replicator

[1]Note that for simplicity, a merger and a replicator are usually depicted as mixed nodes, but here we magnify them in order to explain how to calculate their rewards. However, the names of their source and sink nodes are omitted.

In the case of a replicator, it takes a data-item from its source node and dispenses the duplication of the data-item to its all sink nodes. Consider the replicator in Figure 4.6. It has a source node a and 3 sink nodes b, c, and d whose respective rewards are π_a, π_b, π_c, and π_d, then the comprised reward of this replicator is $\pi_a \otimes (\pi_b \oplus \pi_c \oplus \pi_d)$.

As an example for the composed reward of Reo connectors, Figure 4.7 depicts the Stochastic Reo extended with reward information for LossySync and FIFO1, together with the connector resulting from the composition of the two.

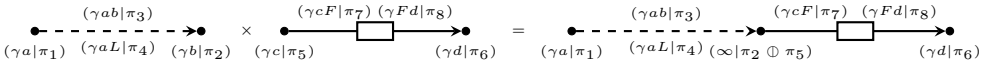


Figure 4.7: Stochastic Reo for LossyFIFO1 with rewards

Note that the $\times$ notation in Figure 4.7 represents joining the source node in a LossySync channel and the sink node in a FIFO1 channel.

4.3.2 Stochastic Reo Automata with reward information

As an operational semantic model for Stochastic Reo extended with reward information, we introduce an extended Stochastic Reo Automata model in this section (Definition 4.3.5). The reward information, which is described using reward sequences in Stochastic Reo, is propagated to the semantic model by pairing a stochastic rate with its relevant reward sequence.

Before moving to the definition of the semantic model for Stochastic Reo extended with reward information, we slightly modify extended Stochastic Reo to reuse the operations and the properties of Stochastic Reo Automata described in the previous sections. This modification is necessary to accommodate the rewards for mixed nodes. The original Stochastic Reo discards the rates for mixed nodes, but the extended Stochastic Reo explicitly represents them as ∞ to make a pair with the rewards for mixed nodes. In order to deal with this difference and reuse the methods for the original Stochastic Reo, an actual mixed node is replaced with an auxiliary Sync channel, and newly arising mixed nodes are considered as usual in the original Stochastic Reo. That is, the new mixed nodes are assumed not to consume any time and entail no rewards for pumping data. For compliance with this assumption, the LossyFIFO1 connector in Figure 4.7 is modified as follows:

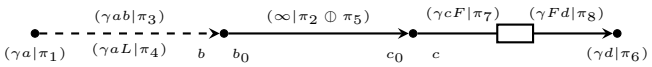


Figure 4.8: Modified LossyFIFO1 connector

Note that arbitrary names can be used for new mixed nodes in the modified con-

nectors, but here we use names as b_0 and c_0 to compare its corresponding automata model to the original Stochastic Reo Automaton corresponding to a LossyFIFO1 connector later. Adding an auxiliary Sync channel does not change the semantics of a connector and enables us to reuse the existing operations for Stochastic Reo Automata.

Definition 4.3.5. *[Stochastic Reo Automata extended with reward information] A Stochastic Reo Automaton with reward information is a tuple* $(\mathcal{A}, \mathbf{r}', \mathbf{t}', \mathcal{R})$ *where* $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ *is a Reo Automaton and*

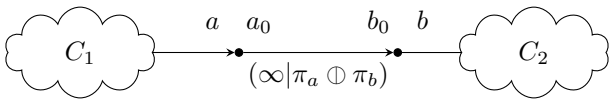
- $\mathbf{r}' : \Sigma \to \mathbb{R}^+ \times \mathbb{R}^*$ *is a function that associates with each node a pair consisting of an arrival rate and a reward sequence.*
- $\mathbf{t}' : \delta_{\mathcal{A}} \to 2^{\Psi}$ *is a function that associates with a transition* $q \xrightarrow{g|f} q' \in \delta_{\mathcal{A}}$ *a subset of* $\Psi = 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^+ \times \mathbb{R}^*$.
- $\mathcal{R}$*: a labeled Q-algebra* $(C, \oplus, \otimes, \oslash, \mathbf{0}, \mathbf{1})$ *with domain* C *of rewards.*

■

For each $\psi \in \Psi$, the projection functions that access its elements are $I : \Psi \to 2^{\Sigma}$, $O : \Psi \to 2^{\Sigma}$, $V : \Psi \to \mathbb{R}^+$, $R : \Psi \to \mathbb{R}^*$, and $pair : \Psi \to \mathbb{R}^+ \times \mathbb{R}^*$. I, O, and V return input nodes, output nodes, and its relevant rate, respectively, which correspond to i, o, and v in Section 4.2. R projects a relevant reward sequence from ψ. The function $pair$ returns a pair of a rate and its relevant reward sequence from ψ. We use $rate : \mathbb{R}^+ \times \mathbb{R}^* \to \mathbb{R}^+$ and $rew : \mathbb{R}^+ \times \mathbb{R}^* \to \mathbb{R}^*$ to access the elements of the results of the function $\mathbf{r}'$ and the function $pair$.

Table 4.2 shows Stochastic Reo Automata extended with reward information corresponding to the basic Stochastic Reo channels in Figure 4.5.

Now we show that using auxiliary Sync channels to retain rewards for mixed nodes does not affect the structure of Stochastic Reo Automata at all. Consider two connectors $\mathcal{C}_1$ and $\mathcal{C}_2$ with their boundary nodes, respectively, a and b, and these two connectors are connected by a Sync channel with ends a_0 and b_0 as follows:



Note that no rewards are assigned for the mixed nodes in the above connector, thus, we can reuse the definitions and the properties of the product and the synchronization for Stochastic Reo Automata, as mentioned in Section 4.2.

When the Stochastic Reo Automata extended with reward information for the connectors $\mathcal{C}_1$ and $\mathcal{C}_2$ are $(\mathcal{A}_1, \mathbf{r}'_1, \mathbf{t}'_1, \mathcal{R})$ with $\mathcal{A}_1 = (\Sigma_1, Q_1, \delta_1)$ and $(\mathcal{A}_2, \mathbf{r}'_2, \mathbf{t}'_2, \mathcal{R})$ with $\mathcal{A}_2 = (\Sigma_2, Q_2, \delta_2)$, respectively, the composition result of $\mathcal{C}_1$, $\mathcal{C}_2$, and the $\mathsf{Sync}(a_0, b_0)$ with $a_0,\ b_0 \notin \Sigma_1 \cup \Sigma_2$ is given by:

$$
\begin{aligned}
&(\partial_{b,b_0}((\partial_{a,a_0}(\mathcal{A}_1 \times \mathsf{Sync}(a_0, b_0))) \times \mathcal{A}_2), \mathbf{r}', \mathbf{t}', \mathcal{R}) \\
&= (\partial_{b,b_0}(\mathcal{A}_1[b_0/a] \times \mathcal{A}_2), \mathbf{r}', \mathbf{t}', \mathcal{R}) \qquad (1) \\
&= (\partial_{a,b}(\mathcal{A}_1 \times \mathcal{A}_2), \mathbf{r}', \mathbf{t}', \mathcal{R}) \qquad (2)
\end{aligned}
$$

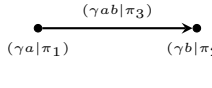
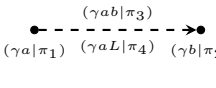
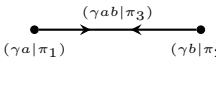
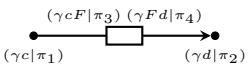
Synchronous Channels		
	$ab\|ab,\ \{(a,b,\gamma ab,\pi_3)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r}' \\ \hline a & (\gamma a,\pi_1) \\ b & (\gamma b,\pi_2) \\ \hline \end{array}$
	$ab\|ab,\ \{(a,b,\gamma ab,\pi_3)\}$ $a\bar{b}\|a,\ \{(a,\emptyset,\gamma aL,\pi_4)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r}' \\ \hline a & (\gamma a,\pi_1) \\ b & (\gamma b,\pi_2) \\ \hline \end{array}$
	$ab\|ab,\ \{(ab,\emptyset,\gamma ab,\pi_3)\}$ 	$\begin{array}{\|c\|c\|}\hline & \mathbf{r}' \\ \hline a & (\gamma a,\pi_1) \\ b & (\gamma b,\pi_2) \\ \hline \end{array}$
Asynchronous Channel		
	$c\|c,\ \{(c,\emptyset,\gamma cF,\pi_3)\}$  $d\|d,\ \{(\emptyset,d,\gamma Fd,\pi_4)\}$	$\begin{array}{\|c\|c\|}\hline & \mathbf{r}' \\ \hline c & (\gamma c,\pi_1) \\ d & (\gamma d,\pi_2) \\ \hline \end{array}$

Table 4.2: Stochastic Reo Automaton extended with reward information

where

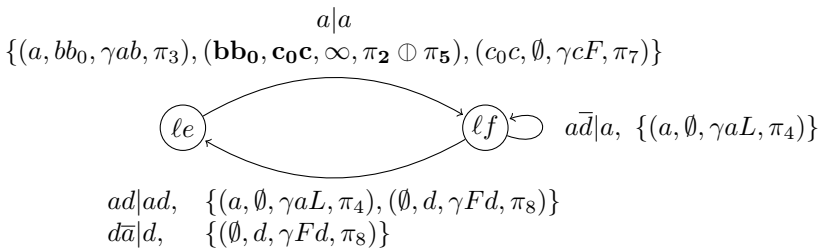
$$\begin{aligned}\mathbf{r}' &= \mathbf{r}'_1 \cup \mathbf{r}'_2 \cup \mathbf{r}'_{Sync(a_0b_0)} | (\Sigma_1 \cup \Sigma_2 \cup \{a_0, b_0\}) \setminus \{a, a_0, b, b_0\} \\ &= \mathbf{r}'_1 \cup \mathbf{r}'_2 | (\Sigma_1 \cup \Sigma_2) \setminus \{a, b\}\end{aligned}$$

since the $\mathbf{r}'$ function is defined only for boundary nodes, and

$$\begin{aligned}&\mathbf{t}'((q,p) \xrightarrow{gg'|ff'} (q',p')) = \mathbf{t}'_1(q \xrightarrow{g|f} q') \cup \mathbf{t}'_2(p \xrightarrow{g'|f'} p') \\ &\qquad\qquad \cup \{(\{a,a_0\},\{b_0,b\},\infty,\pi_a \odot \pi_b)\} \\ &\mathbf{t}'((q,p) \xrightarrow{gp^\sharp|f} (q',p)) = \mathbf{t}'_1(q \xrightarrow{g|f} q') \\ &\mathbf{t}'((q,p) \xrightarrow{gq^\sharp|f} (q,p')) = \mathbf{t}'_2(p \xrightarrow{g|f} p')\end{aligned}$$

Equality (1) follows by [19, Lemma 4.13] and (2) by an easily proven substitution property of node names.

The above implies that a reward sequence goes along with the relevant rate associated with a transition and does not affect the structure of Stochastic Reo Automata at all. Therefore, without loss of generality, Stochastic Reo Automata extended with reward information also support compositional specification and describe context-dependent connectors. Using the definitions for the composition of Stochastic Reo Automata in Section 4.2, the following figure shows the Stochastic Reo Automaton extended with reward information, corresponding to the modified LossyFIFO1 connector in Figure 4.8:



This result has the same structure as that of the Stochastic Reo Automaton in Figure 4.3, except for the 3-tuple $(bb_0, c_0c, \infty, \pi_2 \uplus \pi_5)$ which contains the reward information for the mixed node in the LossyFIFO1 connector in Figure 4.7.

4.4 Translation into CTMC

In this section, we show how to translate a Stochastic Reo Automaton into a homogeneous CTMC model. This translation is similar to the translation from QIA into CTMCs explained in Section 3.3, hence, to avoid repetition, in this section, we only show the different translation steps while using the same notations, if possible.

A CTMC model derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is a pair (S, δ). With a set of boundary nodes $\Sigma' \subseteq \Sigma$, the set S_A and the preliminary set of request-arrival transitions of the CTMC derived for $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ are defined as:

$$\begin{aligned} S_A &= \{\langle q, P\rangle \mid q \in Q,\ P \subseteq \Sigma'\} \\ \delta'_{Arr} &= \{\langle q, P\rangle \xrightarrow{\mathbf{r}(c)} \langle q, P \cup \{c\}\rangle \mid \langle q, P\rangle,\ \langle q, P \cup \{c\}\rangle \in S_A,\ c \notin P\} \end{aligned}$$

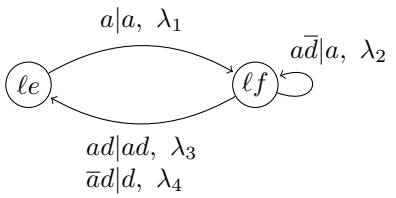
The set δ'_{Arr} is used to define δ_{Arr}, which includes preemptive request-arrivals arising in this translation process, used in the definition of δ above.

4.4.1 Synchronized data-flows

Synchronized data-flows are represented in a single transition of a Stochastic Reo Automaton. To divide this macro-step transition, corresponding to the synchronized data-flows, into a number of micro-step transitions, corresponding to each data-flow, the occurrence order of the synchronized data-flows need to be determined. This decision step is explained in Section 3.3.2 using a delay-sequence and Algorithm 3.3.1. Applying Algorithm 3.3.1 to the LossyFIFO1 example of Figure 4.3 yields the following result in Figure 4.9:

4.4.2 Deriving the CTMC

We now show how to derive the transitions in the CTMC model from the transitions in a Stochastic Reo Automaton. We do this in two steps:



λ_1 :	$(a, bc, \gamma ab)$; $(bc, \emptyset, \gamma cF)$
λ_2 :	$(a, \emptyset, \gamma aL)$
λ_3 :	$(a, \emptyset, \gamma aL)$ | $(\emptyset, d, \gamma Fd)$
λ_4 :	$(\emptyset, d, \gamma Fd)$

Figure 4.9: Extracting delay-sequences

1. For each transition $p \xrightarrow{g|f} q \in \delta_{\mathcal{A}}$, we derive transitions $\langle p, P\rangle \xrightarrow{\lambda} \langle q, P \setminus f\rangle$ for every set of pending requests P that suffices to activate the guard g (i.e., $\widehat{P} \leq g \setminus \widehat{\overline{\Sigma}}$), where λ is the delay-sequence associated with the set of 3-tuples $\mathbf{t}(p \xrightarrow{g|f} q)$. This set of derived transitions is defined below as δ_{Macro}.

2. We divide a transition in δ_{Macro} labeled by λ into a combination of micro-step transitions, each of which corresponds to a single event.

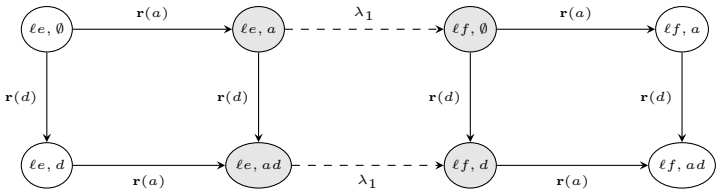
Given a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of boundary nodes Σ', a macro-step transition relation for the synchronized data-flows is defined as:

$$\delta_{Macro} = \{\langle p, P\rangle \xrightarrow{\lambda} \langle q, P \setminus f\rangle \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ P \subseteq \Sigma',\ \widehat{P} \leq g \setminus \widehat{\overline{\Sigma}},\ \lambda = \mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))\}$$

As an example of obtaining a macro-step transition relation, let us consider the transition $\ell e \xrightarrow{a|a,\lambda_1} \ell f$ with $\lambda_1 = (a, bc, \gamma ab)$; $(bc, \emptyset, \gamma cF)$ in Figure 4.9. Given the guard $g = a$ and the set of boundary nodes $\Sigma' = \{a, d\}$, $g \setminus \widehat{\overline{\Sigma}} = a \setminus \overline{a}\overline{b}\overline{c}\overline{d} = a$, and P is $\emptyset$, $\{a\}$, $\{d\}$, or $\{a, d\}$. Thus,

$$\widehat{P} = \begin{cases} a & \text{if } P = \{a\} \\ d & \text{if } P = \{d\} \\ ad & \text{if } P = \{a, d\} \\ \top & \text{otherwise} \end{cases}$$

Then, $\widehat{P} \leq g \setminus \widehat{\overline{\Sigma}}$ is satisfied when P is either $\{a\}$ or $\{a, d\}$, i.e., $a \leq a$ and $ad \leq a$. This generates the following macro-step transitions $\langle \ell e, a\rangle \xrightarrow{\lambda_1} \langle \ell f, \emptyset\rangle$ and $\langle \ell e, ad\rangle \xrightarrow{\lambda_1} \langle \ell f, d\rangle$, and these transitions are represented as dashed transitions in the state diagram that includes S_A and δ'_{Arr} as follows:



We explicate a macro-step transition with a number of micro-step transitions, each of which corresponds to a single data-flow. The detailed technical explanation on this division of a delay-sequence has been shown in Section 3.3.3. Thus, we skip the explanation of the division in this chapter.

The division into micro-step transitions ensures that each transition has a single 3-tuple in its label. Thus, the micro-step transitions can be extracted as:

$$\delta_{Proc} = \{\langle p, P\rangle \xrightarrow{v(\theta)} \langle p', P'\rangle \mid \langle p, P\rangle \xrightarrow{\theta} \langle p', P'\rangle \in div(t) \text{ for all } t \in \delta_{Macro}\}$$

As mentioned in Section 3.3.4, splitting synchronized data-flows allows non-interfering events, in particular request-arrivals, to interleave with micro-step events, disregarding the strict sense of the atomicity of the synchronized data-flows. The consideration of these preemptive request-arrivals is explained in Section 3.3.4.

4.4.3 Rewards

In this section, we show the translation from the Stochastic Reo Automata extended with reward information into CTMCs with state reward. As mentioned in Section 4.3, the reward sequence is independent of the structure of its Stochastic Reo Automaton. Thus, for the generation of CTMCs with state rewards, the translation from Stochastic Reo Automata into CTMCs can be reused with a small modification. When the CTMC (S, δ) is derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$, the derived CTMC, thus, is $(S \times \mathbb{R}^*, \delta)$ for the extended Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}', \mathbf{t}', \mathcal{R})$ which is extended from $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with Q-algebra $\mathcal{R}$ for reward information:

- Stochastic Reo Automata
  - $\mathbf{r} : \Sigma \to \mathbb{R}^+$
  - $\mathbf{t} : \delta_{\mathcal{A}} \to 2^{\Theta}$ where $\Theta \subseteq 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^+$
- Stochastic Reo Automata with reward information
  - $\mathbf{r}' : \Sigma \to \mathbb{R}^+ \times \mathbb{R}^*$
  - $\mathbf{t}' : \delta_{\mathcal{A}} \to 2^{\Psi}$ where $\Psi \subseteq 2^{\Sigma} \times 2^{\Sigma} \times \mathbb{R}^+ \times \mathbb{R}^*$

Note that the extensions of $\mathbf{r}'$ and $\mathbf{t}'$ by adding a reward sequence do not affect the structure of a connector, the structural information of which is used for our translation. Thus, $\mathbf{r}$ and $\mathbf{t}$ in the previous translation method can be replaced with, respectively, $\mathbf{r}'$ and $\mathbf{t}'$ easily.

In general, a state has more than one outgoing transition, which illustrates more than one activity is possible in a state. These activities have generally different rewards. Thus, we need to calculate the proper state reward considering all possible rewards. For this purpose, state rewards of CTMCs are decided after the whole diagram is drawn. This requires that the reward sequences should be kept until the complete CTMC diagram is drawn. The following shows the translation method into CTMCs considering this requirement.

While the CTMC derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is (S, δ), for the extended Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}', \mathbf{t}', \mathcal{R})$ with reward information, the complete CTMC diagram is described as a tuple (S, δ') where $S = S_A \cup S_M$ and $\delta' = \delta_{Arr} \cup \delta_{Proc} \subseteq S \times \mathbb{R}^+ \times \mathbb{R}^* \times S$. Each label on a transition in δ' is a pair of a rate, specifying request-arrivals and processing delay of the transition, and its relevant reward sequence.

For transitions of request-arrivals, given the Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}', \mathbf{t}', \mathcal{R})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of boundary nodes $\Sigma' \subseteq \Sigma$, the set S_A and the preliminary set of request-arrival transitions of the CTMC are defined as:

$$\begin{aligned} S_A &= \{\langle q, P\rangle \mid q \in Q,\ P \subseteq \Sigma'\} \\ \delta'_{Arr} &= \{\langle q, P\rangle \xrightarrow{\mathbf{r}'(c)} \langle q, P \cup \{c\}\rangle \mid \langle q, P\rangle, \langle q, P \cup \{c\}\rangle \in S_A,\ c \notin P\} \end{aligned}$$

The set δ'_{Arr} is used to define δ_{Arr} below.

For the division of synchronized data-flows, a new delay-sequence is redefined with the 4-tuple $\psi \in \Psi$:

$$\psi ::= \epsilon \mid \mu \mid \psi|\psi \mid \psi;\psi$$

The characteristics of the new delay-sequence μ is inherited from the existing delay-sequence λ in Section 3.3.1, and μ can also be extracted by Algorithm 3.3.1. Thus, the division of synchronized data-flows is carried out by the method mentioned in Section 4.4.2. The arrangement of labels on the divided result is described as:

$$\delta_{Proc} = \{\langle p, P\rangle \xrightarrow{pair(\mu)} \langle p', P'\rangle \in div(t) \mid t \in \delta_{Macro}\}$$

For the preemptive request-arrivals, with a set of micro-step states S_M, obtained through the division of synchronized data-flows, the full set of request-arrival transitions is defined as:

$$\delta_{Arr} = \delta'_{Arr} \cup \{\langle p, P\rangle \xrightarrow{\mathbf{r}'(d)} \langle p, P \cup \{d\}\rangle \mid \langle p, P\rangle,\ \langle p, P \cup \{d\}\rangle \in S_M, d \in \Sigma,\ d \notin P\}$$

So far we have derived a complete CTMC diagram from a Stochastic Reo Automaton extended with reward information, whereby the calculation of state rewards is shown below.

