Centrum Wiskunde & Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

From coordination to stochastic models of QoS

F. Arbab, T. Chothia, R.D. van der Mei, S. Meng,
Y.J. Moon, C.G. Verhoef

REPORT SEN-E0901 MARCH 2009

Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Science Park 123, 1098 XG Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

From coordination to stochastic models of QoS

ABSTRACT
Reo is a channel-based coordination model whose operational semantics is given by Constraint Automata (CA). Quantitative Constraint Automata extend CA (and hence, Reo) with quantitative models to capture such non-functional aspects of a system's behaviour as delays, costs, resource needs and consumption, that depend on the internal details of the system. However, the performance of a system can crucially depend not only on its internal details, 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. In this paper we propose Quantitative Intentional Automata (QIA), an extension of CA that allow incorporating the influence of a system's environment on its performance. Moreover, we show the translation of QIA into Continuous-Time Markov Chains (CTMCs), which allows us to apply existing CTMC tools and techniques for performance analysis of QIA and Reo circuits.

2000 Mathematics Subject Classification: -
1998 ACM Computing Classification System: F.4.3
Keywords and Phrases: Performance evaluation, Coordination language, Reo, Markov Chains
Note: This work was carried out under project SEN3 and PNA2 - CooPer project.

From Coordination to Stochastic Models of QoS

Farhad Arbab[1], Tom Chothia[2], Rob van der Mei[1,3], Sun Meng[1], YoungJoo Moon[1], and Chrétien Verhoef[1]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[2] School of Computer Science, Univ. of Birmingham, United Kingdom
[3] Vrije Universiteit Amsterdam, The Netherlands
{Farhad.Arbab,R.D.van.der.Mei,M.Sun,Y.J.Moon,C.G.Verhoef}@cwi.nl
T.P.Chothia@cs.bham.ac.uk

Abstract. Reo is a channel-based coordination model whose operational semantics is given by Constraint Automata (CA). Quantitative Constraint Automata extend CA (and hence, Reo) with quantitative models to capture such non-functional aspects of a system's behaviour as delays, costs, resource needs and consumption, that depend on the internal details of the system. However, the performance of a system can crucially depend not only on its internal details, 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. In this paper we propose Quantitative Intentional Automata (QIA), an extension of CA that allow incorporating the influence of a system's environment on its performance. Moreover, we show the translation of QIA into Continuous-Time Markov Chains (CTMCs), which allows us to apply existing CTMC tools and techniques for performance analysis of QIA and Reo circuits.

Keywords: Performance evaluation, Coordination language, Reo, Markov Chains.

1 Introduction

Service-oriented Computing (SOC) provides the means to design and deploy distributed applications that span organization boundaries and computing platforms by exploiting and composing existing services available over a network. 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 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). 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. Yet, the end-to-end QoS of a composed service is often as important as its functional properties in determining its viability in its market.

The coordination language Reo [3, 5] provides a flexible, expressive model for compositional construction of connectors that coordinate service behaviour. CA [6] were introduced to express the operational semantics of Reo. Indeed, CA provides a unified model to capture the semantics of components and services, as well as Reo connectors and their composition. Quantitative Reo and Quantitative Constraint Automata (QCA) [4] extend Reo and CA with the means to describe and combine the QoS aspects of composed systems. 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. While QCA provide a useful model for service selection and composition [19], the performance of a system can crucially depend not only on its internal details, 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 which belong to stochastic aspects. However, such stochastic aspects are not investigated in [19]. Intentional Automata (IA) [14] take into account the influence of the environment as well as internal details of a system by describing the pending status of I/O operators interacting with the environment. A particular class of IA models, called the Reo Automata class, is defined in [14], which provides precise characterization of context-dependent connectors [6].

In this paper we propose QIA, an extension of IA that allows incorporating the influence of a system's environment on its performance. The QIA model extends the semantics of Reo by admitting annotations on its channel ends and the channels to represent the stochastic properties of request arrivals at those ends, data-flows, and data processing and transportation delays through those channels. The resulting *Stochastic Reo* model retains its compositional semantics through QIA: the QIA of a composed system is the product (composition) of the QIA of the individual channels and components/services used in its construction.

The QIA of a system typically has more states than its counterpart CA or QCA, reflecting the (epistemologically) intentional configurations of the system that CA and QCA abstract away. In addition to the synchronization and data constraints of the CA model, the transitions in QIA carry extra information in their labels to convey arrival and firing of data/requests, and their stochastic properties. This information is adequate to allow the analysis of the performance of a system in the context of the stochastic processes in its environment that determine the arrival of data/requests on its ports and their delays. In order to carry out such analysis, in this paper we show the translation of QIA into CTMCs [12], which allows us to apply existing CTMC tools and techniques for performance analysis of QIA and (Stochastic) Reo circuits.

The main contributions in this paper include:

- Stochastic Reo as a compositional model for specifying system behaviour that captures its non-functional (QoS) aspects and takes into account the influence of the environment on its performance,
- QIA as the operational semantics for Stochastic Reo which serves as an intermediate model for generating CTMCs, and

– translation from QIA specifications into CTMC models for performance evaluation.

The Reo and automata editors in the Eclipse Coordination Tools (ECT) [1] have been extended to support Stochastic Reo and QIA, and the automatic derivation of the QIA semantics of Reo circuits. We have implemented the translation of QIA to CTMCs described in this paper as a plug-in within this platform. We have also developed a bridge plug-in that generates the proper input for other stochastic analysis tools like PRISM [2, 18] from our CTMC models to allow performance analysis of Stochastic Reo.