A state reward is decided by the outgoing transitions from each state, since the real values in the sequence represent the amount of resources that are required or released (gained or lost) for a transition materializing a request-arrival or a data-flow. When a label on a transition in δ' is denoted by $\kappa \in K \subseteq \mathbb{R}^+ \times \mathbb{R}^*$, the state

reward is obtained by a function $reward : S \to \mathbb{R}^*$: for every transition $s \xrightarrow{\kappa_1} s_1 \in \delta'$(e.g., $\delta_{Arr} \cup \delta_{Proc}$)

$$reward(s) \quad = \quad \begin{cases} \dfrac{\bigoplus_{i=1}^{n}(rate(\kappa_i) \boxdot rew(\kappa_i))}{\sum_{i=1}^{n} rate(\kappa_i)} & \text{if } \exists\ s \xrightarrow{\kappa_2} s_2 \in \delta', \cdots, \\ & \exists\ s \xrightarrow{\kappa_n} s_n \in \delta' \text{ and } s_i \neq s_j \text{ if } i \neq j \\ rew(\kappa_1) & \text{otherwise} \end{cases}$$

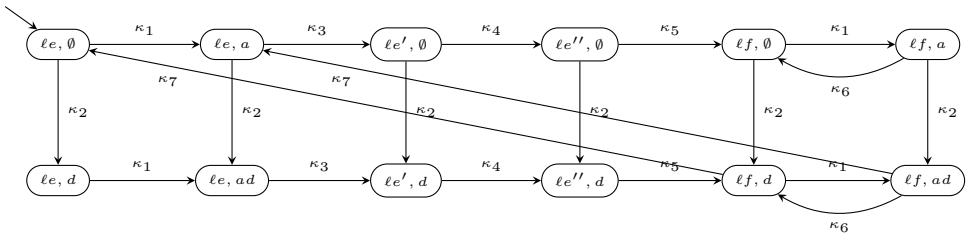
Note that $\boxdot : \mathbb{R}^+ \times \mathbb{R}^* \to \mathbb{R}^*$ is a function that returns the result of the multiplication of real number for its first parameter ($\mathbb{R}^+$) with every element in a reward sequence ($\mathbb{R}^*$), for instance, when $\pi = [\pi(0), \pi(1), \ldots, \pi(n)]$, the multiplication result of x and π is $x \boxdot \pi = [x \times \pi(0), x \times \pi(1), \ldots, x \times \pi(n)] \in \mathbb{R}^*$ where $\pi \in \mathbb{R}^*$ and $x, \pi(i) \in \mathbb{R}$ for $0 \leq i \leq n$. In addition, the summation $\boxplus : \mathbb{R}^* \times \mathbb{R}^* \to \mathbb{R}^*$ of reward sequences adds the values having the same index from each reward sequence. For example, for $\pi_1, \pi_2 \in \mathbb{R}^*,\ \pi_1 \boxplus \pi_2 = [\pi_1(1) + \pi_2(1), \pi_1(2) + \pi_2(2), \ldots, \pi_1(n) + \pi_2(n)] \in \mathbb{R}^*$. $\bigoplus_i^j$ is used to represent applying the summation $\boxplus$ for the reward sequences with the index from i to j.

After the calculation of state rewards, the extraction of the relevant rate for each transition δ' is done as:

$$\delta = \{s \xrightarrow{rate(\kappa)} s' \mid s \xrightarrow{\kappa} s' \in \delta'\}$$

The following example shows the calculation of state rewards from the complete CTMC diagram of the LossyFIFO1 connector extended with reward information.

Example 4.4.1. Consider the LossyFIFO1 example in Figure 4.7 and the CTMC diagram derived from it.



where the pairs κ_i of a rate and its corresponding reward sequence are shown below:

$$\begin{aligned} \kappa_1 &= (\gamma a, \pi_1) \\ \kappa_2 &= (\gamma d, \pi_6) \\ \kappa_3 &= (\gamma ab, \pi_3) \\ \kappa_4 &= (\infty, \pi_2 \oslash \pi_5) \\ \kappa_5 &= (\gamma cF, \pi_7) \\ \kappa_6 &= (\gamma aL, \pi_4) \\ \kappa_7 &= (\gamma Fd, \pi_8) \end{aligned}$$

Then, each state reward is given by:

$$
\begin{aligned}
reward_{\mathsf{LossyFIFO1}} = \{ \quad \langle \ell e, \emptyset \rangle &\mapsto \frac{(\gamma a \boxdot \pi_1) \boxplus (\gamma d \boxdot \pi_6)}{(\gamma a + \gamma d)} \\
\langle \ell e, a \rangle &\mapsto \frac{(\gamma ab \boxdot \pi_3) \boxplus (\gamma d \boxdot \pi_6)}{(\gamma ab + \gamma d)} \\
\langle \ell e', \emptyset \rangle &\mapsto \frac{(\infty \boxdot (\pi_2 \oplus \pi_5)) \boxplus (\gamma d \boxdot \pi_6)}{(\infty + \gamma d)} \\
\langle \ell e'', \emptyset \rangle &\mapsto \frac{(\gamma cF \boxdot \pi_7) \boxplus (\gamma d \boxdot \pi_6)}{(\gamma cF + \gamma d)} \\
\langle \ell f, \emptyset \rangle &\mapsto \frac{(\gamma a \boxdot \pi_1) \boxplus (\gamma d \boxdot \pi_6)}{(\gamma a + \gamma d)} \\
\langle \ell f, a \rangle &\mapsto \frac{(\gamma aL \boxdot \pi_4) \boxplus (\gamma d \boxdot \pi_6)}{(\gamma aL + \gamma d)} \\
\langle \ell e, d \rangle &\mapsto \pi_1 \\
\langle \ell e, ad \rangle &\mapsto \pi_3 \\
\langle \ell e', d \rangle &\mapsto \pi_2 \oplus \pi_5 \\
\langle \ell e'', d \rangle &\mapsto \pi_7 \\
\langle \ell f, d \rangle &\mapsto \frac{(\gamma a \boxdot \pi_1) \boxplus (\gamma Fd \boxdot \pi_8)}{(\gamma a + \gamma Fd)} \\
\langle \ell f, ad \rangle &\mapsto \frac{(\gamma aL \boxdot \pi_4) \boxplus (\gamma Fd \boxdot \pi_8)}{(\gamma aL + \gamma Fd)} \ \}
\end{aligned}
$$

In the case of $\langle \ell e', \emptyset \rangle$, its state reward is $[\infty, \ldots, \infty]/\infty$, i.e. the result value is not meaningful. However, the rate ∞ implies that its activity occurs immediately, and other possible activities can be ignored. Therefore, in this example, the state reward for the state $\langle \ell e', \emptyset \rangle$ is considered as $\pi_2 \oplus \pi_5$. ◊

4.5 Interactive Markov Chains and Reo

Interactive Markov Chains (IMCs) are a compositional stochastic model [43] which can be used to provide quantitative semantics to concurrent systems. In IMCs, delays can be represented by combinations of exponential delay transitions, it allows to accommodate non-exponential distributions within the models. That is, it can represent delays from the large class of phase-type distributions [72, 70] which can approximate general continuous distributions. This enables a more general usage of Stochastic Reo Automata, if IMCs are used instead of CTMCs as the translation target of Stochastic Reo Automata models.

In this section, we discuss to what extent IMCs are an appropriate semantic model for Stochastic Reo, instead of Stochastic Reo Automata. In addition, we provide a translation from Stochastic Reo into IMCs, which enables the use of the latter as an alternative target stochastic model.

4.5.1 Interactive Markov Chains

An IMC specifies a reactive system and is formally described as a tuple $(S, Act, \rightarrow, \Rightarrow, s_0)$ where S is a finite set of states; Act is a set of actions; s_0 is an initial state in S; $\rightarrow$ and $\Rightarrow$ are two types of transition relations:

- $\rightarrow \subseteq S \times Act \times S$ for *interactive* transitions and
- $\Rightarrow \subseteq S \times \mathbb{R}^+ \times S$ for *Markovian* transitions.

Thus, an IMC is a Labeled Transition System (LTS) if $\Rightarrow = \emptyset$ and $\rightarrow \neq \emptyset$, and is a CTMC if $\Rightarrow \neq \emptyset$ and $\rightarrow = \emptyset$.

Compared to other stochastic models such as CTMCs, the main strength of IMCs is their compositionality. Thus, one can generate a complex IMC as the composition of relevant simple IMCs, which enables compositional specification of complex systems.

Definition 4.5.1. ***(Product)*** *[43] Given two IMCs* $\mathcal{I}_1 = (S_1, Act_1, \rightarrow_1, \Rightarrow_1, s_{(1,0)})$ *and* $\mathcal{I}_2 = (S_2, Act_2, \rightarrow_2, \Rightarrow_2, s_{(2,0)})$*, the composition of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *over a set of actions* A *is defined as* $\mathcal{I}_1 \times \mathcal{I}_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, \Rightarrow, s_{(1,0)} \times s_{(2,0)})$ *where* $\rightarrow$ *and* $\Rightarrow$ *are defined as:*

$$
\begin{array}{rcl}
\rightarrow & = & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2') \mid \alpha \in A,\ s_1 \xrightarrow{\alpha}_1 s_1' \ \wedge\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
 & \cup & \{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2) \mid \alpha \notin A,\ s_2 \in S_2,\ s_1 \xrightarrow{\alpha}_1 s_1'\} \\
 & \cup & \{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2') \mid \alpha \notin A,\ s_1 \in S_1,\ s_2 \xrightarrow{\alpha}_2 s_2'\} \\
\Rightarrow & = & \{(s_1, s_2) \overset{\lambda}{\Longrightarrow} (s_1', s_2) \mid s_2 \in S_2\ ,\ s_1 \overset{\lambda}{\Longrightarrow}_1 s_1'\} \\
 & \cup & \{(s_1, s_2) \overset{\lambda}{\Longrightarrow} (s_1, s_2') \mid s_1 \in S_1\ ,\ s_2 \overset{\lambda}{\Longrightarrow}_2 s_2'\}
\end{array}
$$

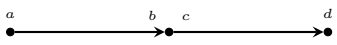
■

The product of interactive transitions is similar to the ordinary automata product, which includes interleaving and synchronized compositions of interactive transitions. The product of Markovian transitions consists of interleaved transitions only.

We now discuss IMCs from two different perspectives:

1. As a semantic model for Stochastic Reo: translating primitive Stochastic Reo channels into IMCs and then composing the derived IMCs using the product operation defined above; or
2. As an alternative translation target model: composing the Stochastic Reo Automata of primitive channels and then translating the composed Stochastic Reo Automaton into an IMC.

We show now that the first case is not adequate since it provides a wrong semantics for connectors that involve propagation of synchrony. For example, consider the following connector, denoted as 2sync, that consists of two Sync channels joined at nodes b and c.



The behavior of primitive channels consists of request-arrivals and data-flows which occur sequentially, i.e., data-flows follow request-arrivals. Both request-arrivals and data-flows are divided into two phases: an action and the random processing delay for each action. For instance, a request-arrival at node a consists of the arrival action at node a and waiting for the acceptance by node a. To reason about the end-to-end QoS, the IMCs for each Sync channel must have Markovian transitions for the random processing delays of both request-arrivals and data-flows. The two phases of channels must be considered sequentially, that is, the phase of random processing delays follows that of the action. Figure 4.10 shows the possible IMCs for the Sync channels ab and cd.

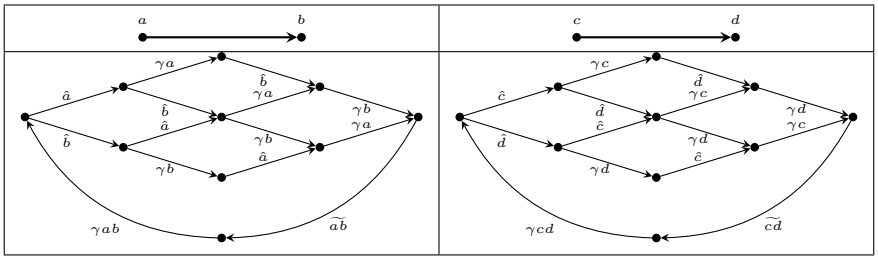


Figure 4.10: IMCs for each Sync channel

Here, we use '^' and '~' over node names in order to represent request-arrivals and data-flows, respectively. Rates for each request-arrival and each data-flow are represented with the prefix γ.

However, the composition of the IMCs for the two Sync channels does not capture the correct behavior of 2sync as specified by Reo. Figure 4.11 shows a fragment of the IMC product result where, for simplicity, the rest of the product result is omitted and represented as a cloud shape.

If we apply the assumption that the synchronization by joining nodes is an immediate action, then transitions with $(\hat{b}, \hat{c})$, γb, and γc labels are considered internal interactive transitions or discarded by certain refinements before or after the product. The result of the product and certain refinements is depicted in Figure 4.12.
Consider the diamond shape (1) in Figure 4.12, formed by the two data-flows from a to b (γab) and from c to d (γcd), which occur interleaved. In the 2sync circuit, these two data-flows occur sequentially, which means that data-flows do not occur concurrently. This example illustrates that using the concurrent composition of IMCs is not appropriate for specifying the behavior of connectors because it does not properly model the propagation of synchrony. It is natural and interesting to consider whether it is possible to adapt the composition operator of IMCs in order to delete unintended transitions (such as in the diamond shape (1) in Figure 4.12) and still retain a compositional model. However, we did not investigate this possibility since it is out of the scope of this thesis.

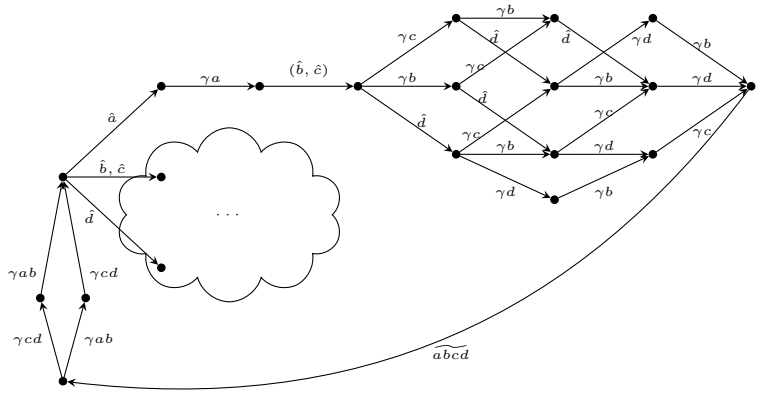


Figure 4.11: Composed IMC for 2sync

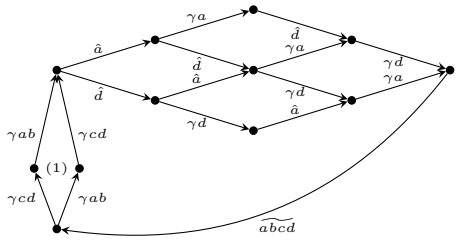


Figure 4.12: IMC after refinements on 2sync

We now show how IMCs can be used as a target stochastic model, instead of CTMCs. In this approach, the synchronization is considered in Stochastic Reo Automata, and we do not need to consider the IMC level refinements for synchronization such as the transitions with the labels $(\hat{b}, \hat{c})$, γb, and γc in Figure 4.11.

Since a Stochastic Reo Automaton does not have an initial state, the derived result is precisely an IMC transition system (IMCTS) [43], i.e., an IMC without an initial state. However, an initial state can be decided by the interpretation of the behavior of a connector. Thus, in this section, we consider the IMCTS derived from a Stochastic Reo Automaton as an IMC. An IMC derived from a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ is a tuple $(S, Act, \rightarrow, \Rightarrow)$ where $S = S_A \cup S_M$ is the set of states. S_A represents the configurations of the system derived from its Stochastic Reo Automaton and the pending status of I/O requests; S_M is the set of states that represent the occurrences of synchronized data-flows and result from the micro-step divisions of the synchronized data-flows. In general, *Act* is a set of actions of the

arrival of a data item at a boundary node and synchronized data-flows through a connector. Thus, $Act = \Sigma' \cup Frs$ where Σ' is a set of boundary nodes, and Frs is a set of firings, i.e., the counterpart of f in a label on a transition $s \xrightarrow{g|f} s' \in \delta_{\mathcal{A}}$, $f \in Frs$. The relation $\rightarrow = \zeta_{Arr} \cup \zeta_{Proc} \subseteq (S \times 2^{\Sigma'} \times S) \cup (S \times 2^{Frs} \times S)$ is a set of interactive transitions, and $\Rightarrow = \delta_{Arr} \cup \delta_{Proc} \subseteq S \times \mathbb{R}^+ \times S$ is a set of Markovian transitions. The sets indexed with Arr and $Proc$ represent transitions for request-arrivals and data-flows, respectively.

A state in $S \subseteq Q \times 2^{\Sigma'} \times 2^{\Sigma'}$ represents three kinds of configurations: configurations of a connector (Q), the occurrences of actions (first $2^{\Sigma'}$), and the presence of the I/O requests pending on its boundary nodes (second $2^{\Sigma'}$), if any. The set of S_A and the preliminary sets of request-arrival transitions are defined as:

$$
\begin{aligned}
S_A &= \{(q, A, P) \mid q \in Q,\ P \subseteq A \subseteq \Sigma'\} \\
\zeta'_{Arr} &= \{(q, A, P) \xrightarrow{\hat{c}} (q, A \cup \{c\}, P) \mid (q, A, P),\ (q, A \cup \{c\}, P) \in \Sigma',\ c \notin A\} \\
\delta'_{Arr} &= \{(q, A, P) \xRightarrow{\mathbf{r}(c)} (q, A, P \cup \{c\}) \mid (q, A, P),\ (q, A, P \cup \{c\}) \in \Sigma',\ c \notin P\}
\end{aligned}
$$

The sets ζ'_{Arr} and δ'_{Arr} are used to define below ζ_{Arr} and δ_{Arr}, respectively.

As mentioned in Section 4.4.1, synchronized data-flows are described by a single transition in a Stochastic Reo Automaton. From the interactive transition perspective, the synchronized data-flows are also described by a single interactive transition. However, from the Markovian transition perspective in a continuous time domain, a transition corresponding to multiple synchronized data-flows needs to be divided into micro-step transitions. For this purpose, we reuse the delay-sequence which is extracted by Algorithm 3.3.1. We now derive transitions for synchronized data-flows in two steps:

1. For each transition $p \xrightarrow{g|f} q \in \delta_{\mathcal{A}}$, we derive interactive and macro Markovian transitions $(p, A, P) \xrightarrow{\widetilde{f}} (p, A \setminus f, P)$ and $(p, A \setminus f, P) \xrightarrow{\lambda} (q, A \setminus f, P \setminus f)$, respectively, for every set of pending requests P that suffices to activate the guard g ($\widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}}$), where λ is the delay-sequence extracted by Algorithm 3.3.1 $\mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))$. The sets of derived transitions are defined below as ζ_{Macro} and δ_{Macro} for interactive and macro Markovian transitions, respectively.

2. We divide a transition $s \xrightarrow{\lambda} s' \in \delta_{Macro}$ into a combination of micro-step transitions, each of which corresponds to a single event.

Given a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $=(Q, \Sigma, \delta_{\mathcal{A}})$ and a set of boundary nodes Σ', a macro-step transition for synchronized data-flows is defined as:

$$
\begin{aligned}
\zeta_{Macro} = \quad & \{(p, A, P) \xrightarrow{\widetilde{f}} (p, A \setminus f, P) \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ A \subseteq P \subseteq \Sigma',\ \widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}}\} \\
\delta_{Macro} = \quad & \{(p, A, P) \xrightarrow{\lambda} (q, A, P \setminus f) \mid p \xrightarrow{g|f} q \in \delta_{\mathcal{A}},\ A \cap f = \emptyset,\ A \subset P \subseteq \Sigma', \\
& \qquad \widehat{P} \leq g \setminus \widehat{\widehat{\Sigma}},\ \lambda = \mathbf{Ext}(\mathbf{t}(p \xrightarrow{g|f} q))\}
\end{aligned}
$$

To derive an IMC from a Stochastic Reo Automaton, we reuse the function *nodes* and modify the definitions of functions *states* and *div* in Section 3.3.3. Then, $S_M =$

$state((p, A, P) \xrightarrow{\lambda} (q, A, P'))$ where

$$states((p, A, P) \xrightarrow{\lambda} (q, A, P')) = \begin{cases} \{(p, A, P), (q, A, P')\} & \text{if } \lambda = \theta \\ \bigcup states(m)\ \forall m \in div((p, A, P) \xrightarrow{\lambda} (q, A, P')) & \text{otherwise} \end{cases}$$

The function $div : \delta_{Macro} \to 2^{\delta_{Macro}}$ is defined as:

$$div((p, A, P) \xrightarrow{\lambda} (q, A, P')) = \begin{cases} \{(p, A, P) \xrightarrow{\theta} (q, A, P')\} & \text{if } \lambda = \theta \wedge \nexists (p, A, P) \xrightarrow{\theta} (p', A, P') \in \delta_{Macro} \\ div((p, A, P) \xrightarrow{\lambda_1} (p_{\lambda_1}, A, P'')) \cup div((p_{\lambda_1}, A, P'') \xrightarrow{\lambda_2} (q, A, P')) & \\ \qquad \text{if } \lambda = \lambda_1; \lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_1) & \\ \{m_1 \boxtimes m_2 \mid m_i \in div((p, A, P) \xrightarrow{\lambda_i} (p_{\lambda_i}, A, P'')),\ i \in \{1, 2\}\} & \\ \qquad \text{if } \lambda = \lambda_1 | \lambda_2 \text{ where } P'' = P \setminus nodes(\lambda_i) & \\ \emptyset \qquad \text{otherwise} & \end{cases}$$

where the function $\boxtimes : \delta_{Macro} \times \delta_{Macro} \to 2^{\delta_{Macro}}$ computes all interleaving compositions of the two transitions as follows. For a transition $(p, A, P) \xrightarrow{\lambda_1|\lambda_2} (q, A, P') \in \delta_{Macro}$, $(p, A, P) \xrightarrow{\lambda_1} (p_{\lambda_1}, A, P \setminus nodes(\lambda_1))$ and $(p, A, P) \xrightarrow{\lambda_2} (p_{\lambda_2}, A, P \setminus nodes(\lambda_2))$ correspond to, respectively, m_1 and m_2 of the third condition in the definition of the div function. While m_1 and m_2 are handled by the div function recursively, some auxiliary states, i.e., $states(m_1)$ and $states(m_2)$, are generated. In the interleaving composition, m_1 can occur at any states that are generated by $states(m_2)$, and vice-versa. This interleaving composition of m_1 and m_2 is represented as:

$$\begin{aligned} m_1 \boxtimes m_2 = \ \{\ & div\big((p_1, A, P_1) \xrightarrow{\lambda_2} (p_{(1,\lambda_2)}, A, P_1 \setminus nodes(\lambda_2))\big), \\ & div\big((p_2, A, P_2) \xrightarrow{\lambda_1} (p_{(2,\lambda_1)}, A, P_2 \setminus nodes(\lambda_1))\big) \ \mid \\ & (p_1, A, P_1) \in states(m_1) \text{ and } (p_2, A, P_2) \in states(m_2)\ \} \end{aligned}$$

This composition is similar way to the function $\bowtie$ explained in Section 4.4.2. The only difference between these two functions is the structure of their states: CTMC states are elements of $Q \times 2^{\Sigma'}$, whereas IMC states are in $Q \times 2^{\Sigma'} \times 2^{\Sigma'}$ where Σ' is a set of boundary nodes in a Stochastic Reo connector.