The remainder of this paper is organized as follows. In Section 2, we provide a short overview of Reo, CA, and their quantitative variants. In Section 3 we introduce Stochastic Reo. In Section 4 we define QIA and their composition through product and refinement. In Section 5, we show the translation from QIA into its corresponding CTMC. In Section 6, we show an example of how our CTMC model can be analyzed in PRISM. We review related work in Section 7. Conclusions and future work comprise Section 8. A crucial step in the translation of QIA into a CTMC, as described in Section 5, consists of the sequencing of the delays of synchronized actions that appear on the label of a single transition. We present the algorithm for the sequencing of these delays in Appendix A.

2 Preliminaries

2.1 Reo

Reo is a channel-based exogenous coordination model wherein complex coordinators, called connectors, are compositionally built out of simpler ones. We summarize only the main concepts of Reo and its CA semantics here. Further details about Reo and its semantics can be found in [3, 6].

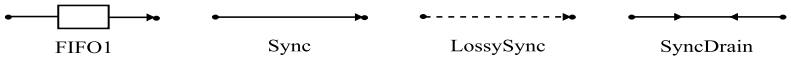

Fig. 1. Some basic Reo channels

Complex connectors in Reo are organised in a network of primitive connectors, called *channels*. Connectors serve to provide the protocol that controls and organises the communication, synchronization and cooperation among the components/services that they interconnect. Each channel has two *channel ends*, and there are two types of channel ends: *source* and *sink*. A source channel end accepts data into its channel, and a sink channel end dispenses data out of its channel. Reo places no restriction on the behaviour of a channel, so it is possible for the ends of a channel to be both sources or both sinks. Figure 1 shows the graphical representation of some simple channel types. A *FIFO1 channel*

(FIFO1) represents an asynchronous channel with one buffer cell. A *synchronous channel* (Sync) has a source and a sink end and no buffer. It accepts a data item through its source end if it can simultaneously dispense it through its sink. A *lossy synchronous channel* (LossySync) is similar to a synchronous channel except that it always accepts all data items through its source end. The data item is transferred if it is possible for the data item to be dispensed through the sink end, otherwise the data item is lost. A *synchronous drain* (SyncDrain) has two source ends and no sink end. It accepts a data item through one of its ends if and only if a data item is also available to be accepted simultaneously through the other end as well.

Connectors are constructed by composing simpler ones via the *join* operation. Channels are joined together in a node which consists of a set of channel ends. Nodes are categorised into source, sink and mixed nodes, depending on whether all channel ends that coincide on a node are source ends, sink ends or a combination of both. In remainder of this paper, we call source and sink nodes boundary nodes since they interact with the environment. Reo allows an open-ended set of user-defined channels with arbitrary behaviour, but it fixes the semantics of the nodes. A source node acts as a synchronous replicator. A sink node acts as a merger. A mixed node combines the behaviour of the the other two nodes and acts as a self-contained "pumping station" that atomically consumes an item out of one of its selected sink ends and replicates it to all of its source ends. Nodes have no memory or buffer and perform their actions atomically. This forces synchrony and exclusion constraints to propagate through the nodes, which causes the channels involved in each synchronous region of a circuit to synchronize their actions in atomic steps.

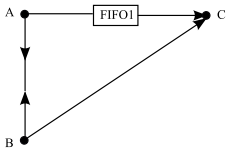

Fig. 2. Ordering circuit

For example, the connector shown in Figure 2 is an alternator that imposes an ordering on the flow of 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 success of 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 Constraint Automata

CA were introduced [6] as a formalism to capture the operational semantics of Reo, based on timed data streams, which also constitute the foundation of the coalgebraic semantics of Reo [5].

We assume a finite set $\mathcal{N}$ 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, $A, B \in \mathcal{N}$, d_A is a symbol for the observed data item at node A and $d \in Data$. $DC(N)$ denotes the set of data constraints that at most refer to the observed data items d_A at node $A \in N$. 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 $\mathscr{A} = (S, S_0, \mathcal{N}, \rightarrow)$ where S is a set of states, also called configurations, $S_0 \subseteq S$ is the set of its initial states, $\mathcal{N}$ is a finite set of nodes, $\rightarrow$ is a finite subset of $S \times \{N\} \times DC(N) \times S$ with $N \in 2^{\mathcal{N}}$, 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.

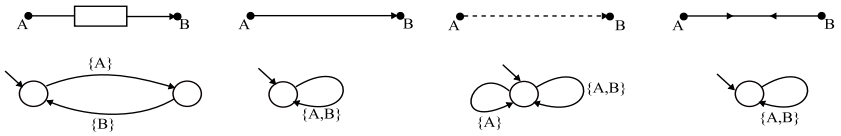

Fig. 3. Constraint Automata for basic Reo channels

Figure 3 shows the CA for the primitive Reo channels in Figure 1. In this figure and the remainder of this paper, for simplicity, we assume the data constraints of all transitions are `true` (which simply imposes no constraints on the contents of the data-flows) and omit them to avoid clutter. For proper full treatment of data constraints in CA, see [6].

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

– If $s_1 \xrightarrow{N_1,g_1}_1 s_1'$, $s_2 \xrightarrow{N_2,g_2}_2 s_2'$, $N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1$ and $g_1 \wedge g_2$ is satisfiable, 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 $s_1 \xrightarrow{N_1,g_1}_1 s_1'$, where $N_1 \cap \mathscr{N}_2 = \emptyset$ then $\langle s_1, s_2\rangle \xrightarrow{N_1,g_1} \langle s_1', s_2\rangle$.
– If $s_2 \xrightarrow{N_2,g_2}_2 s_2'$, where $N_2 \cap \mathscr{N}_1 = \emptyset$ then $\langle s_1, s_2\rangle \xrightarrow{N_2,g_2} \langle s_1, s_2'\rangle$.