The division into micro-step transitions ensures that each transition has a single 3-tuple in its label. Thus, the micro-step transitions can be extracted as:

$$\begin{aligned} \delta_{Proc} = \{(p, A, P) \xRightarrow{v(\theta)} (p', A, P') \mid & \\ (p, A, P) \xrightarrow{\theta} (p', A, P') \in div(t) \text{ for all } t \in \delta_{Macro}\} & \end{aligned}$$

As mentioned above, interactive transitions in ζ_{Macro} do not need to be divided, thus, $\zeta_{Proc} = \zeta_{Macro}$

Splitting synchronized data-flows allows non-interfering events to interleave with their micro-steps, disregarding the strict sense of their atomicity. In order to allow

such interleaving, we must explicitly add such request-arrivals. For a Stochastic Reo Automaton $(\mathcal{A}, \mathbf{r}, \mathbf{t})$ with $\mathcal{A} = (\Sigma, Q, \delta_{\mathcal{A}})$ and a set of micro-step states S_M, its full sets of request-arrival transitions, including its request-arrivals, are defined as:

$$\zeta_{Arr} = \zeta'_{Arr} \cup \{(p, A, P) \xrightarrow{\hat{d}} (p, A \cup \{d\}, P) \mid (p, A, P), (p, A \cup \{d\}, P) \in S_M,\ d \in \Sigma,\ d \notin A\}$$

$$\delta_{Arr} = \delta'_{Arr} \cup \{(p, A, P) \xrightarrow{\mathbf{r}(d)} (p, A, P \cup \{d\}) \mid (p, A, P), (p, A, P \cup \{d\}) \in S_M,\ d \in \Sigma,\ d \notin P\}$$

Applying this method, Figure 4.13 shows the IMC corresponding to our 2sync example. The derived result is similar to the IMC for a Sync in Figure 4.10 and captures the correct behavior of the 2sync connector. Note that we use an abbreviated notation for states, for example, we use p, ab, a instead of $(p, \{a, b\}, \{a\})$.

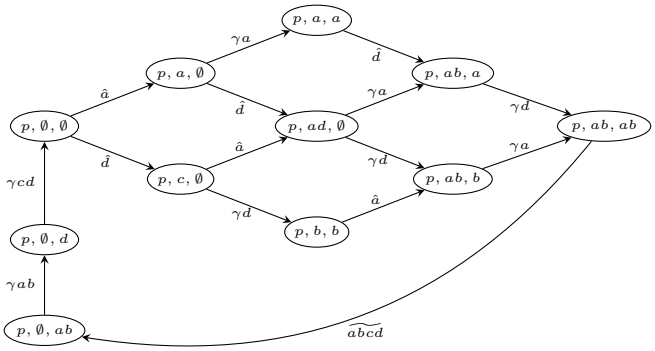


Figure 4.13: Derived IMC for 2sync

The foregoing illustrates that IMCs can serve as another alternative target model for the translation from Stochastic Reo Automata, instead of CTMCs. Although doing so does not exploit the compositionality of IMCs, translation into IMCs is still meaningful. The derived IMCs, for instance, can represent not only exponential distributions, but also non-exponential distributions, especially phase-type distributions. The analysis of IMCs is supported by tools such as the Construction and Analysis of Distributed Processes (CADP) [40]. CADP verifies the functional correctness of the specification of system behavior and also minimizes IMCs effectively [39]. Moreover, IMCs can be used in various other applications, such as Dynamic Fault Trees (DFTs) [21, 22, 23], Architectural Analysis and Design Language (AADL) [20, 25, 24], and so on [44].

4.6 Discussion

In this chapter, we introduced Stochastic Reo Automata by extending Reo Automata with functions that assign stochastic values of arrival rates and processing delay rates

to boundary nodes and channels in Stochastic Reo. This model is very compact and tractable to handle, compared to QIA. Various formal properties of Stochastic Reo Automata are obtained, reusing the formal justifications of the respective properties of Reo Automata [19], such as compositionality.

The technical core in this chapter shows the complexity of the original problem whence it stems from: derivation of stochastic models for formal analysis of end-to-end QoS properties of systems composed of services/components supplied by disparate providers, in their user environments. This complexity highlights the gross inadequacy of informal, or one-off techniques and emphasizes the importance of formal approaches and sound models that can serve as the basis for automated tools.

Stochastic Reo does not impose any restriction on the distribution of its annotated rates such as the rates for request-arrivals at channel ends or data-flows through channels. However, for the translation from Stochastic Reo into a homogeneous CTMC model, we considered only the exponential distributions for the rates. For more general usage of Stochastic Reo Automata, we also want to consider non-exponential distributions by considering phase-type distributions or using Semi-Markov Processes [50] as target models of our translation. A simulation engine [51, 89], already integrated into our toolset, the Extensible Coordination Tools (ECT) [35] environment, supports a wide variety of more general distributions for Stochastic Reo.

We discussed why IMCs are not an appropriate semantic model for Stochastic Reo, and showed the translation from Stochastic Reo into IMCs via Stochastic Reo Automata. A natural and interesting future work is to consider whether it is possible to adapt the composition operator of IMCs in order to delete unintended transitions that it currently produces in synchrony propagation scenarios, and still remain within a compositional framework.

Moreover, we have shown the extension of Stochastic Reo Automata with rewards information to specify stochastic behavior. Such an extension allows us to consider more general QoS aspects since, without rewards, Stochastic Reo Automata specify and reason about only timing information. The translation from the extended Stochastic Reo Automata into CTMCs with state reward is also shown.

So far, the translation from Stochastic Reo is carried out using QIA (instead of Stochastic Reo Automata) in our ECT environment. We are currently extending and improving these tools to use Stochastic Reo Automata, as well as the extension with reward information, so that the more compact sizes of the automata then allow us to analyze larger models.

Chapter 5

Tool implementation

5.1 Introduction

The growing complexity and importance of coordination models in software applications necessarily lead to a higher relevance of performance issues for coordinators in the development of systems. In this context, the performance of such models plays an important role in the quality of the final software system. Unfortunately, the lack of tools that support the performance analysis of coordination models makes it difficult to automatically analyze the performance properties of coordination models.

In this chapter, we introduce a tool that integrates a Stochastic Reo editor and a generator of CTMCs from Stochastic Reo models. The Stochastic Reo editor is an extension of the existing Reo editor in the Extensible Coordination Tools (ECT) [35], which is an integrated toolset for the design and the verification of (Stochastic) Reo. We have implemented the CTMC generator based on the semantics of Quantitative Intentional Automata (QIA) and their translation algorithm presented in Chapter 3. This tool, called *Reo2MC*, is provided as a plug-in for the ECT.

This chapter consists of two parts: (1) the description of the implementation of the Reo2MC tool and (2) a manual for its usage. In the first part, we explain the data structures for certain elements such as stochastic rates and delay-sequences, which were mentioned in the previous chapters.

In the second part, we explain how to use the tool: for instance, how to generate a QIA model corresponding to a Stochastic Reo connector, or translate a CTMC model from a Stochastic Reo connector or a QIA model generated from a Stochastic Reo connector. In addition, we show the link to other stochastic analysis tools, in particular PRISM [76, 57].

5.2 Reo2MC: description and implementation

Reo2MC is a plug-in for the ECT, which was introduced in [8]. The tool allows users to draw Stochastic Reo models, using an extension of the existing Reo editor

in ECT, and automatically derive the QIA semantics of Stochastic Reo models and their corresponding CTMCs.

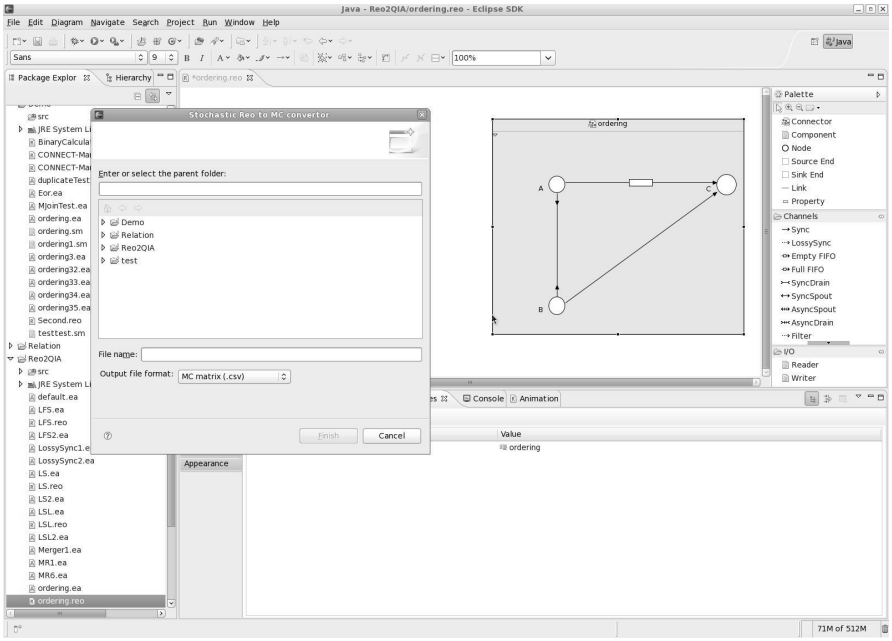


Figure 5.1: A Snapshot of Reo2MC

Reo2MC works as an Eclipse application through a graphical user interface (see Figure 5.1). The execution flow is depicted in Figure 5.2: the user provides as input a Reo circuit (which is obtained either by using the graphical editor in ECT, or by automatic synthesis from other specifications, such as UML sequence diagrams or BPMN models, the tools for which are also provided in ECT [28]) and a textual description of the stochastic constraints on the connector and its environment. Once all this input has been provided, the model can be automatically translated into QIA and CTMCs, both represented in XML. The GMF framework in Eclipse is used to generate the graphical representations of QIA and CTMCs, and the XML files of the generated CTMCs can be parsed into several other file formats, which are used as input to PRISM, MATLAB, and Maple to analyze the performance of connectors.

5.2.1 Implementation

In this section, we show the data structures and underlying functionalities used for the implementation of Reo2MC. In Reo2MC, we have implemented QIA as an operational semantic model for Stochastic Reo and as an intermediate model for the translation to CTMCs.

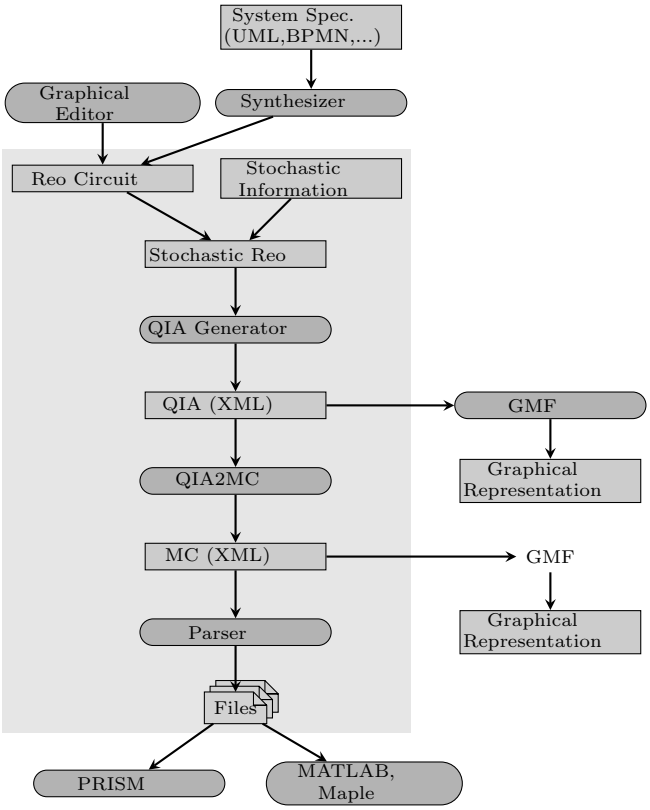


Figure 5.2: Architecture of Reo2MC

The main data structures, the implementations of which are explained in this section, consist of *structural stochastic information* for the activities involved in Reo channels and a *delay-sequence* which is the sequence of individual structural stochastic information for synchronized activities. The implementation of these data structures is based on their definitions and features as covered in Chapters 3 and 4.

The functionalities implemented in Reo2MC are also based on the definitions and the algorithms in Chapters 3 and 4. The main functionalities of Reo2MC are the product of QIA, and the extraction and the division of delay-sequences to generate their corresponding CTMCs from QIA. The following sections show the implementation details of these functionalities.

Data structures

Structural stochastic rates In Stochastic Reo, stochastic values are used to represent the arrival rates at each channel end and the processing delay rates of each

channel. The stochastic values must be non-negative real values since our target model is CTMCs, and hence are defined with type *double* in the implementation. For simplicity of modeling, we assume that each stochastic process has only one rate value.

In the existing Reo editor, a primitive Reo channel consists of two types of objects: two channel ends and an arrow between the two channel ends. A channel end is associated with one *double* value (its arrival rate), whereas the arrow in a channel is able to have more than one *double* value, according to its behavior. For instance, a Sync has one processing delay rate for a data-flow from its source to its sink nodes; a FIFO1 has two processing delay rates for data-flows from its source node to its buffer and from its buffer to its sink node; a LossySync also has two processing delay rates for a successful data-flow from its source to its sink nodes and data-loss at its source node.

Thus, we define the data type of the rates as an array of *double* values. According to the assumptions mentioned above, an arrival rate of I/O request is only one value, which we assume to be the first value in the array, i.e., the rest will be ignored. In the case of processing delay rates, the values in the array are sequentially taken as corresponding to the actions of a channel from a successful processing (e.g., a data-flow in a LossySync) to a non-successful processing (e.g., a data-loss in a LossySync) or its sequential data-flows (e.g., in a FIFO1 data-flow from its source node to the buffer and from the buffer to its sink node correspond to, respectively, the first and the second values in the array). This *double* array can be a general data structure, but this array itself is not user-friendly because it requires users to have the insight of this array, e.g., the meaning of the order of elements in the array. The implementation in the Reo editor provides some guidelines to users by presenting specific rate labels for respective Reo channels. The way of setting rates is shown in Section 5.2.2.

These values are propagated to and reused in a QIA model, corresponding to a given Stochastic Reo connector, as the elements in the labels on QIA transitions. The mapping from Stochastic Reo to QIA is carried out as follows. First, primitive channels comprising a connector are mapped to their corresponding QIA models. Since mapping the primitive channels to their corresponding QIA is a one-to-one mapping, this is quite straightforward. Then the product of the corresponding QIA models generates the QIA model for the connector. Thus, the actual pairing of a rate with its relevant activity takes place during the mapping of primitive channels to their corresponding QIA.

This pairing is decided according to the node names relevant to each stochastic process, as the rates are named after their respective node names in Stochastic Reo (this naming is explained on page 11). In addition, for the translation to CTMC, the rates need to be delineated according to the connection information of the Stochastic Reo connector. For this purpose, we propose structural stochastic information that describes the stochastic rates with explicit mention of their relevant source (input) and sink (output) nodes. Such structural stochastic information is defined as a 3-tuple and represented as an element in the labels of QIA transitions. Such a tuple, denoted by a `DelayElement`, has been implemented as:

```
public class DelayElement extends EObjectImpl implements EObject {
  protected EList<String> input;
  protected EList<String> output;
  protected double delay = DELAY_EDEFAULT;
  ...
}
```

The attributes in this class are `input` (source nodes), `output` (sink nodes), and `delay` (a rate).

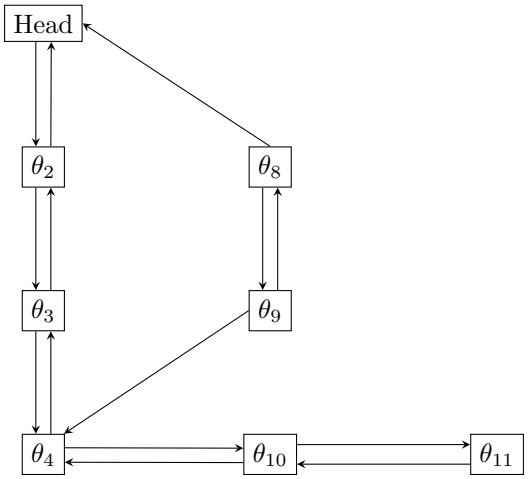
Delay-sequences In the translation from QIA to CTMC models, a delay-sequence, defined in Section 3.3.1, is generated for each transition of synchronized data-flows. Each data-flow has a 3-tuple $\theta = (I, O, r)$ that depicts its connection information reflecting the topology of a Reo connector, which is implemented by `DelayElement`.

A delay-sequence is composed by the operators | and ; for, respectively, parallel and sequential compositions. A delay-sequence composed by | describes that the data-flows corresponding to each element in the delay-sequence occur interleaved. A delay-sequence composed by ; describes that the data-flows occur sequentially, from the leftmost element to the rightmost one.

In order to represent such delay-sequences, we define our own linked-list and its elements, called `DElmNode`. The content of the elements of this linked-list is the tuple θ, i.e., the data type E below is `DelayElement`. Each `DElmNode` has four pointers of `prev`, `next`, `right`, and `left` as shown below.

```
public class DElmNode<E> {
   public DElmNode<E> prev = null;
   public DElmNode<E> next = null;
   public DElmNode<E> left = null;
   public DElmode<E> right = null;
   public E data = null;
   ...
}
```

We use the above pointers to generate a linked-list, representing a delay-sequence. The following example shows the concrete idea of the structure of our linked-list. Recall the delay-sequence $\lambda_1 = ((\theta_2; \theta_3)|(\theta_8; \theta_9)); (\theta_4|\theta_{10}|\theta_{11})$ in Example 3.3.1. The linked-list representation of this delay-sequence is described as:



where we have omitted redundant pointers that indicate *null*.

In the implementation of our linked-list (see below), there are two different ways to insert nodes: horizontal and vertical insertion. At a current node, horizontal inserting is used to insert a node that is composed with the current node in parallel; vertical inserting is used to insert a node that is composed with the current node sequentially. In implementation, horizontal and vertical insertings are described by the functions `parallelInsertion` and `sequentialInsertion`, respectively.

```
public class NewLinkedList{
   DElmNode<DelayElement> head = null; //The head of the list
   DElmNode<DelayElement> tail = null; //The tail of the list
   DElmNode<DelayElement> current = null; //Last modified node
                                          //of the list
   ...
   public void sequentialInsertion(DelayElement f){
      if(this.isEmpty())    addDatatoTail(f);
      else if(this.current.next==null){
         DElmNode<DelayElement> temp = new DElmNode<DelayElement>();
         temp.prev = current;
         temp.next = null;
         temp.left = null;
         temp.right = null;
         temp.data = f;
         current.next = temp;
         current = temp;
         tail = temp;
     }
   }
```

```
   public void parallelInsertion(DelayElement b){
      if(!this.isEmpty() && current!=null){
         DElmNode<DelayElement> temp = new DElmNode<DelayElement>();
         temp.prev = current.prev;
         temp.next = current.next;
         temp.left = current.getLastright();
         temp.left.right = temp;
         temp.right = null;
         temp.data = b;
      }
   }
   ...
}
```

Algorithms

Next, we explain the implementation of the algorithms for the mapping between stochastic Reo and QIA, the product operation of QIA, and the translating of a QIA to a CTMC.

Obtaining QIA for Stochastic Reo We obtain the QIA model corresponding to a Reo connector in two steps:

1. map primitive channels, which constitute the connector, to their corresponding QIA models.
2. compose the QIA models obtained from Step 1.

We assume that the types of primitive Reo channels are fixed (Sync, LossySync, SyncDrain, FIFO1, and so on). Thus, we use a template that provides a one-to-one mapping for each primitive channel and its QIA models.

Each QIA model in the template uses temporary names for its nodes such as `SOURCE0` and `SINK0`. These temporary names in a QIA model are renamed according to the node names of the primitive channel corresponding to the QIA model. When a node is shared by more than two channels, e.g., a replicator or a merger, indices are used in the renaming procedure to explicitly describe the processing delay rates of the channels. The following code shows the implementation of this renaming where `sourceName` is a name given by users and `SOURCE` is a default name in the QIA template:

```
int i = 0;
for (PrimitiveEnd sourceEnd: channel.getSourceEnds()){
  String sourceName = endNames.getName(sourceEnd);
  QIARefactoring.renamePortName(SOURCE+ i++, sourceName, copy);
  ...
}
```

`SOURCE` is replaced with `sourceName`, and the index `i` is attached to the replaced name.