2.3 Quantitative Constraint Automata and Quantitative Reo

Quantitative Reo and QCA are extensions of Reo and CA, respectively, with quantitative aspects by Q-algebra [13] and form the basis for compositional specification and reasoning on QoS issues for connectors. A Q-algebra is an algebraic structure $R = (C, \oplus, \otimes, \odot, \mathbf{0}, \mathbf{1})$ such that $R_{\otimes} = (C, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ and $R_{\odot} = (C, \oplus, \odot, \mathbf{0}, \mathbf{1})$ are both constraint semirings [9, 20]. C is a set of QoS values and is called the *domain* of R. The operation $\oplus$ induces a partial order $\leq$ on C, which is defined by $c \leq c'$ iff $c \oplus c' = c'$. The other two operators $\otimes$ and $\odot$ can combine QoS values when they occur, respectively, sequentially and concurrently. In these constraint semirings, $\mathbf{0}$ is the identity for $\oplus$, and $\mathbf{1}$ is the identity for $\otimes$ and $\odot$.

A QCA is a tuple $\mathscr{A} = (S, S_0, \mathscr{N}, R, \longrightarrow)$ where S is a set of states, $S_0 \subseteq S$ is the set of its initial states, $\mathscr{N}$ is a finite set of nodes, $R = (C, \oplus, \otimes, \odot, \mathbf{0}, \mathbf{1})$ is a Q-algebra with domain C of QoS values, $\longrightarrow$ is a finite subset of $S \times \{N\} \times DC(N) \times C \times S$ with $N \in 2^{\mathscr{N}}$.

The synchronous behaviour of each Quantitative Reo channel has a certain QoS value in its label, which is in the domain C of a Q-algebra. The following types of QoS for the basic channels in Reo are considered: t (execution time for data transmission), c (allocated memory cost for the message transmission) and p (reliability represented by the probability of successful transmission). The corresponding Q-algebras are given as:

– execution time: $(\mathbb{R}_+ \cup \{\infty\}, max, +, max, 0, 0)$
– memory cost: $(\mathbb{N}_+ \cup \{\infty\}, max, +, +, 0, 0)$
– reliability: $([0, 1], min, \times, \times, 1, 1)$

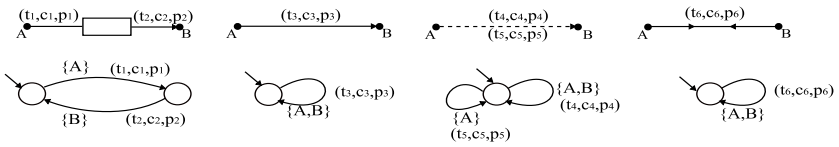

Fig. 4. Quantitative Constraint Automata for basic Quantitative Reo channels

Quantitative Reo keeps a compositional framework with the same *join* operation of Reo, and QCA, as operational semantics of Quantitative Reo, provide a corresponding composition method (product). Two QCA $\mathscr{A}$ and $\mathscr{B}$ with

the same Q-algebra turn into a new QCA by the product operation. For $\mathscr{A} = (S_1, S_{0,1}, \mathscr{N}_1, R, \longrightarrow_1)$ and $\mathscr{B} = (S_2, S_{0,2}, \mathscr{N}_2, R, \longrightarrow_2)$, their product is defined as

$$\mathscr{A} \bowtie \mathscr{B} = (S_1 \times S_2, S_{0,1} \times S_{0,2}, \mathscr{N}_1 \times \mathscr{N}_2, R, \longrightarrow)$$

where $\longrightarrow$ is given by the following rules:

- If $s_1 \xrightarrow{N_1,g_1,c_1}_1 s_1'$, $s_2 \xrightarrow{N_2,g_2,c_2}_2 s_2'$, $N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1 \neq \emptyset$ and $g_1 \wedge g_2$ is satisfiable, then $\langle s_1, s_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2, c_1 \oplus c_2} \langle s_1', s_2' \rangle$.
- If $s_1 \xrightarrow{N,g,c}_1 s_1'$, where $N \cap \mathscr{N}_2 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N,g,c} \langle s_1', s_2 \rangle$.
- If $s_2 \xrightarrow{N,g,c}_2 s_2'$, where $N \cap \mathscr{N}_1 = \emptyset$ then $\langle s_1, s_2 \rangle \xrightarrow{N,g,c} \langle s_1, s_2' \rangle$.

The quantitative version of the circuit in Figure 2 and its corresponding QCA are shown in Figure 5. The relevant QoS values are given by the tuple (t_i, c_i, p_i) that represents the QoS values for the basic channels, as specified in Figure 4.

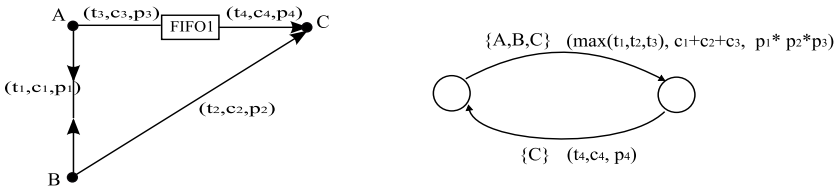

Fig. 5. Ordering circuit in Quantitative Reo and its QCA

3 Stochastic Reo

Stochastic Reo is an extension of Reo annotated with stochastic properties, such as processing delays on channels and arrival rates of data/requests at the channel ends, allowing general distributions. Figure 6 shows the primitive channels of Stochastic Reo that correspond to the primitives of Reo in Figure 1. In this figure and the remainder of this paper, for simplicity, we delete node names, but these names can be inferred from the names of their respective arrival processes: for instance, 'dA' means an arrival process at node 'A'. The labels annotating Stochastic Reo channels can be separated into the following two categories:

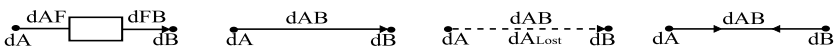

Fig. 6. Basic Stochastic Reo channels

- channel delays
  To model the stochastic behaviour of Reo channels, we assume every Reo channel has one or more associated delays represented by their corresponding random variables. Such a delay represents how long it takes for a channel to deliver or throw away its data. For instance, a LossySync has two associated variables 'dAB' and 'dA_{Lost}' for stochastic delays of, respectively, successful data-flow through the nodes 'A' and 'B' and losing data at node 'A' when a read request is absent at node 'B'. In a FIFO1 'dAF' means the delay for data-flow from its source 'A' into the buffer, and 'dFB' for sending the data from the buffer to the sink 'B'. Similarly, the random variable of a Sync (and a SyncDrain) indicates the delay for data-flow from its source node 'A' to its sink node 'B' (and losing data at both ends, respectively).
- arrivals at nodes
  I/O operations are performed on the source and sink nodes of a Reo circuit through which it interacts with its environment. We assume the time between consecutive arrivals of read and write requests at the sink and source nodes of Reo connectors depends on their associated stochastic processes. For instance, 'dA' and 'dB' in Figure 6 represent the associated arrival processes at nodes 'A' and 'B'. Furthermore, at most one request at each boundary node can wait for acceptance. 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 joining nodes as 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 their composition. Nodes are categorized into mixed, source, and sink nodes. Boundary nodes receive data/requests from the environment, after that mixed nodes are synchronized for data-flow and then merely pump data in the circuit, i.e., mixed nodes do not interact with the environment. This account shows the causality of the events happening in the circuit, such as arrivals of data/requests at its boundary nodes, synchronizing its mixed nodes, and occurrences of data-flow, sequentially. Besides, we assume that pumping data by mixed nodes is an immediate action and therefore mixed nodes have no associated stochastic variables [4]. Boundary nodes have their corresponding stochastic arrival processes, yet when they are combined into mixed nodes by a *join* operation, they lose their stochastic variables. As mentioned in Section 2, a source node and a sink node act as a replicator and a non-deterministic merger, respectively, and each activity, such as selecting a sink end or replicating data to its source ends, has its own stochastic property. In order to describe stochastic delays of a channel explicitly, we name the delay by the combination of a pair of (source, sink) nodes and the buffer of the channel. For example, the stochastic

[4] This assumption is not a real restriction. A mixed node with delay can be modelled by replacing this mixed node with a Sync channel with the delay. Moreover, according to the required level of specification detail, each input and output of the mixed node can be modelled by adding corresponding Sync channels with their stochastic values.

property 'dAF' of FIFO1 in Figure 6 stands for the data-flow from the source end 'A' into the buffer of the FIFO1. However, in cases where, for instance, a source node (as a replicator) A is connected to two different FIFO1s (buffers), then the corresponding stochastic processes have the same name, e.g., dAF. To avoid such an ambiguous situation, we rename the stochastic processes by adding a number after its node name like dA_1F and dA_2F when the node has more than one outgoing channel or one incoming channel. As an example of composed Stochastic Reo, Figure 7 shows the ordering circuit with the annotation of its stochastic variables.

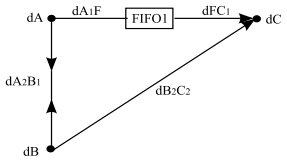

Fig. 7. Ordering circuit in Stochastic Reo

4 Quantitative Intentional Automata

In this section we introduce the notion of QIA which is an extension of CA and provides operational semantics for Stochastic Reo. Whereas CA transitions describe system configuration changes, QIA transitions describe the changes of not only the system configuration but also the status of its pending I/O operations. In CA, configurations are shown as states, and processes causing state changes are shown in transition labels as a set of nodes where data are observed. Similarly, in QIA, system configurations and the status of pending I/O operations are shown as states. Data-flow or firing through nodes causes changes in the system configuration, and arrivals of data/requests at the nodes or synchronization of nodes changes the status of pending data/requests. These two different types of changes are shown in the transition labels by two different sets of nodes. Moreover, QIA transitions carry their relevant stochastic properties in their labels. We use such QIA as an intermediate model for translation Stochastic Reo into a homogeneous CTMC.

Definition 1. ***QIA***
A Quantitative Intentional Automaton is a tuple $\mathscr{A}=(S, S_0, \mathscr{N}, \rightarrow)$ *where*

- $S \subseteq L \times 2^{\mathscr{N}}$ *is a finite set of states.*
  - L *is a set of system configurations.*
  - $R \in 2^{\mathscr{N}}$ *is a set of pending nodes, that describes the pending status in the current state.*
- $S_0 \subseteq S$ *is a set of initial states.*

– $\mathcal{N}$ *is a finite set of nodes.*
– $\rightarrow \subseteq \bigcup_{M,N \subseteq \mathcal{N}} S \times \{M\} \times \{N\} \times DC(N) \times 2^{DI} \times S$ *is the transition relation.*
  • $DI \subseteq 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathbb{R}^{+}$.