Like temporary node names in the template, the values of processing delay rates also use null temporary values. During the mapping, these values will be updated according to the user's input. However, providing values is optional at this stage and can be postponed until the CTMC model corresponding to a connector is generated. Next, we describe how to compose QIA models, by means of the product operation.

QIA product According to the QIA product in Definition 3.2.2, the resulting transition of the product of a pair of transitions fires them together, in case they agree on the common nodes of the two automata, or independently, otherwise. In the case of two synchronized firing transitions, the product result is implemented as joining all the elements of the two transitions. For instance, given two transitions $p_1 \xrightarrow{A|B,\Theta_1} p_2$ and $q_1 \xrightarrow{C|D,\Theta_2} q_2$, the composition result is $\langle p_1, q_1\rangle \xrightarrow{A\cup C|B\cup D,\Theta_1\cup\Theta_2} \langle p_2, q_2\rangle$. For the product of interleaved transitions, a different implementation is required, but for code reusability, we decided to reuse this joining method. For this purpose, a null transition $q \xrightarrow{\emptyset|\emptyset,\emptyset} q$ is added to all states in the two automata as pre-processing for the product. Then, for any interleaved transition $p_1 \xrightarrow{A|B,\Theta_1} p_2$, its composition result is $\langle p_1, q\rangle \xrightarrow{A|B,\Theta_1} \langle p_2, q\rangle$.

Extracting delay-sequences The extraction of a delay-sequence is implemented based on Algorithm 3.3.1. According to this algorithm, each independent sub-delay-sequence λ_θ is generated and then the sub-delay-sequences are composed in parallel to generate the whole delay-sequence S. Each initial 3-tuple itself becomes a starting point for an independent sub-delay-sequence. As a newly appended 3-tuple, each initial one is used to choose adjacent 3-tuples, which are appended to the end of the relevant sub-delay-sequences until no more adjacent 3-tuples exist. In the following implementation, `Post` is used to denote the set of adjacent 3-tuples and it is appended to the linked-list mentioned in Section 5.2.1, which is the data structure for a delay-sequence. This function is a direct implementation of Algorithm 3.3.1.

```
List<DelayElement> Init = getEdgeInput(DI);
List<NewLinkedList> dlist = new Vector<NewLinkedList>();

for(DelayElement a : Init){
  NewLinkedList temp = new NewLinkedList();
  temp.sequentialInsertion(a);
  dlist.add(temp);

  List<DelayElement> Pre = new Vector<DelayElement>();
  List<DelayElement> Post = new Vector<DelayElement>();
  Pre.add(a);
```

```
  Post = getNext(DI,Pre);
  while(!Post.isEmpty()){
    for(DelayElement b : Post){
      temp.contains_removes(b);
    }
    temp.sequentialInsertion(Post.get(0));
    for(int i=1;i<Post.size();i++){
      temp.parallelInsertion(Post.get(i));
    }
    Pre.clear();
    Pre.addAll(Post);
    Post = getNext(DI,Pre);
  }
}
```

Deriving CTMC We discuss how to divide a macro-step transition with a delay-sequence into a number of micro-step transitions. This is implemented based on the function *div*, which is explained in Section 3.3.3. The first and the fourth conditions in the *div* function are trivial, thus, here, we consider the second and the third conditions only.

In the implementation, an element `Current` (see below) in a linked-list, which points to the current position of the list, traverses the linked-list corresponding to a delay-sequence. We extract the necessary information to generate a CTMC model from the structural properties of the nodes in this list. For example, when the `Current` element is a *head* node of a list, it corresponds an initial state of a CTMC model to be generated. When the `Current` element is an actual element, this element must be handled by adding a new transition with a new target state to the CTMC state that is generated by its `prev` element. Moreover, when the pointer `right` of the `Current` element indicates an actual element, the current element and its `right` element must be interleaved.

We now consider the second and third conditions in the *div* function. The second condition in *div* implies a sequentially composed delay-sequence. In this case, `right` of the `Current` element is *null*. While traversing a linked-list, e.g., `Current = Current.next`, it generates a linear state diagram corresponding to the sequentially composed delay-sequence represented by the list.

```
while(Current!=null){
  ...
  else if(Current.right==null){
    State source = preTarget;
    State target = new State();
    Transition t = new Transition();
    ...
    t.setSource(source);
```

```
    t.setTarget(target);
    Current = Current.next;
  }
  ...
}
```

The third condition in *div* implies a delay-sequence composed in parallel. Then `right` of the `Current` element must not be *null*. We divide this into two different cases according to their possible structure: first a number of delay-sequences are composed in parallel; second a number of 3-tuples are composed in parallel. These two cases are denoted by, respectively, `parallelList` and `parallelNode` in the implementation, and distinguished by the types of the `Current` element, i.e., a *head* or a normal node. For example, if the `right` of the `Current` element is a *head* node, then it implies that a delay-sequence is composed in parallel since the instance of each delay-sequence is a linked-list and every linked-list starts with a *head* node. Thus, the `parallelList` case is considered to generated a CTMC model.

```
if(Current.right!=null){
  DElmNode<DelayElement> horizontal = Current;
  parallelList = false;
  parallelNode = false;
  while(horizontal!=null){
    if(horizontal.isHead()){
      horizontal.setVisited(true);
      store.push(horizontal.next);
      parallelList = true;
    }
    else{
      store.push(horizontal);
      parallelNode = true;
    }
    horizontal = horizontal.right;
  }
  ...
}
```

The elements connected to the `Current` element in the linked-list by the pointers `left` and `right` are stored in the stack `store` in the implementation (see below). The stored elements are used to generate the corresponding CTMC fragment according to whether their structure identifies them as *parallelList* or *parallelNode*.

The CTMC fragment for *parallelList* is built as follows. Let L be a linked-list and $l_1, \ldots, l_n$ be the linked-lists that are composed in parallel to constitute L. We first generate the CTMC fragments corresponding to $l_1, \ldots, l_n$, in a recursive procedure, since each l_i is also a linked-list. After that, the CTMC fragments of all l_i are interleaved. Thus, the CTMC fragment for the whole linked-list L guarantees the independence of $l_1, \ldots, l_n$ while retaining the precedence order of the elements in each l_i.

```
while(!store.isEmpty()){
  if(parallelList){
    DElmNode<DelayElement> tempNode = store.pop();
    DelayElement temp = tempNode.getData();
    ...
    Automaton tempAutomaton1 = new Automaton();
    NewLinkedList sub1 = list.subSeq4(temp);
    costA =
     product(costA, addNewParts(tempAutomaton1, transition, sub1));
  }
  ...
}
```

For a `parallelNode`, each element in the stack `store` corresponds to an automaton with two states and one transition, which is represented as `tempAutomaton2` below. The automata generated for these elements are also interleaved in the composition.

```
else if(parallelNode){
  while(!store.isEmpty()){
    DElmNode<DelayElement> tempElement2 = store.pop();
    tempElement2.setVisited(true);

    // Make an automaton for each parallel delay
    Transition tempTransition2 = new Transition();
    State tempSource2 = new State();
    State tempTarget2 = new State();
    tempTransition2.setAutomaton(tempAutomaton2);
    tempTransition2.setSource(tempSource2);
    tempTransition2.setTarget(tempTarget2);
    ...
    // Get an automaton of whole parallel delays
    costA = product(costA, tempAutomaton2);
  }
}
```

5.2.2 Usage

In this section, we explain the usage of Reo2MC. Reo2MC is a plug-in for the ECT, which generates the semantic model, QIA, of a Stochastic Reo circuit and translates it into its corresponding stochastic model, CTMCs. Depending on the type of analysis to be performed on a Stochastic Reo circuit, users can choose the target model of the translation as QIA or CTMCs.

The Stochastic Reo circuit input to Reo2MC provides auxiliary information such as explicit node names and stochastic rates for request-arrivals and data-flows. Node names are used to denominate rates that are used for the analysis. Thus, we need to

name only the nodes relevant to our stochastic processes of interest, instead of all the nodes in a circuit. The rate values do not necessarily have to be set in the drawing phase of the Reo circuits; their assignments can be postponed.

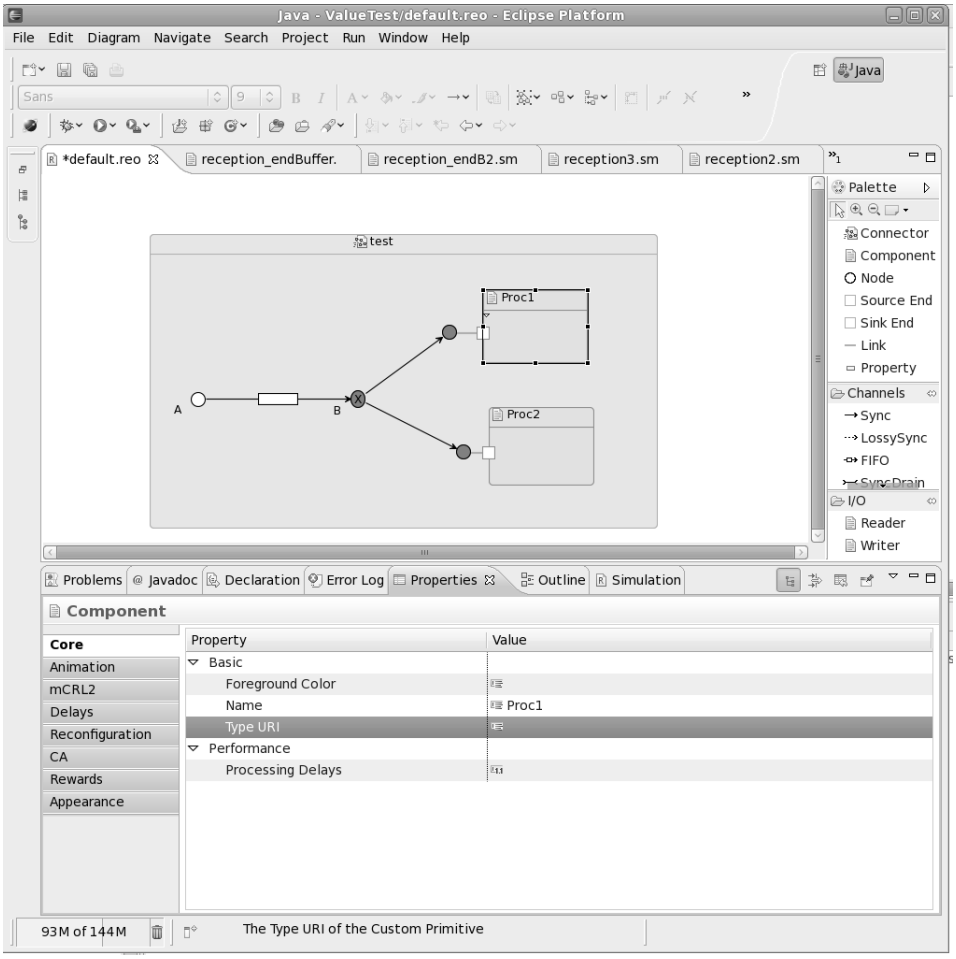


Figure 5.3: Hierarchical modeling in Reo editor

Reo editor

In the Reo editor[1], components and connectors between the components can be specified. A component in the Reo editor is described as a black box with source (input)

[1]The graphical user interface of Reo2MC, including Reo editor and the basic template for automata, have been implemented by Christian Krause. Details can be focused in his thesis [55].

and sink (output) ends to be connected to connectors. Such components function as the environment for connectors. Drawing a component is done intuitively as follows:

- select `Component` in the `Palette` at the left of the Reo editor (see Figure 5.3 which is a screen shot of the Reo editor.)
- click any spot at the editing canvas

As mentioned before, connectors are specified by composing basic Reo channels. The types of the basic channels are finite such as Sync, LossySync, and FIFO1. The Reo editor provides a template of these basic channels in the `Channels` section below the `Palette` section. Drawing connectors is very similar to the way of drawing components. That is, first draw a `Connector` at the editing canvas instead of a `Component`, then nodes and channels can be drawn inside the `Connector`. For instance, in Figure 5.3, a FIFO1 channel and two Sync channels are drawn inside the `Connector test`.

In addition, a component can be used hierarchically as a sub-component in connectors or other components. This provides better visualization and readability for modeling, and moreover, it guarantees reusability. See the model in Figure 5.3. The `Components` `Proc1` and `Proc2` belong to `Connector test`. The specification of these `Components` can be provided by external files that must include QIA or CA models corresponding to their behavior. The location of these external files are set in the `TypeURL` entry of the `Basic` field in the `Core` section in the `Properties` tab at the

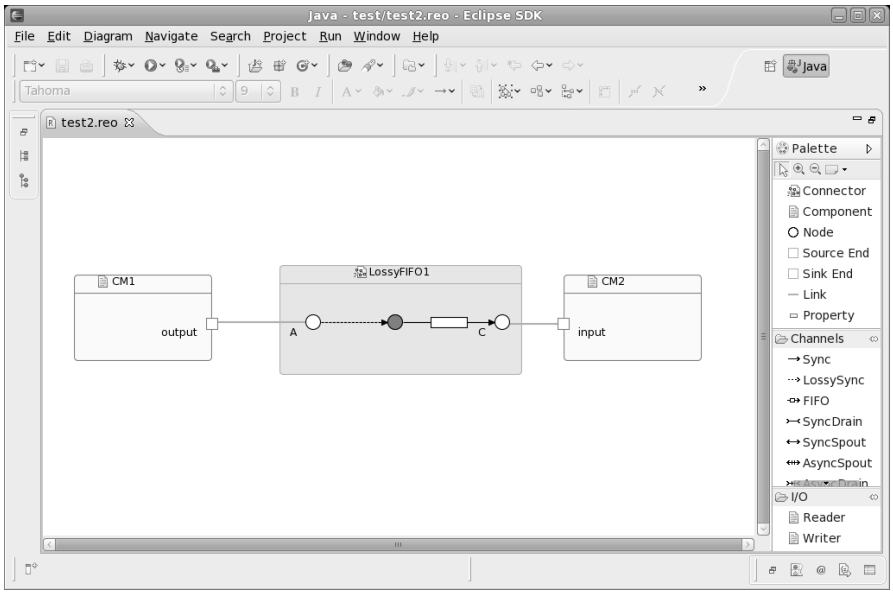


Figure 5.4: Connector between components

bottom of the Reo editor. In Figure 5.3, this `TypeURL` entry is selected for setting the location of external files. Such usage of external files for modeling enables one to reuse existing automata models for other specifications.

As the environment of connectors, components interact with their adjacent connectors through the boundary nodes of the connectors. Figure 5.4 shows a reader (CM2) and a writer (CM1) components and a LossyFIFO1 circuit that connects them in the Reo editor. Note that in Reo2MC, node names are represented in upper case. Even though one can input node names in lower case, Reo2MC changes them into upper case. Thus, here and in the remainder of this chapter, examples of Reo circuits have node names in upper case.

As mentioned before, a Stochastic Reo connector has two different kinds of stochastic values: arrival rates and processing delay rates. For setting these values, users first have to choose a node or a primitive channel in the editor by clicking on it. Then

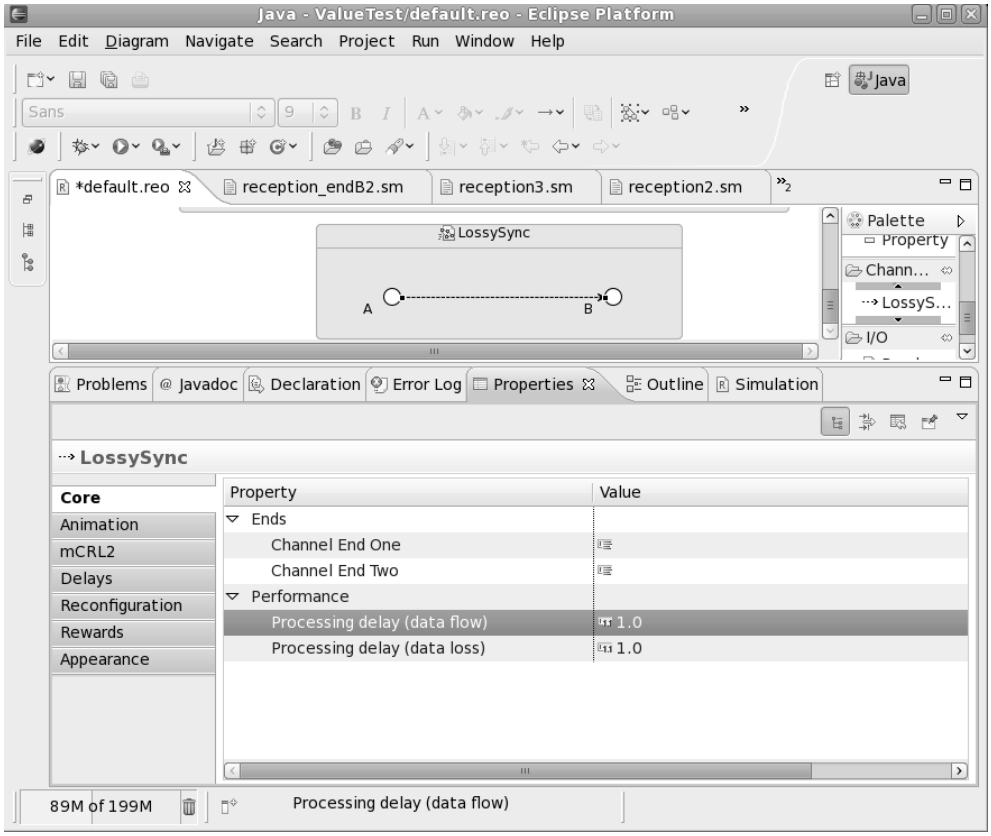


Figure 5.5: Setting processing delay rates for LossySync

the rates can be set in the `Performance` field of the `Core` section in the `Properties` tab at the bottom of the editor. As an example, Figure 5.5 shows how to set the processing delay rates of a LossySync circuit. According to the types of rates, i.e., arrival rates and processing delay rates, the `Performance` field shows relevant labels. In Figure 5.5, a channel is selected, thus, the `Performance` field shows the label `Processing delay`, otherwise, `Arrival rate` would be presented. Moreover, if a channel has more than one activity, e.g., a LossySync or a FIFO1, the `Performance` field presents all possible types of processing delay rates explicitly. For instance, for a LossySync channel, `Processing delay (data flow)` and `Processing delay (data loss)` in Figure 5.5.

Setting values for the rates is optional. For fixed rate values, users can set the values in this specification phase. However, if users want to analyze how the system's behavior changes by tweaking the rates, then the rates can be left without setting their values to be decided after the translation into a CTMC model.

In order to set and manipulate the rate values in a derived stochastic model, users need to fix the node name before the translation procedure. Otherwise, Reo2MC will decide node names automatically and this makes it difficult to figure out which name

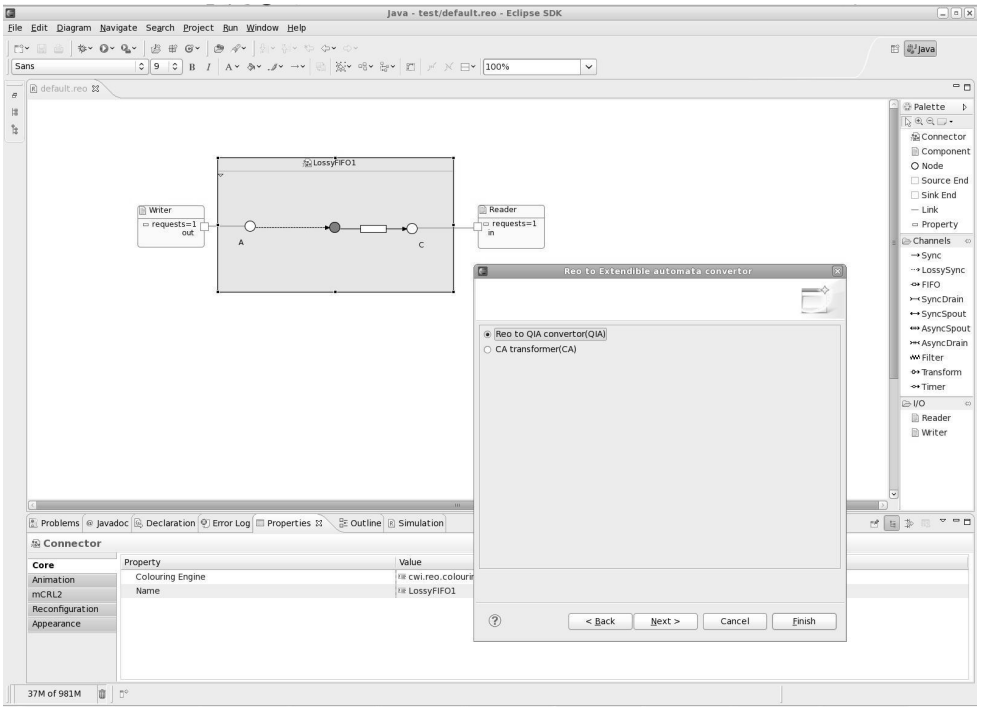


Figure 5.6: Translation of a Reo circuit into its QIA

corresponds to which node. Thus, if some nodes are not important (like mixed nodes), then the user does not need to give them names. For example, we can skip naming the mixed node of a LossyFIFO1 circuit, shown in gray in Figure 5.4.

Generating QIA for Stochastic Reo

CA and QIA are operational semantic models for Reo and Stochastic Reo, respectively. Reo2MC is a part of the ECT toolset, and the ECT supports converting a Reo circuit to a CA and a Stochastic Reo circuit to a QIA. As mentioned above, Stochastic Reo is an extension of Reo with the annotation of rates. In addition to that, setting rates can be postponed by setting all the rates with the default value 0.0. That is, any Reo circuit can be considered as a Stochastic Reo circuit. Users decide which semantic model will be the target of the translation. For converting a Reo circuit to its semantic model, "*Convert to Extensible Automaton*" must be chosen in the pop-up menu (which can be invoked using the mouse right-click on the circuit) of the Reo circuit; then the user can decide the file name for the conversion result and which semantic model to use as the target model for the conversion. Figure 5.6 shows the step of selecting the target semantic model.