A transition in a QIA is represented as $\langle l, R \rangle \xrightarrow{M,N,g,D} \langle l', R' \rangle$ where M is the set of nodes that exchange data or synchronize for data-flow through the transition, N is the set of nodes to be released by the firing of the transition, and $D \subseteq DI$ is the set of delay information tuples (I, O, r) where I and O are sets of, respectively, source (input) and sink (output) nodes, and r indicates the stochastic delay rate for the data-flow from I to O or the arrival rate of data/request from the environment at nodes in $I \cup O$. Furthermore, let $D = \{(I_j, O_j, r_j) | 1 \leq j \leq n\}$, then $\bigcup_{1 \leq j \leq n} (I_j \cup O_j) = N \cup M$.

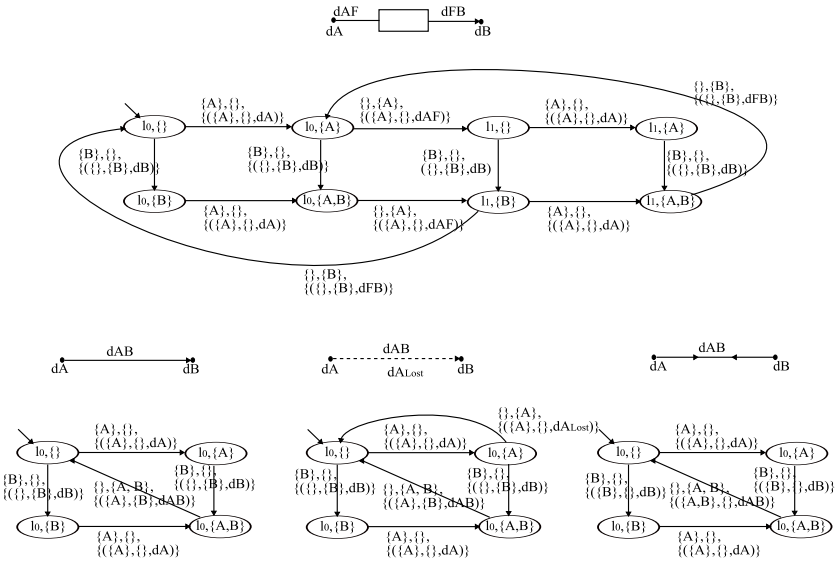

Fig. 8. QIA for each channel of Figure 6

Definition 1 is not enough to specify the system behaviour correctly. The causality of activities, such as arrivals of data/requests and firing, is not embraced in this definition. Moreover, in continuous time scale, all events occur one by one: only a single event, such as one request arrival or a single firing in a set of synchronized atomic firings, is taken into consideration at a time. Taking these features into account, we explore additional conditions that can be placed on QIA, and define the well-formedness condition of QIA. Hence, the QIA corresponding to the primitive Stochastic Reo channels are represented like Figure 8.

Definition 2. ***QIA Well-formedness***
A QIA $\mathscr{A} = (S, S_0, \mathscr{N}, \rightarrow)$ *is well-formed if* $\forall \langle l, R\rangle \xrightarrow{M,N,g,D} \langle l', R'\rangle \in \rightarrow$ *all following conditions are satisfied:*

1. $N \subseteq R \cup M$
2. $M \cap R = \emptyset$
3. $(R \cup M) \setminus N = R'$
4. $((N \neq \emptyset \wedge M \subseteq N) \ \vee \ (N = \emptyset \wedge M \neq \emptyset \rightarrow |M| = 1))$

According to the assumption on synchronization and the causality of the activities in a Reo circuit, the well-formedness conditions are interpreted as follows:

1. A data-flow can occur only when all necessary nodes are ready to transfer the data.
2. A node is blocked when the node is suspended and occupied by another data item.
3. A firing releases the nodes involved in the firing.
4. A firing and a data arrival are mutually exclusive:
   - A firing and its relevant synchronization happen simultaneously, i.e., after the synchronization of nodes, those nodes are immediately released by their corresponding firing.
   - Only one single data/request arrives at a time.

QIA provide a compositional framework for Stochastic Reo. Hence, the QIA corresponding to a circuit is obtained by the product of the QIA of all primitive channels that constitute the circuit, for example, the QIA model in Figure 9 corresponding to the ordering in Figure 7 is obtained by composing the QIA of its primitive channels. The mixed nodes from such a composition are obtained by a function $newMixed : \mathscr{A} \times \mathscr{A} \rightarrow 2^{\mathscr{N}}$. As mentioned above, the synchronization of its relevant mixed nodes has no associated stochastic property and occurs simultaneously with its corresponding firing. Hence, the mixed nodes involved in a certain firing must be considered as undergoing an atomic change through the firing, and the stochastic properties of the mixed nodes are deleted in the composed result. To represent such simultaneous occurrence, the relevant mixed nodes must be collected and shown in the label of their firing together.

Definition 3. ***Synchronization of mixed nodes***
For two QIA $\mathscr{A} = (S_1, S_{0,1}, \mathscr{N}_1, \rightarrow_1)$, $\mathscr{B} = (S_2, S_{0,2}, \mathscr{N}_2, \rightarrow_2)$, *the firing with synchronization of its mixed nodes is defined as* $s \xrightarrow{C \cup M,N,g,D}{}^{*}_{i} \ s'$ *for* $C \subseteq newMixed(\mathscr{A}, \mathscr{B})$ *such that there are consecutive transitions with the mixed nodes until its firing appears*

$$s \xrightarrow{\{B_i\},\emptyset,true,D_i}{}_{i} \ s_1 \xrightarrow{\{B_k\},\emptyset,true,D_k}{}_{i} \cdots \xrightarrow{M,N,g,D}{}_{i} \ s'$$

where $B_i, B_k, \cdots \in C \wedge C \subseteq N$ *for* $i = 1, \ 2$.