Translation from QIA to CTMC

From a generated QIA model, we can obtain its corresponding CTMC model. For this translation, "*Translate to MC diagram*" must be chosen in the pop-up menu of a QIA model. The result of this translation is a state diagram of a CTMC model. Figure 5.7 shows the CTMC model derived from the QIA for the LossyFIFO1 circuit above.

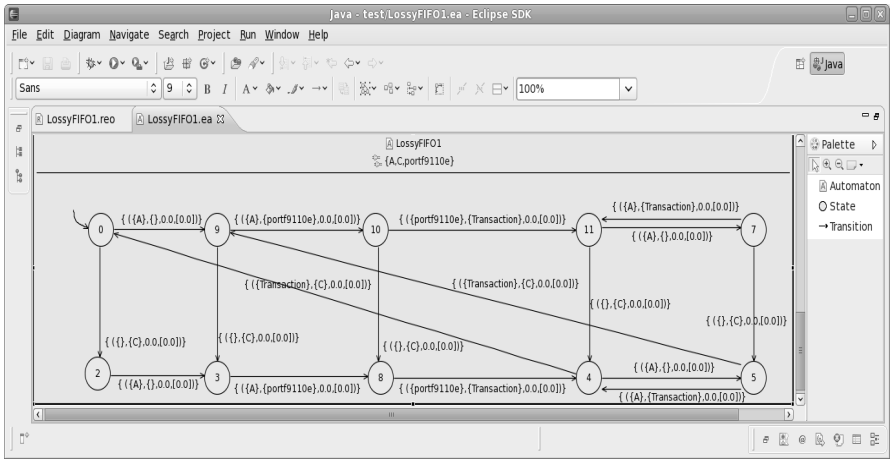


Figure 5.7: Generated CTMC model for LossyFIFO1 in Figure 5.4

For stochastic analysis, this result is fed into other analysis tools such as PRISM,

MATLAB, and Maple. For this purpose, Reo2MC supports generating the textual file describing the derived CTMC. For this generation, "*Generate MC textual file*" must be chosen in the pop-up menu of the derived CTMC model. In the window (see Figure 5.8), a result file name and its file format are specified in, respectively, *File name* and *Output file format* entries. Reo2MC supports the "*sm*" file format for PRISM and the "*csv*" file format for MATLAB and Maple.

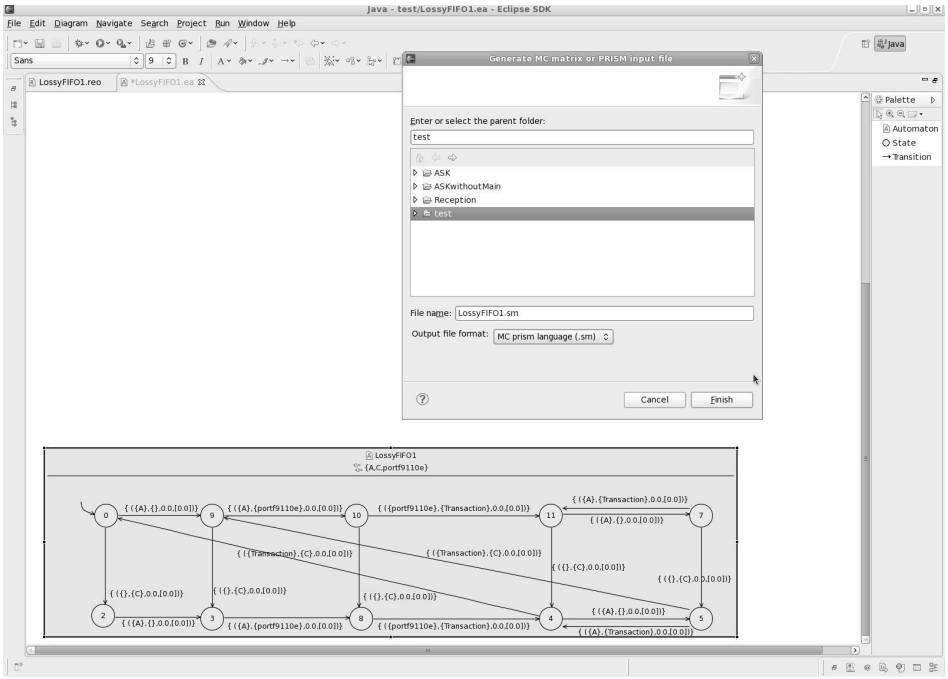


Figure 5.8: Generation of a textual file for the derived CTMC

Translation from Reo to CTMC

In general, the state space of MC grows drastically fast which causes the state-explosion problem. As shown in Figure 5.7, the result of the translation into CTMC models via QIA describes the whole state diagram. In the case of large systems, it may not be feasible to generate the state diagram of the CTMC for the whole Stochastic Reo circuit since, in general, it takes long to draw and deploy the whole state diagram. Moreover, the large graphical result of the translation is neither tractable nor readable. Thus, Reo2MC also provides the translation from Stochastic Reo circuits into the textual representation of the derived CTMCs without showing the whole CTMC diagrams. To skip drawing the state diagram of the derived CTMC, "*Convert Reo to Markov Chain - no diagram*" must be chosen in the pop-up menu of a Reo circuit.

Figure 5.9 shows the translation from the LossyFIFO1 circuit into its textual CTMC model for PRISM with the "sm" file extension.

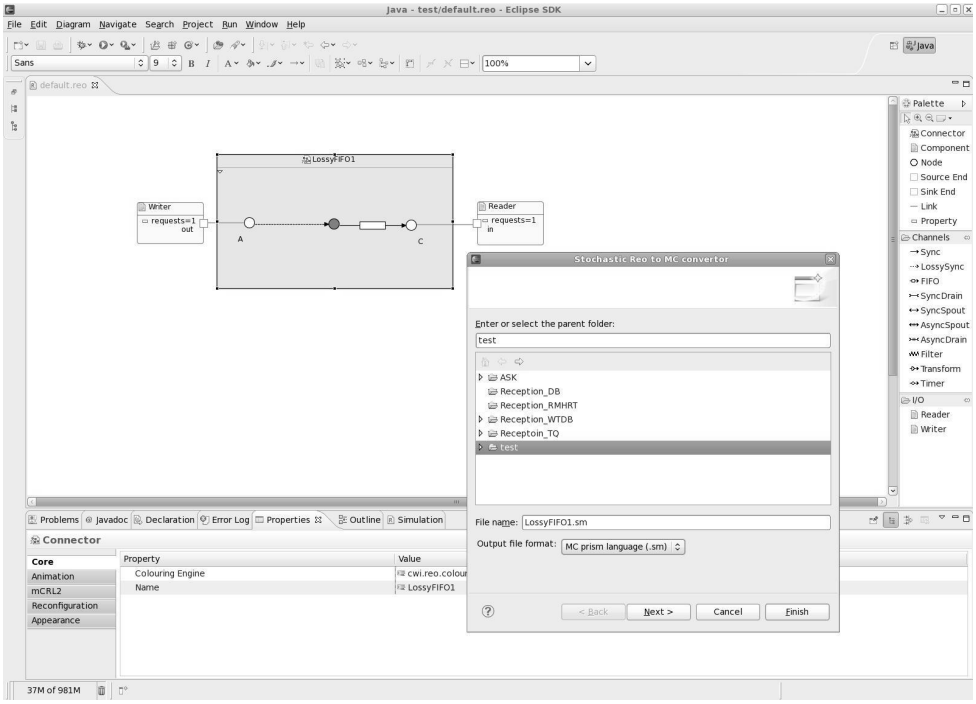


Figure 5.9: Translation of LossyFIFO1 into a CTMC model

Link to existing stochastic analysis tools The files generated by Reo2MC can then be fed to other tools. Figure 5.10 shows a file opened in PRISM, generated from the LossyFIFO1 circuit in Figure 5.9. As an example of stochastic analysis in PRISM, Figure 5.11 shows the graph of the variation of the probability of data-loss at node A in the LossyFIFO1 circuit when other rates of data-flow through the connector and I/O request arrivals at the other boundary node C are all set as 1. The graph shows as the frequency of I/O request arrivals at node A increases, the data-loss increases since node A is blocked more frequently.

5.3 Discussion

In this chapter, we introduced the Reo2MC tool in the ECT which is an integrated toolset for the design and the verification of Stochastic Reo. As a plug-in for the ECT, it provides the following functionalities: 1) editor for Stochastic Reo 2) generating

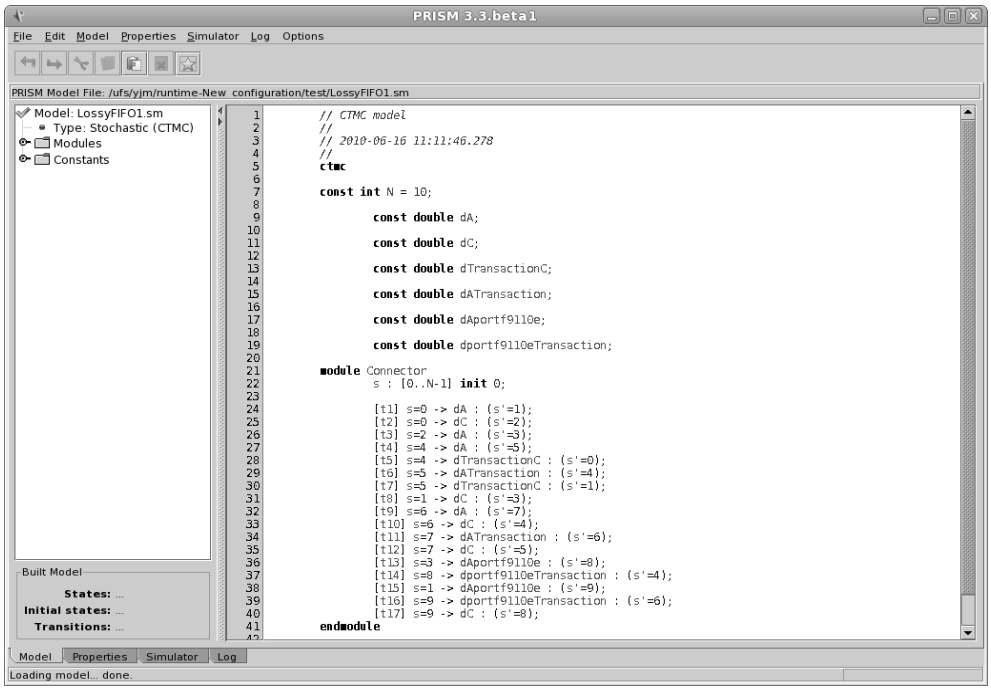


Figure 5.10: Using the PRISM with the generated file for LossyFIFO1

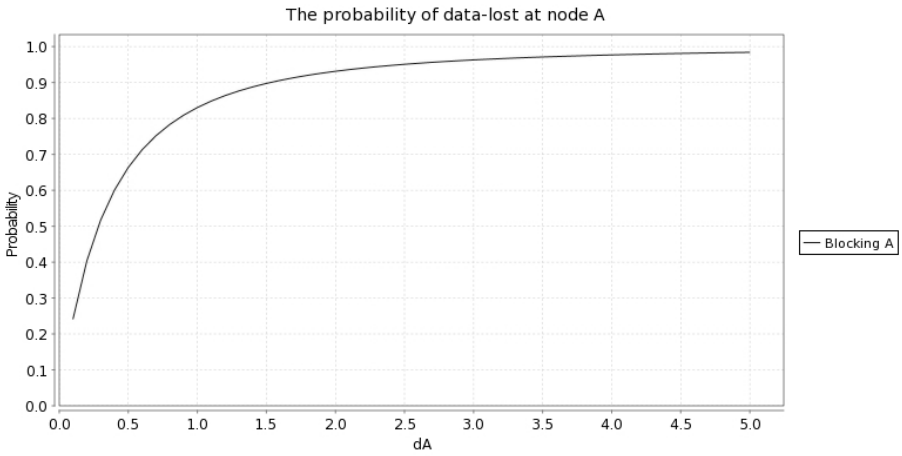


Figure 5.11: Example of using the PRISM

semantic models for Stochastic Reo, in particular, QIA, 3) deriving a CTMC model for a Stochastic Reo circuit.

Moreover, in this chapter, we also explained some implementation details of the Reo2MC tool. We showed the data structures of certain elements such as stochastic rates and delay-sequences: stochastic rates are used in Stochastic Reo and QIA; delay-sequences are used as an intermediate object for the translation from Stochastic Reo to a CTMC model.

Stochastic rates are annotated as properties of stochastic processes on a Reo connector modeling request-arrivals at channel ends and data-flows between channels. These constitute a delay-sequence for the translation from Stochastic Reo to its corresponding CTMC. A delay-sequence conveys the local topology of a connector, which is used to delineate synchronized data-flows. Thus, the data structure of Stochastic Reo contains the information about the topology of a connector.

The implementation is based on the definitions and the algorithms that we explained in Chapters 3 and 4.

Chapter 6

Case study

6.1 Introduction

In this chapter we show how the theory developed in previous chapters, implemented as tools, can be used to model a part of a real industrial system, perform QoS analysis, and help the developers get an insight into the system behavior, which enables to improve the performance of the system. We model and analyze the ASK system, a software system developed by the Dutch company Almende, which provides efficient matching between service providers and clients. An example of the application of the ASK system consists of a service-based system running in a call center that matches calling clients with the appropriate representatives that can provide them with the specialized customer service that they need.

One challenge that arises when installing particular instances of the ASK system is how to allocate resources, which are typically scarce or expensive. For instance, in the particular example above, the call center wants to have an optimal distribution of its operators' schedules in order to reduce waiting time for the customers without increasing enormously its personnel costs. A stochastic model of the ASK system can be used to perform analysis and provide advice to solve such problems.

The main contributions of this chapter are the following:

1. a stochastic Reo model of the ASK system[1]. The distributions in this model were obtained by statistical analysis of real values filtered out of the logs of an actual running ASK system.

2. analysis of several interesting properties using the probabilistic model checker PRISM [57, 76] which allowed to produce suggestions for the performance improvement of the ASK system. This analysis is done on a CTMC obtained from the Reo model.

[1]Details available at `http://reo.project.cwi.nl/cgi-bin/trac.cgi/reo/wiki/CaseStudies/SimulatoronASK/Reception`.

3. analysis of the system using a simulator which enables the study of properties involving non-exponential distributions (CTMCs can deal only with exponential distributions).

6.2 The ASK system

The "Access Society's Knowledge" (ASK) system [83] is an industrial software developed by the Dutch company Almende [1], and marketed by their daughter company ASK Community Systems [10]. The ASK system is a communication software product that acts as a mediator between service consumers and service providers, for instance, connecting rescue institutions (e.g., fire departments) and professional volunteers. The connection established by the ASK system is provided by mechanisms for matching users requiring information or services with potential suppliers. For this purpose, the matching mechanisms use the profiles and availability offered by people who provide or require services.

The main goal of the ASK system is to do the matching in an efficient way. To achieve that, the system collects feedback on the quality of services after the connection. Such feedback is used to decide better connections for the subsequent requests of the same type. In addition, the system uses self-learning and self-organizing mechanisms by continuously updating to users' preferences and available resources. Moreover, the ASK system enables users to inform others about their status, their availability, and how they can be contacted best. This information is used to select the right people for a communication session as well as the feedback.

To offer efficient connections, the ASK system considers the following aspects:

- human knowledge and skills of service providers
- time schedules of the provision of services
- communication media such as telephones, SMS, and emails

When people request a certain service from specialists or service providers, the ASK system attempts to select the best possible service provider. This selection is based on the rating of the knowledge and the skills of service providers who are available at that moment. This rating, in turn, is based on the feedback on the quality of services offered by the service providers.

The occurrences of events can follow either regular schedules or ad-hoc schedules. The ASK system deals with both of these situations while satisfying the constraints and the purposes of users' requests.

The ASK system generally considers the telephone as a primary communication medium, but other means of communication, such as email or SMS, are also supported. These types of media must be considered according to the reachability and the preferences of the users requests. For example, people can have more than one email address and telephone number, with different associated usage constraints and

user preferences. Such information must be indicated in the system to allow for efficient connections.

Some representative applications of the ASK system include:

- *Workforce deployment.* To offer deployment of temporary workers by finding those who are available for an assignment. For instance, this can the setup in an employment agency.

- *Customer services.* To directly connect customers to proper, available service providers, according to the customers' requirements. For instance, the Dutch housing corporation Vestia increases its tenant satisfaction by using the ASK system to put its tenants in direct contact with a repairman in case some house facilities need to be repaired.

- *Emergency response.* To collect the status information of emergent situations and provide safety by utilizing all possible means of communications. For instance, this can be used to find available volunteers with the best accessibility for emergency situations.

- *Flexible resource allocation.* To increase the flexibility of workforce and to decrease scheduling workload. For instance, this can be used to provide the best matches between working schedules and private lives of employees. In fact, the European mail distribution company TNT Post uses the ASK system for this type of flexible resource allocation.

- *Knowledge sharing.* To collect, share, and distribute information, experiences, skills, and preferences of users in order to provide high quality of service. For instance, in patient care applications that involve multiple care giver professionals or institutions, such as individuals, hospitals, and/or pharmaceutical companies, the ASK system can be used to update and share patient information to provide proper services in synchronization.

The ASK system acts as an agent that connects service providers and service consumers in an *efficient* way, handling multitudes of such connections simultaneously at any given time. The ASK system has a hierarchical modular architecture, i.e., it consists of a number of high-level components, which in turn consist of lower-level components, etc., running as threads. In order to handle massive numbers of connections concurrently, the components need to utilize multiple threads that provide the same functionalities. In this setting, allocation of system resources, e.g. the number of threads, to various components plays a critical role in the performance and responsiveness of an installed system in its actual deployment environment (e.g., properties of servers, available telephone lines, call traffic, available human operators, etc.), but determining the proper resource allocations to provide good performance is far from trivial. Deriving and analyzing a stochastic model for an installed ASK system provides valuable input and insight for improving its performance. Among other possibilities, such a model allows system architects and installation operators to play what-if

games by changing various resource and demand parameters and discover how a deployed system would perform under such scenarios, in order to adjust and fine-tune the system for cost-effective, optimal performance.

Various methods for performance evaluation have been suggested. Rigorous methods require mathematical models of a system involving variables that represent the parameters relevant to its behavior. Stochastic models describe random system behavior, leading to more realistic models of behavior than their deterministic counterparts. CTMCs, one of stochastic models, are frequently used to model randomized behavior in various systems and their features, and efficient closed-form and numerical techniques [85] exist for analysis. Traditionally, such models are constructed by human experts whose experience and insight constitute the only link between an actual system and the resulting models.

Ideally, mathematical models for the analysis of the behavior of a system should be derived from the same (hopefully, verified correct) models used for its design and construction. Such automation makes the derivation of these models less error-prone, and ensures that a derived analytical model corresponds to its respective implemented system. An expressive modeling formalism that simultaneously reflects structural, functional, and QoS properties of a modeled system constitutes a prerequisite for this automation. Reo serves as an example of such a formalism: (1) it provides structural model elements whose composition reflects the composition of their counterpart system components with architectural fidelity; (2) it allows formal verification of functional and behavioral properties of a modeled system; (3) it supports derivation of executable code form its models; and (4) it supports derivation of mathematical models for the analysis of the QoS properties of systems.

A Reo model of the ASK System was developed as a case study [34] within the context of the EU project Credo [33] for verification of its functional properties. In the work we report in this chapter, we refined and augmented this Reo model with stochastic delays extracted from actual system logs to derive a Stochastic Reo model for the ASK System. Together with the Almende company, we use this model to analyze and study the QoS properties of the ASK system in various settings. For instance, using the approach in [68], we derive CTMC models from the Stochastic Reo model of the interesting parts of the ASK System, and feed them into CTMC analysis tools, which enables us to do model checking of the stochastic behavior of the system. We will show the analysis of several such properties using PRISM in Section 6.4.1. The following sections describe the architecture of the ASK system in some detail. The figures and the descriptions we use here are based on [84].

6.2.1 Overview of the ASK system

The top-level architecture of the ASK System is shown in Figure 6.1. Every component in this architecture has its own internal architecture, with several levels of hierarchical nesting. At its top-level, the ASK system consists of three parts: a *web front-end*, a *database* (Domain Data in Figure 6.1), and a *contact engine*. The *web front-end* deals with typical domain data, such as users, groups, phone numbers, mail

address, and so on. The *database* stores typical domain data, together with the feedback from users and knowledge from past experience. The *contact engine* handles the communication between the system and the outside world (e.g., by responding to or initiating telephone calls, SMS, emails, etc.) and provides appropriate matching and scheduling functionalities.

As mentioned above, the ASK system connects service providers and consumers for incoming requests. A connection is made when appropriate participants for a certain request are found. Until its proper connection is established, an incoming request loops through the system repeatedly as (sub-)tasks. This feature is called *Request loop* and it is represented by **thick arrows** in the contact engine in Figure 6.1.