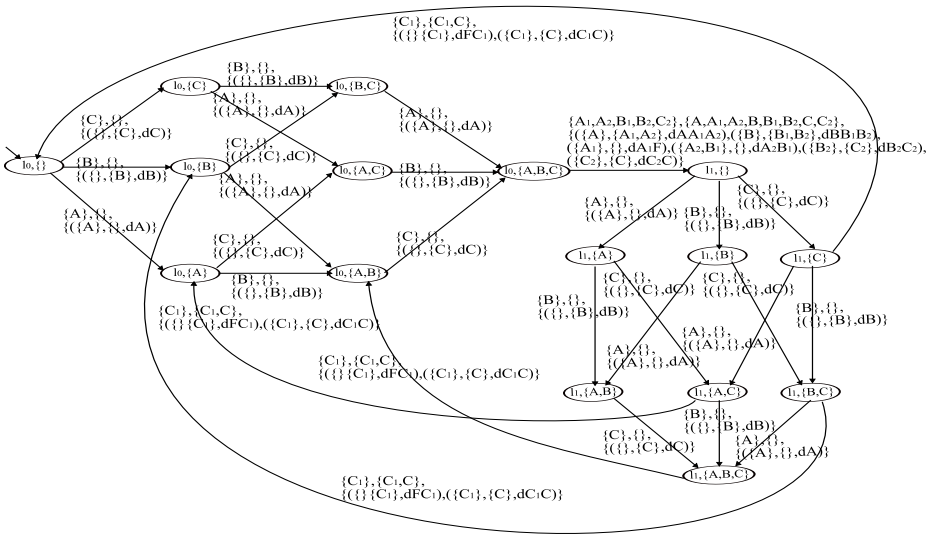

Fig. 9. Corresponding QIA to the ordering circuit in Figure 7

Definition 4. ***QIA Product***
Given two QIA $\mathscr{A} = (S_1, S_{1,0}, \mathscr{N}_1, \longrightarrow_1)$ *and* $\mathscr{B} = (S_2, S_{2,0}, \mathscr{N}_2, \longrightarrow_2)$*, their product is defined as* $\mathscr{A} \bowtie \mathscr{B} = (S_1 \times S_2, S_{1,0} \times S_{2,0}, \mathscr{N}_1 \cup \mathscr{N}_2, \longrightarrow)$ *where* $\longrightarrow$ *is given by the following set of rules:*

1. *for every* $\langle l_1, R_1\rangle \xrightarrow{M_1,N_1,g_1,D_1}_1 \langle l_1', R_1'\rangle$ *and* $\langle l_2, R_2\rangle \xrightarrow{M_2,N_2,g_2,D_2}_2 \langle l_2', R_2'\rangle$
   - *if* $M_1 \cap \mathscr{N}_2 = \emptyset \ \wedge \ N_1 \cap \mathscr{N}_2 = \emptyset$*, then*
     $\langle (l_1, l_2), R_1 \cup R_2\rangle \xrightarrow{M_1,N_1,g_1,D_1} \langle (l_1', l_2), R_1' \cup R_2\rangle$.
   - *if* $M_2 \cap \mathscr{N}_1 = \emptyset \ \wedge \ N_2 \cap \mathscr{N}_1 = \emptyset$*, then*
     $\langle (l_1, l_2), R_1 \cup R_2\rangle \xrightarrow{M_2,N_2,g_2,D_2} \langle (l_1, l_2'), R_1 \cup R_2'\rangle$.
2. *for every* $\langle l_1, R_1\rangle \xrightarrow{C_1 \cup M_1,N_1,g_1,D_1}_1{}^{*} \langle l_1', R_1'\rangle$ *and* $\langle l_2, R_2\rangle \xrightarrow{C_2 \cup M_2,N_2,g_2,D_2}_2{}^{*} \langle l_2', R_2'\rangle$ *with* $\forall C_1, C_2 \subseteq newMixed(\mathscr{A}, \mathscr{B})$
   - *if* $N_1 \neq \emptyset \neq N_2 \wedge N_1 \cap \mathscr{N}_2 = N_2 \cap \mathscr{N}_1$*, then*
     $\langle (l_1, l_2), R_1 \cup R_2\rangle \xrightarrow{C_1 \cup C_2 \cup M_1 \cup M_2, N_1 \cup N_2, g_1 \wedge g_2, D_1 \cup D_2} \langle (l_1', l_2'), R_1' \cup R_2'\rangle$

The product of two QIA generates all possible compositions of transitions, though some of the generated transitions are irrelevant. For instance, in the product of the automata LossySync AB and Sync BC (cf. the first automaton in Figure 10), the state $\langle l_0, \{A, C\}\rangle$ has two possible firing transitions: one for losing the data at A and the other for the data-flow from A to C via the mixed node B. However, this state says that some requests are pending on nodes A and C, therefore, only data-flow between A and C can occur in the next step. We

define a notion of refinement in the following that can be used to delete such unnecessary transitions from a product.