The contact engine consists of five components: *Reception*, *Matcher*, *Executer*, *ResourceManager*, and *Scheduler*. The *Reception* component determines which steps must be taken by the ASK system to fulfill a request. According to the determined steps, the result of the Reception component is sent to either the *Matcher* or the *Executer* component. The *Matcher* component determines proper participants for fulfilling a request. The *Executer* component determines the best means of connection

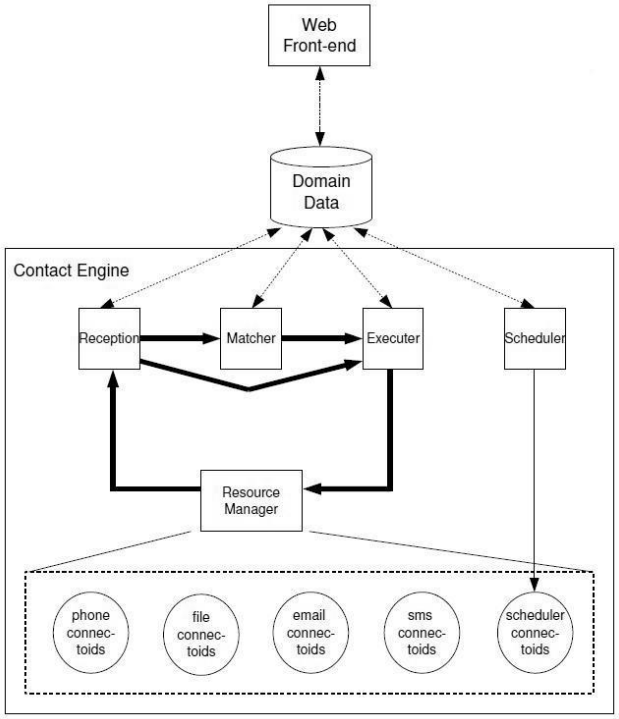


Figure 6.1: Overview of ASK system

between the participants. The *Resource Manager* component either uses the Request loop for complicated requests or establishes direct connections between users for trivial requests. The *Scheduler* component, separated from the components within the request loop, schedules requests based on the time constraints of the requests in the database. For example, an incoming call arrives from the outside. First, the Resource Manager component handles this call. If the request of the call is simple and trivial, then the Resource Manager component establishes the connection between users immediately. Otherwise, the request is sent to the Reception component. The Reception component gathers some information from users and stores the information in the database, and also determines if it needs to decide either the proper service participants or the efficient way of the connection between users. For determining the proper service providers, the request is sent to the Matcher component; for the efficient ways of connections, the request is sent to the Executer component. The Resource Manager provides the connection between the determined service participants using the determined means of connection. Actual connections occur based on the schedule by the Scheduler component.

6.3 Modeling the ASK system

In this section, we consider the contact engine, which contains the Request loop, and focus specifically on the Reception component. The components in the contact engine have very similar architectures, thus, the analysis carried out here for the Reception component can be used for the other ones, as well.

6.3.1 The Reception component

The Reception component consists of multiple threads, the so-called *ReceptionMonks (RMs)*, which handle incoming requests using two different types of functions:

- **HostessTask (HT)** which converts incoming requests into tasks that will be put into the task queue.
- **HandleRequestTask (HRT)** which takes care of the communication flow, interacts with the database, and possibly generates new requests which are dealt with by the Matcher or the Executer component. For example, given an incoming request, HRT may ask questions from users by playing pre-recorded messages, obtain information such as menu item choices, account number, etc., punched in by the users, and store this information into the database. During this communication, new requests can be generated and sent to other components.

Each thread runs one of these two different functions/tasks exclusively. That is, if an RM thread runs the HT function, then it is forbidden to run the HRT function. This implies that the Reception component needs to have at least two threads, one for the HT function and the other for the HRT function. In general, the HRT function

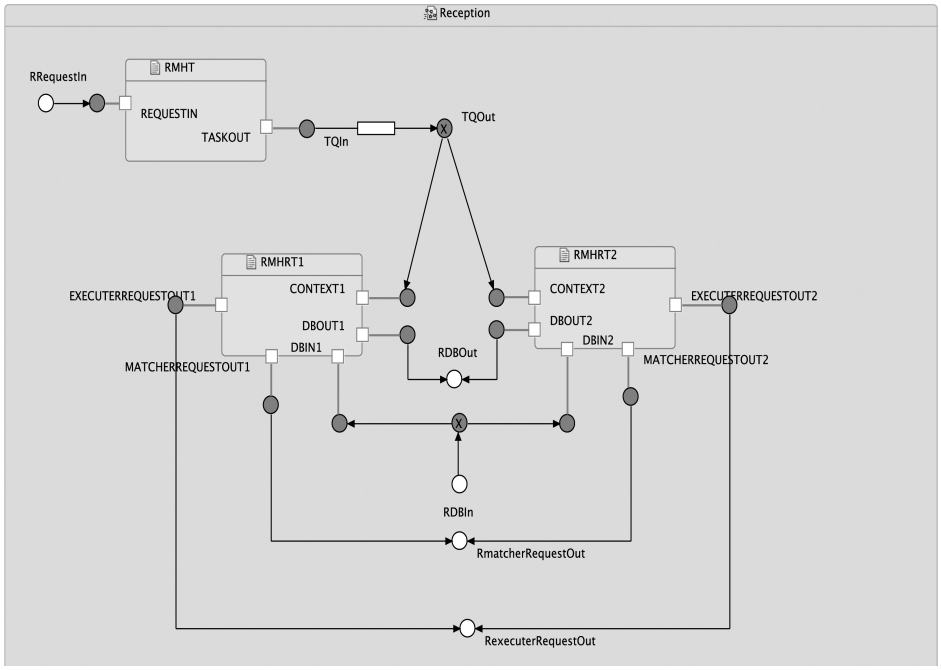
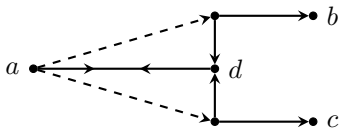


Figure 6.2: Reception component in ECT

takes more time than the HT function, since it actually deals with incoming tasks. Thus, the Reception component needs more threads running the HRT function. For simplicity of modeling, we assume that every thread in the Reception component has only one function, e.g., either the HT or the HRT function. Reflecting this simplification, Figure 6.2 shows the Reception model drawn in the Extensible Coordination Tools (ECT) [35]. This figure shows a Reception component with three RM threads, one with only the HT function and the other two with only the HRT function.

The inside boxes of the RMHT and the indexed RMHRTs in Figure 6.2 correspond to RM threads for a HT and HRT functions, respectively. Incoming requests are converted into tasks by the RMHT, and the converted tasks are stored in the task queue which is represented as a FIFO1 laid between the RMHT and the indexed RMHRTs. The converted tasks are selected and handled by the RMHRTs.

We model task selection as a non-deterministic choice at the *TQOut* node in Figure 6.2 (a sink node of the FIFO1 channel), which will turn into a random process once we associate the distributions of the stochastic variables that describe the actual task mix of a running system, as extracted from its logs, which will be explained in Section 6.3.2. The graphical notation $\otimes$ used for *TQOut* in Figure 6.2 is an abbreviation for an *exclusive router* [3] whose Reo circuit is depicted below.



This circuit delivers an incoming data item at node a to either node b or node c, whichever one can accept it, and non-deterministically selects one when both can. The non-deterministic choice is actually conducted by the merger d. Thus, the rates for the random selection apply to the merger d.

Figure 6.2 serves as a basic template model for the Reception component. Depending on the specific properties of interest in each analysis, we slightly adapt this basic template. For example, for the analysis of the properties of the task queue, we may substitute a LossyFIFO1 connector for the FIFO1 channel, as shown in Section 6.4.1. It should be noted that more than 3 RM threads can be used for modeling the Reception component, but here we use only 3 threads since the CTMC corresponding to the Stochastic Reo model of the Reception component with 3 RM threads is already big to handle.

6.3.2 Extracting distributions from logs

A stochastic model of the ASK system requires the distributions for all activities in the system. To obtain these distributions, we applied statistical data-analysis techniques on the raw values extracted from the real logs of a running ASK system. The logs contained the data of 100 incoming calls. Those calls simultaneously resulted in 369 requests sent to the Reception component. The trace holds exact timings of all actions performed related to each process.

We need to determine the rates for request arrivals (RRequestIn) and processing delay at the Reception component, reading request arrivals from the Matcher (RmatcherRequestOut) and the Executer (Rexecuter-RequestOut). For this purpose, after a cleanup of the raw data by removing outliers and erroneous data, we determined the appropriate distributions, using statistical tests (like the chi-square goodness-of-fit test).

For the Reo model, it is not important which type of distributions we obtain. However, to perform analysis using PRISM, which takes a CTMC as input, only exponential distributions can be used. In the case of request arrival rates, we may indeed assume that the inter-arrival times of the requests are exponentially distributed. This is reasonable since incoming calls to the ASK system are independent from each other, and the inter-arrival times are memoryless. However, in the case of processing delay rates, we were not able to conclude that the rates are exponentially distributed. The statistical tests showed that we may assume that the processing times follow a log-normal distribution.

6.4 QoS analysis

In this section, we show how to analyze the ASK system using both the CTMC and Reo Simulator approach. As mentioned in the previous section, the arrival or service times for some activities are not exponentially distributed. This is one of the reasons to analyze the Reo model with the Reo Simulator (see Section 6.4.2). The simulator was also used when we could not obtain any proper distribution from the logs at all. In this case, we used bootstrapping [69] in the simulator with the original data as special inputs in the simulator for the rates.

6.4.1 Analysis on derived CTMC

In this section, we analyze the ASK system to reveal some of its interesting properties in order to both evaluate and obtain clues for improving its performance. We carry out our analysis on the CTMC model derived from the Stochastic Reo model of the ASK system. We then feed the derived CTMC model as input to PRISM. In PRISM, properties of models are expressed using operations such as P, S, and R operators: the P operator is used to reason about the probability of the occurrence of a certain event; the S operator is used to reason about the steady-state behavior of a model; the R operator is used to analyze reward-based properties. In addition, labels are used to concisely express the formulas representing the properties of a model. Specifically, we use the following labels to express some properties later.

- `num_dataLoss` represents the number of task-loss in the task queue.
- `run` represents the running status of the RMHRT thread.

In general, resources are neither infinite nor free. Thus, one needs to balance cost-effective resource utilization against most efficient performance, i.e., obtaining the best performance taking into account the limited resource. In the Reception component in Figure 6.2, the resources of interest include:

1. the minimum capacity of the task queue
2. the utilization and/or the performance of the RMHRT threads that handle tasks

Task queue

As mentioned above, RMHT merely converts incoming requests into tasks, but it does not actually handles the requests. In general, the conversion into tasks does not take long, whereas handling a request may take considerable time. Thus, if the task queue has a small capacity, then RMHT frequently waits as it is blocked until task queue capacity becomes available. On the other hand, if the task queue has a large capacity, RMHT remains idle most of the time and some queue capacity goes to waste. Therefore, we want to determine a reasonable (the least sufficient) size for the task queue to make the ASK system efficient. We can check the probability of RMHT

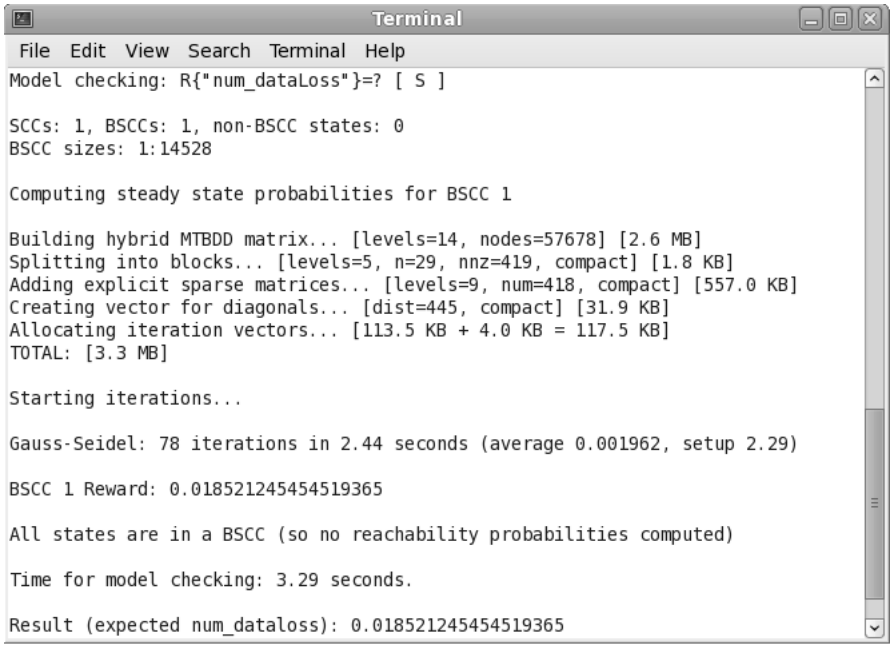


Figure 6.3: Long-run expected number of task-loss

blocking by iteratively increasing the queue capacity in subsequent runs, but this laborious approach is too time consuming. Alternatively, we can assume that the task queue has infinite capacity and try to find how much of it is actually used. With this task queue, we obtained the long-run expected number of task-loss due to unavailable buffer capacity or the unbalanced performance of RMHT and RMHRT threads. For this purpose, we use the following PRISM property `R{"num_dataLoss"}=?[S]`. The result is shown in the screen-shot in Figure 6.3.

To mimic an infinite queue, we use a LossySync channel feeding into a queue with a fixed capacity. This construct always accepts arriving tasks, but arriving tasks are lost when the queue is full. We can approximate the minimum required queue capacity out of the expected number of losing tasks by this construct. Replacing the FIFO1 queue in Figure 6.2 by the LossyFIFO1 connector in Figure 4.1 provides such a pseudo-infinite task queue for this analysis. According to this result, around 18.5^2 requests are lost per second in front of the task queue. From this result, we can conclude that the minimum capacity of the task queue needs to be 20 to guarantee no task-loss.

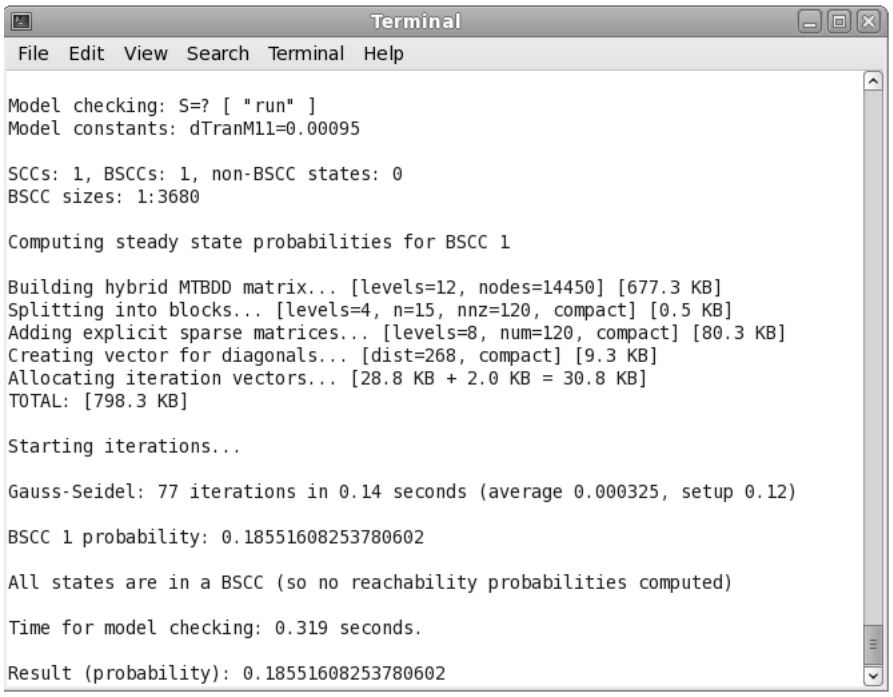


Figure 6.4: Probability that RMHRT1 is running

Functions

The RMHRT threads are the primary task handling processes. Thus, the performance of the Reception component depends on the collective performance of its RMHRT threads. It is interesting to learn how many RMHRT threads are required to handle a task load, or what is the reasonable performance of RMHRT threads that can provide a satisfactory QoS. Instead of changing the number of RMHRT threads, here we fix their number at 2 and vary their performance by changing their processing delay rates. These two threads have the same architecture with the same performance, thus, the analysis on the utilization is carried out on the RMHRT1 thread, the result of which can be used for the other RMHRT thread. We first find the steady-state probability that the RMHRT1 thread is running, expressed as `S=?["run"]` in PRISM. The result, shown in Figure 6.4, implies that the utilization of the RMHRT1 is 18%.

In a series of analysis experiments on this property, we varied the processing delay rates for the RMHRT1 thread. However, the gaps between the experiment results are not significant. For example, when we considered the activity of the RMHRT1 as an immediate activity by setting its rate as infinity, the steady-state probability

[2]The result 0.0185 was derived with millisecond as time unit.

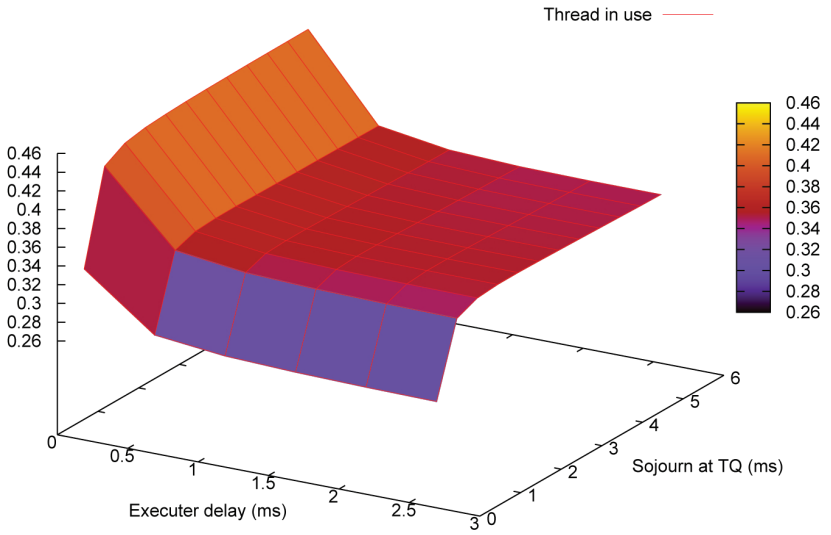


Figure 6.5: Steady-state probability `S=?["run"]`

`S=?[“run”]` from this rate value was 14%. Compared to the huge differences between these two values for the delay rate of the RMHRT1 thread[3], their resulting probabilities are barely changed. This implies that improving the performance of the RMHRT1 thread does not influence the overall performance of the Reception component that much, which suggests the presence of some bottlenecks in this system.

In order to figure out the bottlenecks, we experimented with the model by varying the rates relevant to other activities in the system. Figure 6.5 shows the probability results of these experiments. The label `Sojourn at TQ` presents the exit rate from the task queue. As this rate decreases, incoming requests stay longer in the task queue, and the RMHRT threads become more idle, i.e., the probability of the thread utilization decreases, since the request arrive at the thread less frequently. The graph in Figure 6.5 shows this tendency when one projects this graph onto the (`Prob.`, `Soj.`) plane. This implies that increasing `Sojourn at TQ` value generates higher utilization of the thread.

The label `Executer delay` represents the frequency that the Executer component takes the output from the Reception component. As this rate decreases, the threads in the Reception component need to keep their results waiting longer and block incoming tasks. Thus, the thread becomes less idle, i.e., the utilization of the thread increases, but their throughput becomes low since the thread just waits without doing anything. This tendency is also observable in the graph in Figure 6.5 when one

[3]The original rate value derived from the statistical analysis is 0.095, and we used the value $2^{31}-1$ as an infinity for this comparison.

projects this graph onto the (`Prob.`, `Exe.`) plane. To obtain meaningful utilization, we must increase `Executer delay`.

Based on the graph in Figure 6.5, we now determine bottlenecks in this system. In general, a small change in a bottleneck causes significant differences for the overall performance. The graph in Figure 6.5 shows an instance of this: variations in the rates in the interval [0.1, 0.6] for both `Executer delay` and `Sojourn at TQ` induce a big variation on the probability of utilization of the thread (represented in the vertical axis). Thus, these two rates can be assumed to be bottlenecks, which limit the overall performance. In order to mitigate these bottlenecks, we need to increase both rates at least above 0.6. However, we cannot increase these rates enormously since their relevant resources are neither infinite nor free. As a criterion for this increase, we can consider the convergent disposition of this graph. Above the value 1.3 of the respective rates, the utilization of the thread converges. Thus, we can choose 1.3 as the values of the respective rates for the best cost-effective utilization of the thread in this system.

6.4.2 Simulation

The Stochastic Reo simulator [51, 89] supports performance evaluation of Reo models through simulation. It allows arbitrary distributions for describing stochastic properties of channels and components. The method used by this tool combines simulation techniques and specific stochastic automata models to conduct automated performance analysis of both steady-state and transient properties of the model. The tool uses the coloring semantics [30] of Reo to properly model context-dependent behavior, i.e. to express the availability of requests. The Stochastic Reo simulator tool is developed as a plug-in within the ECT by Oscar Kanters. Through the GUI editor of the ECT, one can develop a model of a system as a Reo circuit in an intuitive way, annotate the circuit with rates, and then use the simulator to get insight into the behavior of the model.

For the simulation of the ASK system, we also focus on the Reception component. Since the other components in the ASK system have very similar architectures, we can use some of the results from this simulation for their simulation as well. The Stochastic Reo simulator tool does not support hierarchical models yet. Therefore, to run our simulation we had to flatten the original Reo model for the ASK system (shown in Figure 6.2), abstract its nested components into FIFO1 channels, and somewhat simplify it to reflect some restrictions. The resulting Reo circuit that we use for this simulation analysis appears in Figure 6.6 as an ECT screen-shot.

As a plug-in within the ECT, the simulator is used and accessed using the `Simulation` tab under the Reo editor (See Figure 6.6). In the `Run` and `Result options` sub-tabs under the `Simulation` tab, users can set some variables such as `Type of simulation`[4], `Max total number of events`[5], and so on. Having clicked the Reo model to be analyzed and pressed the `Start simulation` button in the `Simulation`

[4]Long- or short-term simulation.

[5]When the simulation is based on events, the length of the simulation is determined by the given event numbers.