Definition 5. ***QIA Refinement***
For a QIA $\mathscr{A} = (S, S_0, \mathscr{N}, \rightarrow)$*, the refinement of* $\mathscr{A}$*, Ref*$(\mathscr{A})$*, is defined as* $(S, S_0, \mathscr{N}, \rightarrow')$ *with* $\rightarrow' = \rightarrow \setminus T$*, where* T *is defined as*

$$T = \{s \xrightarrow{M,N,g,D} s' | \exists s \xrightarrow{M_1,N_1,g_1,D_1} s_1 \in \rightarrow \ s.t.\ P\}$$

, P *is the conjunction of the following conditions:*

1. $g \wedge g_1$ *is satisfiable.*
2. $M \subseteq M_1 \wedge \emptyset \neq N \subseteq N_1$
3. $N \setminus M \subseteq N_1 \setminus M_1$
4. $\nexists s \xrightarrow{M_2,N_2,g_2,D_2} s_2$ *s.t.* $(N_1 \setminus M_1) \setminus (N \setminus M) \subseteq (N_2 \setminus M_2) \neq \emptyset$ $\wedge\, (N \setminus M) \cap (N_2 \setminus M_2) = \emptyset$

Intuitively, conditions 1, 2, and 3 in Definition 5 guarantee that a transition with less pending and firing nodes than another transition from the same source state will be removed, and condition 4 ensures that transitions with independent firings of pending nodes are kept. Now we apply such refinement to the product of LossySync AB and Sync BC in Figure 10. The transition from $\langle l_0, \{A, C\}\rangle$ with losing data at node A has less pending and firing nodes than the other transition from the same source state with a data-flow from A to C via the mixed node B, and also there is no independent firing through the node C ($\in \{A, C\} \setminus \{A\}$), which means the firing of pending requests at nodes A and C are dependent. Hence the transition of losing data at node A with pending data at nodes A and C will be removed from the product result. A QIA model specifies the system behaviour with considering the influence of the environment, and provides a compositional framework, i.e., the QIA version of a complex connector is obtained by applying the product to primitive channels which comprise the connector. However, the context-dependency [6] of the connector is ignored in the product, hence we apply the refinement to the product result to retain the dependency.

5 From QIA to CTMC

A CTMC is a stochastic discrete-state process, often used to model and analyse system performance. A CTMC process is defined as $\{X(t)|t \geq 0\}$. $X(t) \in S$ denotes the state in 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:

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

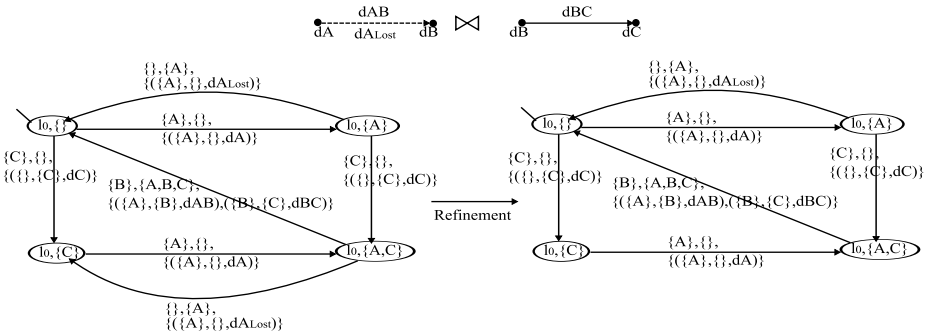

Fig. 10. QIA product of LossySync AB and Sync BC

In this section we propose an approach for translating QIA into CTMC to carry out performance evaluation. Through this translation, we can specify a system in Stochastic Reo, provide its operational semantics with QIA, and then evaluate its performance via the CTMC derived from its QIA. A Markov Chain (MC) is not compositional and it is difficult to obtain a MC model for a complex system. In our approach, QIA provide a compositional framework for the specification, and the corresponding CTMC model even for a complex system can be subsequently derived from the composed QIA by translation.

In a CTMC, all the stochastic variables on each of its transitions must be exponentially distributed. Hence every stochastic event occurs one by one. In QIA, each transition corresponds to an atomic behaviour, i.e., an arrival of a single data item or synchronized multiple events (especially firings). Such synchronized multiple events happen together, and this is where QIA and CTMC differ. Therefore, for our translation, we need to spread and divide such synchronized multi-event firings into micro-step single-event transitions.

Principle 1 *A data-flow in a channel takes place from its input node to its output node.*

Principle 2 *Mixed nodes send and receive data instantaneously.*

Recall that a D in a QIA transition label is a set of delay information tuples (I, O, r) in $2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathbb{R}^+$. Each such tuple describes a data-flow from its input nodes in I to its output nodes in O with the stochastic delay r. The above principles impose a causality-based sequence on the events in D. For example, in $D = \{(\{A\}, \{B\}, dAB), (\{B\}, \{C\}, dBC)\}$, the two tuples directly indicate that data-flow occurs from A to B, with delay dAB, and from B to C, with delay dBC. Moreover, since B appears in the output set of one tuple and the input set of the other, B must be a mixed node, which implies that the data-flow between A and B occurs before data-flow between B and C. From such causality-based sequences we derive a delay-sequence d for each firing, capturing the sequential or parallel properties of each element in its D. The concrete algorithm of extracting such a

delay-sequence from D is given in Appendix A. Syntactically, a delay sequence is:

$$d ::= \epsilon \mid delay \mid d; d \mid d|d \tag{1}$$

where ϵ is the empty sequence, $delay \in D$, '$d; d$' is the sequential composition of delays, and '$d|d$' is the parallel composition of delays. We also use parentheses '(' and ')' to indicate the highest priority for grouping, where more deeply nested groups have higher precedence. The empty sequence ϵ is an identity element for the ';' and '|' operations, i.e., $\epsilon|d = d = d|\epsilon$, $\epsilon; d = d = d; \epsilon$, and '|' is commutative, associative, and idempotent, i.e., $A|B = B|A$, $(A|B)|C = A|(B|C)$, $A|A = A$.