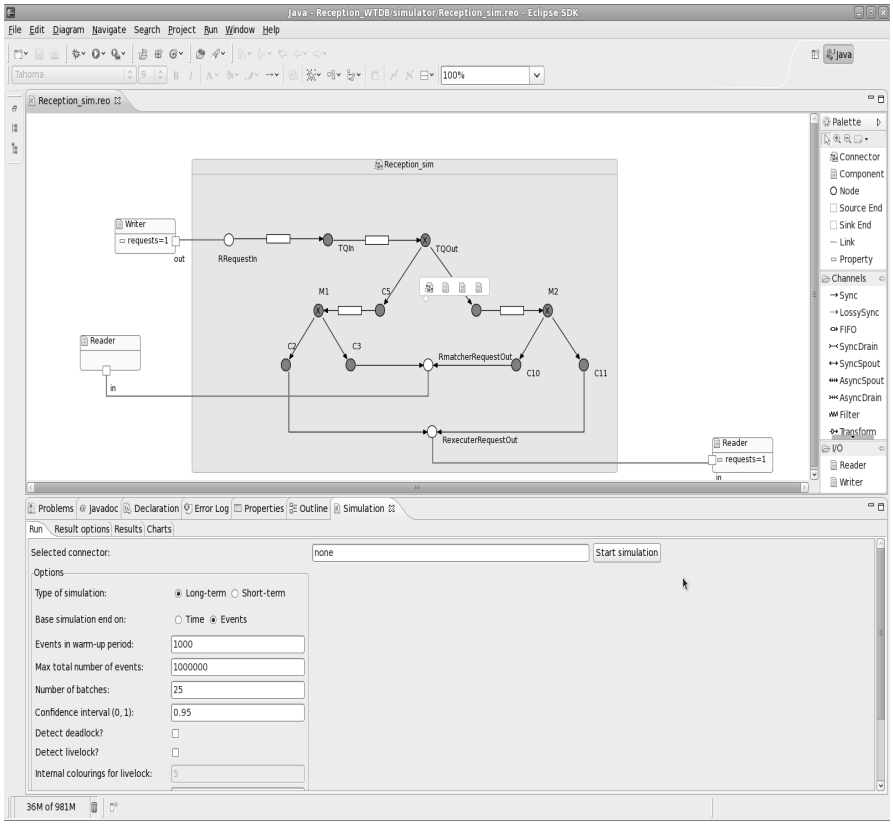


Figure 6.6: Flattened and simplified Reception model

tab, the simulation on the model is carried out. The simulation results are presented in other sub-tabs of `Results` and `Charts`. More detailed explanation on the usage and the underlying methods of this Reo simulator is given by [51].

The simulator provides information about

1. buffer utilization
2. end-to-end delays
3. the average waiting times of I/O requests at boundary nodes
4. channel utilization

The following examples show this information applying the simulator on the ASK system. Because we are not intended in channel utilization in this section, we assumed

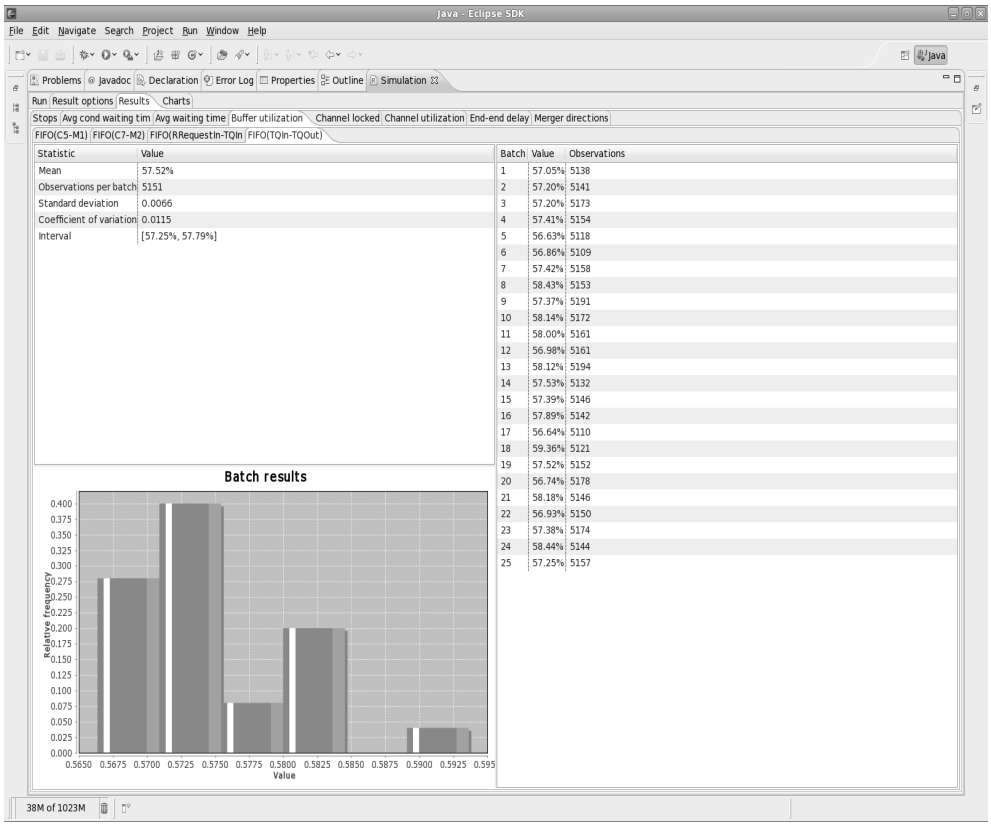


Figure 6.7: Buffer utilization of the task queue

most data-flows through channels are immediate actions. The distributions used for this simulation are also derived from the real logs of an actual running ASK system.

As an example of the simulation results we obtain, Figure 6.7 presents the ECT screen-shot that shows the utilization of the buffer between the TQIn and TQOut nodes. Actually, this buffer corresponds to the task queue in the Reception component, whose average utilization, according to this analysis, is 57.52%. That is, on the average, this buffer is used during 57% of the running time of the ASK system.

In addition, the graph at the bottom of the result in Figure 6.7 shows how frequently this average value occurs during simulation. In general, this value follows a normal distribution, which in this case does not happen. This can be due to a low number of batches: if this number increases, our graph may tend to a normal distribution.

As another example of the simulation, Figure 6.8 presents the ECT screen-shot that shows one of the end-to-end delays from the *RRequestIn* node to the *RmatcherRe-*

questOut node in Figure 6.6. This delay implies how long it takes for the Reception component to handle an incoming request and to send its result to the Matcher component. The average end-to-end delay is around 6500 microseconds, i.e., 6.5 milliseconds.

As the last example, Figure 6.9 presents the screen-shot that shows the waiting time of I/O request at *RRequestIn* node in Figure 6.6. According to this result, on the average, I/O requests wait 1727 microseconds, i.e., around 1.7 milliseconds.

6.5 Discussion

In this chapter, we have presented a stochastic analysis of (a deployed installation of) the ASK system. We modeled the system using Stochastic Reo, from which we generated the CTMCs corresponding to some of the modules of the system. This enabled

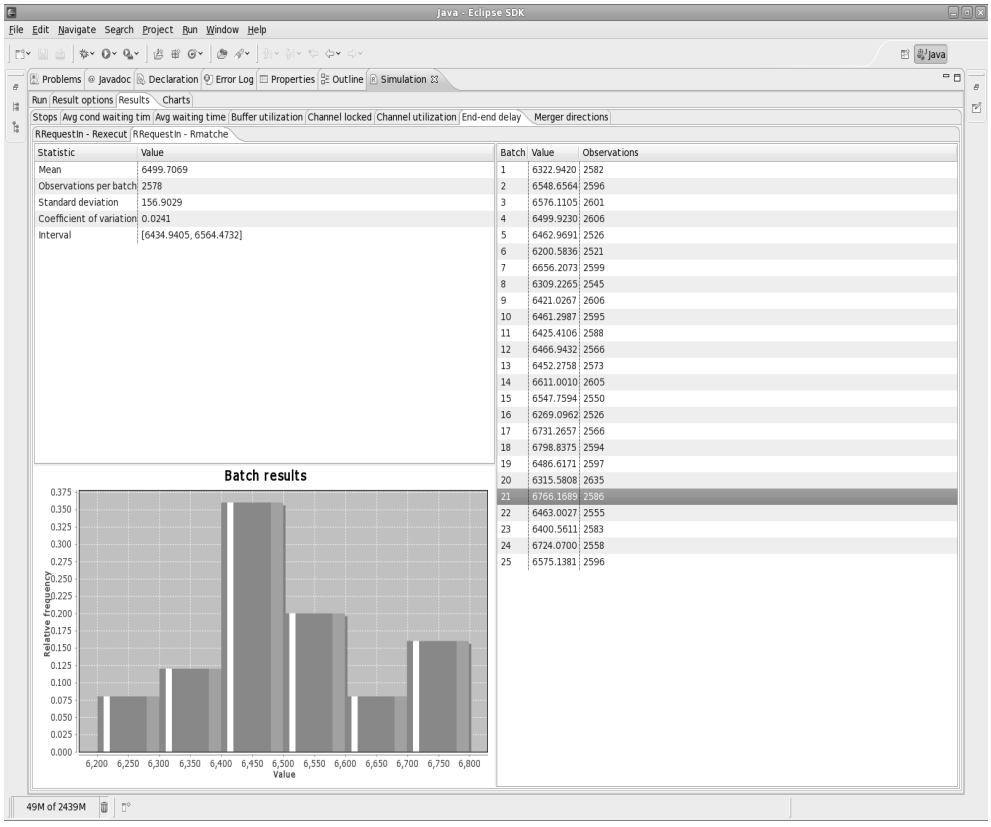


Figure 6.8: End-to-End delay from *RRequestIn* to *RmatcherRequestOut* in Figure 6.6

us to use the probabilistic model checker PRISM to verify some properties of interest, using the concrete stochastic distributions extracted from the logs of the running ASK installation. The results of this verification allowed us to draw conclusions about resource allocation and how the system installation can be adapted in order to improve its performance. CTMC models have the limitation of supporting only exponential distributions. To overcome this limitation, we also used a simulator. Even though the result from the simulation is approximation-based analysis, we can gain insight into the aspects of the behavior of the system that involve non-exponential distributions.

We have focused our analysis in this chapter only on the Reception component of the ASK system. However, the other components have very similar architectures and, thus, all the techniques used in this chapter can be easily applied to them as well.

The distributions used in this case study were obtained by statistical analysis based on the real logs of an actual running ASK system. Our analysis revealed ex-

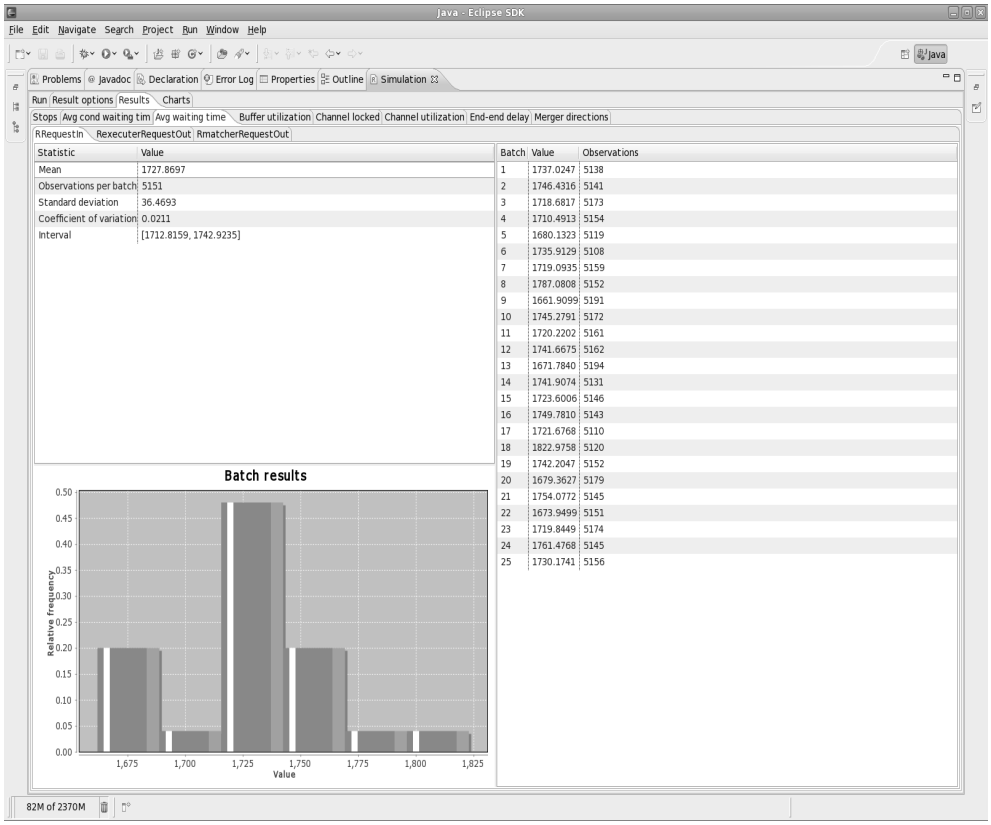


Figure 6.9: Waiting time of I/O request at *RRequestIn*

ponential distributions for the arrivals and I/O requests. However, rates for the processing/service times of some components were not exponentially distributed. This made it necessary to do simulation for additional analysis. We used the Reo simulator [51, 89], an integrated ECT tool, which enables the use of arbitrary distributions and predefined probabilistic behaviors. Using this simulator we can study a model which, for instance, has exponentially distributed data arrivals and log-normal distributed processing rates in some components.

In this analysis, we found two bottlenecks that were caused by (1) the low availability of the Executer component and (2) the long sojourn time at the task queue. In what concerns (1), we observe that we are modeling the connections between the Reception and other components (Executer and Matcher) synchronously (that is, using Sync channels), and that the observation that the consumption rates of the other two components become bottlenecks is not surprising. We have experimented with replacing the Sync channels with FIFOs to decouple the components and remove these bottlenecks. In the process of these experiments, we identified another bottleneck internal to the Executer component itself. In what concerns (2), the bottleneck is caused by congestion between the task queue and the threads. Thus, we can widen the bandwidth of this connection to obtain better performance for the system.

In earlier initiatives to improve the performance of the ASK system, the focus has been primarily on improving the execution times of request handling tasks, through extensive profiling. The work presented in this chapter confirms and explains the observations from small experiments with ASK components in isolation, carried out by Almende last year. As a consequence of this, Almende decided to put additional effort into the optimization of queue sizes and bandwidth between the task queue and the threads in each of the ASK components. First attempts in this direction yield promising results.

Chapter 7

Conclusions and Future work

In this chapter, we conclude this thesis with a summary of what we presented in the previous chapters and a discussion of a number of future activities to extend the work presented in this thesis.

7.1 Conclusions

As the Internet has advanced in terms of accessibility, usability, and utility, the interest in using distributed services over networks for large-scale applications has increased. However, the composition of distributed applications is non-trivial because of their heterogeneity. When it comes to their quantitative aspects, it is challenging to specify and reason about the end-to-end QoS of composed applications.

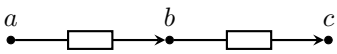
In this thesis, we provided a specification model, Stochastic Reo, to describe coordination in such composition while considering non-functional (QoS) aspects. As an operational semantic model for Stochastic Reo, Quantitative Intentional Automata (QIA) were introduced. This semantic model describes the data-flows through connectors and the interaction with the environment of the connectors separately, thus, it is appropriate to specify and reason about the end-to-end QoS in a composed application. However, in general, QIA have a large number of states. In order to overcome this, we introduced Stochastic Reo Automata as an alternative semantic model to QIA. Stochastic Reo Automata are not only more compact, but they also enabled us to prove a compositionality result easily, which was lacking for QIA. Both semantic models serve as intermediate models for generating corresponding CTMCs for stochastic analysis. In order to consider more general QoS aspects, we have extended Stochastic Reo Automata with reward information, and this extension is also propagated to CTMCs as state rewards using the translation method. This translation method has been implemented, in the Reo2MC tool, in the Extensible Coordination Tools (ECT) [35]. As a plug-in for ECT, Reo2MC provides the following functionalities: 1) editor for Stochastic Reo 2) generating semantic models for Stochastic Reo, in particular, QIA, 3) deriving a CTMC model for a Stochastic Reo. That is, Reo2MC can be seen as an integrated tool of a Stochastic Reo editor and a CTMC generator

from Stochastic Reo. We explained the implementation details of the Reo2MC, as well as its basic definitions and algorithms, and its usage. The output of the Reo2MC can be fed to other tools, such as PRISM, MATLAB, and Maple, for stochastic analysis. As a case study, we analyzed the ASK system, which is an industrial software and acts as a mediator between service consumers and service providers. The ASK system is specified using Stochastic Reo, whereby this model is translated into corresponding CTMC for analysis using PRISM. The rates used in this model were obtained by applying statistical analysis techniques on the raw values that we obtained from the real logs of an actual running ASK system. The results of this verification allowed us to draw conclusions about resource allocation and how the system installation can be adapted in order to improve its performance.

7.2 Future work

Stochastic Reo does not impose any restriction on the distribution classes of its annotated rates, such as the rates for request-arrivals at channel ends or data-flows through channels. However, for the translation from Stochastic Reo into an homogeneous CTMC model, we considered only exponential distributions for the rates. For example, in the case study using the ASK system, if the rates, obtained from statistical analysis on the raw values extracted from the real logs of a running ASK system, were not exponentially distributed, then we had to assume the obtained rates as exponential distributions or used other techniques, such as bootstrapping, to get meaningful rates. Thus, in order to support the general usage of Stochastic Reo, we want to consider non-exponential distributions such as phase-type distributions or using Semi-Markov Processes [50] as target models of our translation.

In addition, during the case study, we encountered models whose state spaces were too large to analyze. However, all the states in such a model are not meaningful because some of them are caused by the structure of Reo primitive channels, not the behavior of connectors. For example, consider the following connector:



The data-flows from the first buffer to node b and from node b to the second buffer are considered as two different events and represented sequentially in its corresponding CTMC model. In general, we may not be interested in which buffer is full when one of the two buffers is occupied. In this case, it is more meaningful to make these two data-flows immediate events, to reduce the configurations of this connector to include only *empty, half-full,* and *full*. For this purpose, we want to *hide* node b in this connector. However, hiding nodes is not trivial since it can lead to the loss of the structural information of connectors, which is used to generate corresponding CTMCs from the connectors. Thus, it is an interesting future work direction to find certain patterns of hiding nodes, which still allows to generate CTMCs with correct operational semantics. Moreover, the implementation of checking these pattens for

hiding will be the next step in order to provide users with safe selection of nodes for hiding. This will help to mitigate the large state space of the derived CTMCs.

Compared to CTMCs, Interactive Markov Chains (IMCs) are compositional, thus, the IMC for a complex system can be built out of IMCs corresponding sub-systems constituting the complex system. Then, one might wonder why IMCs are not used as our stochastic target model without the translation via other operational semantic models. To answer this question, we discussed why IMCs are not an appropriate semantic model for Stochastic Reo since it generates unintended transitions that are produced in synchrony propagation scenarios. In addition, we showed the translation from Stochastic Reo into IMCs via Stochastic Reo Automata. A natural and interesting future work is to consider whether it is possible to adapt the composition operator of IMCs in order to delete the unintended transitions and still remain within a compositional framework.

So far, the Reo2MC tool uses QIA (instead of Stochastic Reo Automata) as an intermediate model for its translation. We are currently extending and improving these tools to use Stochastic Reo Automata, as well as the extension with reward information, so that the more compact sizes of the automata models will then allow us to analyze larger models.

The connection considered in this thesis is described without considering the overhead of establishing the coordination between components. That is, for a Reo connector, we considered only the interaction with the environment of the connector and its internal processing, i.e., data-flows between its boundary nodes, and the coordination was assumed to be established immediately. To be more realistic, we also need to consider the overhead of establishing the coordination between the boundary nodes before the internal processing of the connector occurs. In [54], Action Constraint Automata (ACA) were proposed to specify such a coordination processing in Reo connectors in a compositional manner. However, they do not include the interaction with the I/O requests of the connectors, and moreover, ACA do not account for the QoS aspects of the connectors. Thus, it will be an interesting and meaningful future work to provide a stochastic extension of the specification for the coordination processing in ACA and to, in turn, combine this extension and our specification approach, which enables us to analyze and reason about more realistic end-to-end QoS of a Reo connector.

Bibliography

[1] Almende website. `http://www.almende.com`.

[2] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[3] Farhad Arbab. Abstract Behavior Types: a foundation model for components and their composition. *Science of Computer Programming*, 55(1-3):3–52, 2005.

[4] Farhad Arbab. *Composition of Interacting Computations*, chapter 12, pages 277–321. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] Farhad Arbab, Tom Chothia, Sun Meng, and Young-Joo Moon. Component Connectors with QoS Guarantees. In *COORDINATION*, volume 4467 of *Lecture Notes in Computer Science*, pages 286–304. Springer, 2007.

[6] Farhad Arbab, Tom Chothia, Rob van der Mei, Sun Meng, Young-Joo Moon, and Chrétien Verhoef. From Coordination to Stochastic Models of QoS. In *COORDINATION*, volume 5521 of *Lecture Notes in Computer Science*, pages 268–287. Springer, 2009.

[7] Farhad Arbab, Ivan Herman, and Pål Spilling. An overview of manifold and its implementation. *Concurrency - Practice and Experience*, 5(1):23–70, 1993.

[8] Farhad Arbab, Sun Meng, Young-Joo Moon, Marta Z. Kwiatkowska, and Hongyang Qu. Reo2MC: a tool chain for performance analysis of coordination models. In *ESEC/SIGSOFT FSE*, pages 287–288. ACM, 2009.

[9] Farhad Arbab and Jan J. M. M. Rutten. A Coinductive Calculus of Component Connectors. In *WADT*, pages 34–55, 2002.

[10] ASK Community systems website. `http://www.ask-cs.com`.

[11] Jos C. M. Baeten and W. Peter Weijland. *Process Algebra*. Cambridge University Press, 1990.

[12] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan J. M. M. Rutten. Modeling component connectors in Reo by constraint automata. *Science of Computer Programming*, 61(2):75–113, 2006.

[13] Christel Baier and Verena Wolf. Stochastic Reasoning About Channel-Based Component Connectors. In *COORDINATION*, volume 4038 of *Lecture Nonte in Computer Science*, pages 1–15. Springer, 2006.

[14] Marco Antonio Barbosa, Luís Soares Barbosa, and José Creissac Campos. Towards a Coordination Model for Interactive Systems. *Electronic Notes in Theoretical Computer Science*, 183:89–103, 2007.

[15] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *SEFM*, pages 3–12. IEEE Computer Society, 2006.

[16] Marco Bernardo and Roberto Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. Technical Report UBLCS-96-17, 1996.

[17] Marco Bernardo and Roberto Gorrieri. Extended Markovian Process Algebra. In *CONCUR*, volume 1119 of *Lecture Notes in Computer Science*, pages 315–330. Springer, 1996.

[18] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[19] Marcello Bonsangue, Dave Clarke, and Alexandra Silva. A model of context-dependent component connectors. *Science of Computer Programming*, In Press, Corrected Proof, 2011.

[20] Hichem Boudali, Pepijn Crouzen, Boudewijn R. Haverkort, Matthias Kuntz, and Mariëlle Stoelinga. Architectural dependability evaluation with Arcade. In *International Conference on Dependable Systems and Networks*, pages 512–521. IEEE Computer Society, 2008.

[21] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains. In *ATVA*, volume 4762 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2007.

[22] Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. Dynamic Fault Tree Analysis Using Input/Output Interactive Markov Chains. In *International Conference on Dependable Systems and Networks*, pages 708–717. IEEE Computer Society, 2007.

[23] Hichem Boudali, Pepijn Crouzen, and Marielle Stoelinga. A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis. *IEEE Transactions on Dependable and Secure Computing*, 7(2):128–143, 2010.

[24] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Viet Yen Nguyen, Thomas Noll, and Marco Roveri. The COMPASS Approach: Correctness, Modelling and Performability of Aerospace Systems. In *SAFECOMP*, volume 5775 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2009.

[25] Marco Bozzano, Alessandro Cimatti, Marco Roveri, Joost-Pieter Katoen, Viet Yen Nguyen, and Thomas Noll. Codesign of dependable systems: a component-based modeling language. In *MEMOCODE'09: Proceedings of the 7th IEEE/ACM international conference on Formal Methods and Models for Codesign*, pages 121–130, Piscataway, NJ, USA, 2009. IEEE Press.

[26] Mario Bravetti and Marco Bernardo. Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time. *Electronic Notes in Theoretical Computer Science*, 39(3), 2000.

[27] Peter Buchholz. Hierarchical Markovian Models: Symmetries and Reduction. *Performance Evaluation*, 22(1):93–110, 1995.

[28] Behnaz Changizi, Natallia Kokash, and Farhad Arbab. A Unified Toolset for Business Process Model Formalization. In *7th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA 2010)*, pages 147–156, 2010.

[29] Tom Chothia and Jetty Kleijn. Q-Automata: Modelling the Resource Usage of Concurrent Components. *Electronic Notes in Theoretical Computer Science*, 175(2):153–167, 2007.

[30] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency. *Science of Computer Programming*, 66(3):205–225, 2007.

[31] David Costa. *Formal Models for Context Dependent Connectors for Distributed Software Components and Services.* PhD thesis, Vrij Universiteit Amsterdam, 2010.

[32] David Costa, Milad Niqui, and Jan J. M. M. Rutten. Intentional Automata: A Context-Dependent Model For Component Connectors (Extended Abstract). In *FSEN*, 2011.

[33] Credo project. `http://projects.cwi.nl/credo/`.

[34] Frank S. de Boer, Immo Grabe, Mohammad Mahdi Jaghoori, Andries Stam, and Wang Yi. Modeling and Analysis of Thread-Pools in an Industrial Communication Platform. In *Proc. 11th International Conference on Formal Engineering Methods (ICFEM'09)*, volume 5885 of *Lecture Notes in Computer Science*, pages 367–386. Springer, 2009.

[35] Extensible Coordination Tools. `http://reo.project.cwi.nl/`.

[36] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[37] Paulo Fernandes, Brigitte Plateau, and William J. Stewart. Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks. *Journal of the ACM*, 45(3):381–414, 1998.

[38] Cédric Fournet and Georges Gonthier. The Join Calculus: A Language for Distributed Mobile Programming. In *APPSEM*, volume 2395 of *Lecture Notes in Computer Science*, pages 268–332. Springer, 2000.

[39] Hubert Garavel and Holger Hermanns. On Combining Functional Verification and Performance Evaluation Using CADP. In *FME*, volume 2391 of *Lecture Notes in Computer Science*, pages 410–429. Springer, 2002.

[40] Hubert Garavel, Radu Mateescu, Frédéric Lang, and Wendelin Serwe. CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer, 2007.

[41] David Gelernter. Generative Communication in Linda. *ACM Transaction on Programming Languages and Systems*, 7(1):80–112, 1985.

[42] Peter G. Harrison and Jane Hillston. Exploiting Quasi-reversible Structures in Markovian Process Algebra Models. *The Computer Journal*, 38(7):510–520, 1995.

[43] Holger Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Nonte in Computer Science*. Springer, 2002.

[44] Holger Hermanns and Joost-Pieter Katoen. The How and Why of Interactive Markov Chains. In *Formal Methods for Components and Objects (FMCO)*, volume 6286 of *Lecture Notes in Computer Science*, pages 311—-337. Springer-Verlag, 2010.

[45] Ulrich Herzog. Formal Description, Time and Performance Analysis. A Framework. In *Entwurf und Betrieb verteilter Systeme*, pages 172–190, London, UK, 1990. Springer-Verlag.

[46] Jane Hillston. PEPA: Performance Enhanced Process Algebra. Technical Report CSR-24-93, University of Edinburgh, 1993.

[47] Jane Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, April 1994.

[48] Andrew Hinton, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. In *TACAS*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2006.

[49] C. A. R. Hoare. Communicating sequential processes. *Communication of the ACM*, 21(8):666–677, 1978.

[50] Håkan L. S. Younes and Reid G. Simmons. Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 742–748. California AAAI Press, 2004.

[51] Oscar Kanters. QoS analysis by simulation in Reo. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, 2010.

[52] Joost-Pieter Katoen and Pedro R. D'Argenio. General Distributions in Process Algebra. In *European Educational Forum: School on Formal Methods and Performance Analysis*, volume 2090 of *Lecture Notes in Computer Science*, pages 375–430. Springer, 2000.

[53] Christian Koehler, Alexander Lazovik, and Farhad Arbab. Connector Rewriting with High-Level Replacement Systems. *Electr. Notes Theor. Comput. Sci.*, 194(4):77–92, 2008.

[54] Natallia Kokash, Behnaz Changizi, and Farhad Arbab. A Semantic Model for Service Composition with Coordination Time Delays. In *ICFEM*, volume 6447 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2010.

[55] Christian Krause. *Reconfigurable Component Connectors*. Phd thesis, Universiteit Leiden, To appear in 2011.

[56] Christian Krause, Ziyan Maraikar, Alexander Lazovik, and Farhad Arbab. Modeling Dynamic Reconfigurations in Reo using High-Level Replacement Systems. *Science of Computer Programming*, 76(1):23–36, 2011. Selected papers from the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures - FOCLASA'07.

[57] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation/TOOLS*, pages 200–204, 2002.

[58] Bilung Lee and Edward A. Lee. Hierarchical Concurrent Finite State Machines in Ptolemy. In *ACSD*, pages 34–40. IEEE Computer Society, 1998.

[59] Xiaojun Liu, Yuhong Xiong, and Edward A. Lee. The Ptolemy II Framework for Visual Languages. In *HCC*. IEEE Computer Society, 2001.

[60] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. Modelling with Generalized Stochastic Petri Nets. *SIGMETRICS Performance Evaluation Review*, 26(2):2, 1998.

[61] Sun Meng and Farhad Arbab. QoS-Driven Service Selection and Composition Using Quantitative Constraint Automata. *Fundamenta Informaticae*, 95(1):103–128, 2009.

[62] Vassilis Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, University of Erlangen, 1998.

[63] Robin Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[64] Jayadev Misra and William R. Cook. Computation Orchestration: A basis for wide-area computing. *Software and System Modeling*, 6(1):83–110, 2007.

[65] Michael Karl Molloy. *On the integration of delay and throughput measures in distributed processing models*. PhD thesis, 1981.

[66] Young-Joo Moon, Farhad Arbab, Alexandra Silva, Andries Stam, and Chrétien Verhoef. Stochastic Reo: a Case Study. Accepted for publication in TTSS 2011.

[67] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. A Compositional Model to Reason about end-to-end QoS in Stochastic Reo Connectors. To apper in *Science of Computer Programming*, 2011.

[68] Young-Joo Moon, Alexandra Silva, Christian Krause, and Farhad Arbab. A Compositional Semantics for Stochastic Reo Connectors. In *FOCLASA*, volume 30 of *EPTCS*, pages 93–107, 2010.

[69] Christopher Z. Mooney and Robert D. Duval. *Bootstrapping: a nonparametric approach to statistical inference*. Sage Publications, 1993.

[70] Marcel F. Neuts. *Matrix-geometric Solutions in Stochastic Models: An Algorithmic Approach*. The Johns Hopkins University Press, 1981.

[71] Oscar Nierstrasz. Piccola - A Small Compositional Language (Invited Talk). In *FMOODS*, volume 139 of *IFIP Conference Proceedings*. Kluwer, 1999.

[72] C. A. O'Cinneide. Characterization of phase-type distributions. *Stochastic Models*, 6(1):1–57, 1990.

[73] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *M. Zelkowitz (Ed.), The Engineering of Large Systems*, volume 46 of *Advances in Computers*, pages 329–400. Academic Press, 1998.

[74] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1981.

[75] Brigitte Plateau and Karim Atif. Stochastic Automata Network of Modeling Parallel Systems. *IEEE Transactions on Software Engineering*, 17:1093–1108, 1991.

[76] Prism website. `http://www.prismmodelchecker.org/`.

[77] José Proença. *Synchronous Coordination of Distributed Components*. PhD thesis, Universiteit Leiden, To appear in 2011.

[78] José Proença and Dave Clarke. Coordination Models Orc and Reo Compared. *Electronic Notes in Theoretical Computer Science*, 194(4):57–76, 2008.

[79] Wolfgang Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[80] Juan Guillen Scholten. *Mobile channels for exogenous coordination of distributed systems: semantics, implementation and composition*. Phd thesis, Leiden University, 2007.

[81] Mary Shaw and David Garland. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, 1996.

[82] Markus Siegle. On Efficient Markovian Modelling. In *In Proc. QMIPS Workshop on Stochastic Petri Nets*, pages 213–225, 1992.

[83] Andries Stam. The ASK System and the Challenge of Distributed Knowledge Discovery. In *ISoLA*, volume 17 of *Communications in Computer and Information Science*, pages 663–668. Springer, 2008.

[84] Andries Stam, Sascha Klüppelholz, Tobias Blechmann, and Joachim Klein. ReASK Final Models. Technical Report To be appeared, Almende, The Netherlands and Technical University of Dresden, Germany, 2009.

[85] William J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[86] William J. Stewart, Karim Atif, and Brigette Plateau. The numerical solution of stochastic automata networks. *European Journal of Operational Research*, (3):503–525, 1995.

[87] F. J. W. Symons. Introduction to Numerical Petri Nets, a General Graphical Model of Concurrent Processing Systems. *Australian Telecommunications Research*, 14(1):28–32, 1980.

[88] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[89] Chrétien Verhoef, Christian Krause, Oscar Kanters, and Rob van der Mei. Simulation-based Performance Analysis of Channel-based Coordination Models. In *COORDINATION 2011*, volume 6721 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, 2011.

Abstract

The intensifying need for scalable software has motivated modular development and using systems distributed over networks to implement large-scale applications. In Service-oriented Computing, distributed services are composed to provide large-scale services with a specific functionality. In this way, reusability of existing services can be increased. However, due to the heterogeneity of distributed software systems, software composition is far from trivial, and requires additional mechanisms to impose some form of a coordination on a distributed software system. For this purpose, a number of coordination languages have been proposed, such as Reo, Linda, and Orc.

Besides functional correctness, a composed service must satisfy various quantitative/non-functional requirements for its clients, which are generically called its quality of service (QoS). For instance, although a number of services may offer the same functionality, some of them may accommodate tight deadlines, but others may not. In particular, it is tricky to obtain the overall QoS of a composed service even if the QoS information of its constituent distributed services is given.

In this thesis, Stochastic Reo is proposed, a formalism to specify software composition with QoS aspects. Stochastic Reo is an extension of Reo, a channel/connector-based coordination language, with associated stochastic values which indicate the frequency of I/O interactions and internal processing delays within connector primitives.

As a semantic model of Stochastic Reo, we propose two different automata models, namely, Quantitative Intentional Automata and Stochastic Reo Automata. Stochastic Reo Automata are compositional, which enables us to obtain the automata model of a complex connector by composing the automata models of its constituent primitive connectors. A formal proof of compositionality is included in the thesis. These two semantic models are also used as intermediate models in order to generate their corresponding stochastic models, especially, Continuous-time Markov Chains (CTMCs) and Interactive Markov Chains. These stochastic models can be used for practical analysis of the underlying connectors.

Based on this theory, we have implemented the tool *Reo2MC* as a plug-in within the Reo toolset, Extensible Coordination Tools. Reo2MC generates CTMCs corresponding to Reo connectors, which are given to or drawn in the tool, via the semantic models of the Reo connectors.

As a case study, we have modeled and analyzed the ASK system using Reo2MC. The ASK system is an industrial software developed by the Dutch company Almende. Its analysis results provided the best cost-effective resource utilization and some suggestions to improve the performance of the ASK system. For example, the results provided suggestion of the required minimum capacity of a task queue and detected some bottlenecks in the system.

In summary, this thesis proposes formal models to specify the behavior of connectors coordinating distributed software over a network, and to reason about the end-to-end QoS properties of the connectors. This thesis also shows how to translate the semantic models of connectors into their corresponding stochastic models for further analysis. The theoretical results obtained in this thesis have been implemented and integrated as a plug-in into an existing tool set. The practical relevance of the approach is demonstrated by modeling and analyzing a large industrial software using the tool, which resulted in improvements to the analyzed system.

Samenvatting

De steeds groter wordende behoefte aan schaalbare software is de motivatie geweest voor modulaire ontwikkeling en het gebruik van over netwerken gedistribueerde systemen om grootschalige applicaties te implementeren. In Service-oriented Computing worden gedistribueerde services samengesteld om grootschalige services met een specifieke functionaliteit aan te bieden. Hierdoor kan de herbruikbaarheid van bestaande systemen vergroot worden. Als gevolg van de diversiteit aan gedistribueerde softwaresystemen is de samenstelling van software echter verre van triviaal, en zijn bijkomende mechanismes nodig om gedistribueerde softwaresystemen te kunnen coördineren. Voor dit doeleinde is een aantal coördinatietalen voorgesteld, waaronder Reo, Linda en Orc.

Afgezien van functionele correctheid moet een samengestelde service voldoen aan verschillende kwantitatieve en niet-functionele eisen voor de cliënten ervan, die in het algemeen de 'quality of service' (QoS) genoemd worden. Het kan bijvoorbeeld zo zijn dat, zelfs als een aantal services dezelfde functionaliteit biedt, sommige stricte deadlines accomoderen terwijl andere dit niet doen. Het is in het bijzonder lastig om de algehele QoS van een samengestelde service te verkrijgen, zelfs als de QoS-informatie voor de onderliggende constituenten een gegeven is.

In dit proefschrift wordt Stochastic Reo voorgesteld, een formalisme om samenstelling van software met QoS-aspecten te specificeren. Stochastic Reo is een uitbreiding van Reo, een channel/connector-gebaseerde coördinatietaal, met geassocieerde stochastische waarden die de frequentie van I/O-interacties, en de interne verwerkingsvertragingen in de primitieve connectoren, aangeven.

Als een semantisch model van Stochastic Reo stellen we twee verschillende automatenmodellen voor, namelijk Quantitative Intentional Automata en Stochastic Reo Automata. Stochastic Reo Automaten zijn compositioneel, wat ons in staat stelt om het automatenmodel van een complexe connector te verkrijgen door de automatenmodellen van de onderliggende primitieve connectoren samen te stellen. Een formeel bewijs van de compositionaliteit is in dit proefschrift te vinden. Deze twee semantische modellen worden ook gebruikt als tussenliggende modellen om de ermee corresponderende stochastische modellen te genereren, in het bijzonder Continuous-Time Markov Chains (CMTCs) en Interactive Markov Chains. Deze stochastische mod-

ellen kunnen gebruikt worden voor een praktische analyse van de onderliggende connectoren.

We hebben, gebaseerd op deze theorie, de tool *Reo2MC* geïmplementeerd als een plugin binnen de Reo-toolset, Extensible Coordination Tools. *Reo2MC* genereert CMTCs die corresponderen met Reo-connectoren, die in de tool aangegeven of getekend worden, via de semantische modellen van de Reo-connectoren.

We hebben het ASK systeem als een case study gemodelleerd en geanalyseerd met gebruik van *Reo2MC*. Het ASK systeem is een industrieel softwareproduct, ontwikkeld door het Nederlandse bedrijf Almende. De analyseresultaten gaven het beste kosteneffectieve resourcegebruik aan, en leverden een aantal suggesties voor verbetering van het ASK systeem. De resultaten gaven, als voorbeeld hiervan, een suggestie van de benodigde minimumcapaciteit van een task queue en hebben een aantal bottlenecks in het systeem gedetecteerd.

Samenvattend stelt dit proefschrift formele modellen voor, om het gedrag van connectoren die gedistribueerde software over een netwerk coördineren, te specificeren, en om over de end-to-end QoS-eigenschappen van de connectoren te redeneren. Dit proefschrift laat ook zien hoe de semantische modellen van connectoren vertaald moeten worden in de overeenkomstige stochastische modellen voor verdere analyse. De in dit proefschrift verkregen theoretische resultaten zijn geïmplementeerd en geïntegreerd als een plug-in binnen een bestaande toolset. De praktische relevantie van deze benadering is aangetoond door een groot industrieel softwareproduct te modelleren en analyseren met de tool, wat geresulteerd heeft in verbeteringen in het geanalyseerde systeem.

Titles in the IPA Dissertation Series since 2005

E. Ábrahám. *An Assertional Proof System for Multithreaded Java - Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and*

Rewriting. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization*. Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavicius. *Hybrid Techniques for Hybrid Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15

M.E. Warnier. *Language Based Security for Java and JML*. Faculty of Science, Mathematics and Computer Science, RU. 2006-16

V. Sundramoorthy. *At Home In Service Discovery*. Faculty of Electrical

Engineering, Mathematics & Computer Science, UT. 2006-17

B. Gebremichael. *Expressivity of Timed Automata Models*. Faculty of Science, Mathematics and Computer Science, RU. 2006-18

L.C.M. van Gool. *Formalising Interface Specifications*. Faculty of Mathematics and Computer Science, TU/e. 2006-19

C.J.F. Cremers. *Scyther - Semantics and Verification of Security Protocols*. Faculty of Mathematics and Computer Science, TU/e. 2006-20

J.V. Guillen Scholten. *Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition*. Faculty of Mathematics and Natural Sciences, UL. 2006-21

H.A. de Jong. *Flexible Heterogeneous Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-01

N.K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-02

M. van Veelen. *Considerations on Modeling for Early Detection of Abnormalities in Locally Autonomous Distributed Systems*. Faculty of Mathematics and Computing Sciences, RUG. 2007-03

T.D. Vu. *Semantics and Applications of Process and Program Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2007-04

L. Brandán Briones. *Theories for Model-based Testing: Real-time and Coverage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-05

I. Loeb. *Natural Deduction: Sharing by Presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2007-06

M.W.A. Streppel. *Multifunctional Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2007-07

N. Trčka. *Silent Steps in Transition Systems and Markov Chains*. Faculty of Mathematics and Computer Science, TU/e. 2007-08

R. Brinkman. *Searching in encrypted data*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-09

A. van Weelden. *Putting types to good use*. Faculty of Science, Mathematics and Computer Science, RU. 2007-10

J.A.R. Noppen. *Imperfect Information in Software Development Processes*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2007-11

R. Boumen. *Integration and Test plans for Complex Manufacturing Systems*. Faculty of Mechanical Engineering, TU/e. 2007-12

A.J. Wijs. *What to do Next?: Analysing and Optimising System Behaviour in Time*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2007-13

C.F.J. Lange. *Assessing and Improving the Quality of Modeling: A Series of*

Empirical Studies about the UML. Faculty of Mathematics and Computer Science, TU/e. 2007-14

T. van der Storm. *Component-based Configuration, Integration and Delivery*. Faculty of Natural Sciences, Mathematics, and Computer Science,UvA. 2007-15

B.S. Graaf. *Model-Driven Evolution of Software Architectures*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2007-16

A.H.J. Mathijssen. *Logical Calculi for Reasoning with Binding*. Faculty of Mathematics and Computer Science, TU/e. 2007-17

D. Jarnikov. *QoS framework for Video Streaming in Home Networks*. Faculty of Mathematics and Computer Science, TU/e. 2007-18

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wild-*

cards. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of Mathematics and Computer Science, TU/e. 2008-18

C.M. Gray. *Algorithms for Fat Objects: Decompositions and Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-19

J.R. Calamé. *Testing Reactive Systems with Data - Enumerative Methods and Constraint Solving*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-20

E. Mumford. *Drawing Graphs for Cartographic Applications*. Faculty of Mathematics and Computer Science, TU/e. 2008-21

E.H. de Graaf. *Mining Semi-structured Data, Theoretical and Experimental Aspects of Pattern Evaluation*. Faculty of Mathematics and Natural Sciences, UL. 2008-22

R. Brijder. *Models of Natural Computation: Gene Assembly and Membrane Systems*. Faculty of Mathematics and Natural Sciences, UL. 2008-23

A. Koprowski. *Termination of Rewriting and Its Certification*. Faculty of Mathematics and Computer Science, TU/e. 2008-24

U. Khadim. *Process Algebras for Hybrid Systems: Comparison and Development*. Faculty of Mathematics and Computer Science, TU/e. 2008-25

J. Markovski. *Real and Stochastic Time in Process Algebras for Performance Evaluation*. Faculty of Mathematics and Computer Science, TU/e. 2008-26

H. Kastenberg. *Graph-Based Software Specification and Verification*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-27

I.R. Buhan. *Cryptographic Keys from Noisy Data Theory and Applications*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-28

R.S. Marin-Perianu. *Wireless Sensor Networks in Motion: Clustering Algorithms for Service Discovery and Provisioning*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-29

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for*

Clean. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Fac-

ulty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07

A. Silva. *Kleene Coalgebra*. Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17