In the translation from QIA to CTMC, a single *delay* causes no change to a transition. A synchronized multi-event firing in the '$d; d$' or '$d|d$' form is divided into micro-step single-event transitions by, respectively, enumerating each *delay* element in a sequential delay-sequence and considering the interleaving of all single delays in a parallel delay-sequence. For example, in Figure 11, the consecutive transitions from state Y to the initial state via state X correspond to the result of splitting the synchronized multi-event firing from state $\langle l_1, \{C\}\rangle$ to state $\langle l_0, \{\}\rangle$ in Figure 9, into micro-step single-event transitions. Similarly, the second and third diamond-shaped clusters of transitions (G_2 and G_3 in Figure 11, respectively) represent the result of splitting the synchronized multi-event firing from state $\langle l_0, \{A, B, C\}\rangle$ to state $\langle l_1, \{\}\rangle$ in Figure 9. This splitting is applied until no multi-event firing remains. Consequently, every transition in the result corresponds to a single event with its stochastic property.

In QIA, a synchronized multi-event firing is considered atomic, hence other events cannot interfere with it. However, as we split multiple synchronized events, we cannot guarantee their atomicity any more. A transition having the same source state as another transition that involves a synchronized multi-event firing represents an event that can preempt the sequence of transitions that result from splitting the multi-event firing. For example, state $\langle l_1, \{C\}\rangle$ of the QIA in Figure 9 is connected to the initial state by the transition labeled with the synchronized multi-event firing $\{(\{\}, \{C_1\}, dFC_1), (\{C_1\}, \{C\}, dC_1C)\}$, and there are two other transitions of data arrivals at nodes A and B out of $\langle l_1, \{C\}\rangle$. These arrivals are preemptible events for the sequence of micro-step transitions that result from the splitting of this synchronized multi-event firing. State $\langle l_1, \{C\}\rangle$ in Figure 9 corresponds to state Y in Figure 11, and hence its preemptible events are added as extra transitions tracing the split single-event transitions, like the transitions from state X labeled with data arrivals at nodes A and B.

6 Stochastic Analysis

Our QIA to CTMC translation tool has been incorporated as a plug-in in our ECT environment and can generate input files for analysis in other existing tools like PRISM. For instance, PRISM can be used on a CTMC for the analysis of its steady-state distributions to gain insight not only into the essential states of a system but also about principal performance measures such as delays, throughput, bottlenecks, and blocking probabilities. Moreover, by adjusting values of

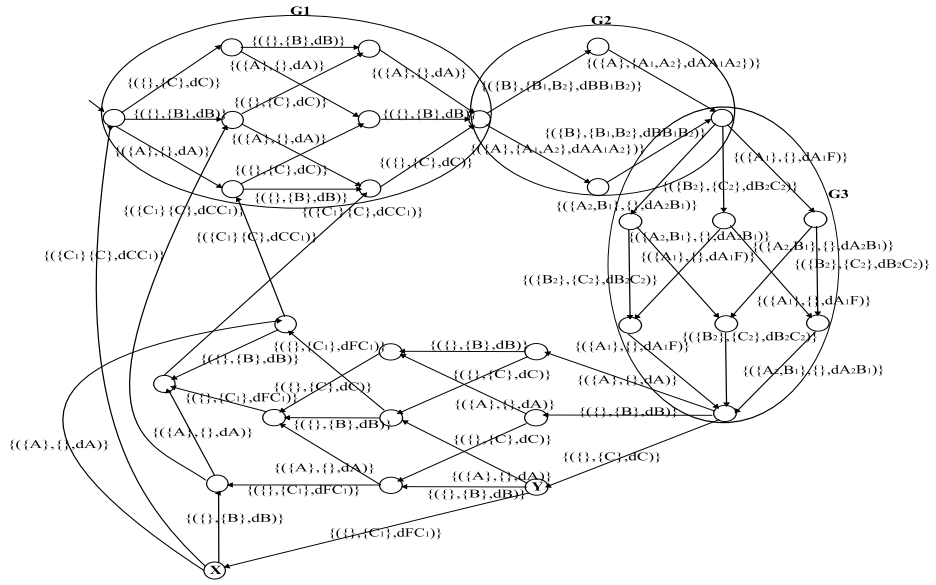

Fig. 11. Derived MC from ordering circuit

some stochastic variables, we can perform sensitivity analysis on the system. To illustrate the relevance of our method and the usefulness of our tool, we briefly consider an example, and show the results of a simple analysis of our generated model.

Consider the ordering circuit in Figure 7. As mentioned in Section 3, a boundary node or a buffer in a connector is blocked when it is occupied by another pending request. Figure 12 shows the blocking probabilities of nodes A, B, and C and the buffer of the FIFO1 channel when the arrival rate dA at node A increases (i.e., the arrival frequency of requests at node A is increasing) while the arrival rates of requests at nodes B and C, and all processing delays are fixed to 1. A blocking probability is calculated by accumulating steady-state probabilities of its corresponding states, i.e., the blocking probability of node A is obtained as the summation of the steady-state probabilities of all the states whose configurations show that node A is blocked. The arrival rate λ is distributed exponentially and its mean is $1/\lambda$ time units. Hence, as dA increases, the blocking probability of node A also increases, which increases the probability that the FIFO1 is full since the request at node A is delivered to its buffer. A request at node B is consumed together with a request at node A, and a request at node C is consumed together with another at node B or the FIFO1 buffer, alternately. Hence as requests arrive at node A more frequently, the probabilities that nodes B and C are released increase (i.e., their blocking probabilities decrease). Because node C is released by both node B and the buffer, its blocking probability decreases more quickly

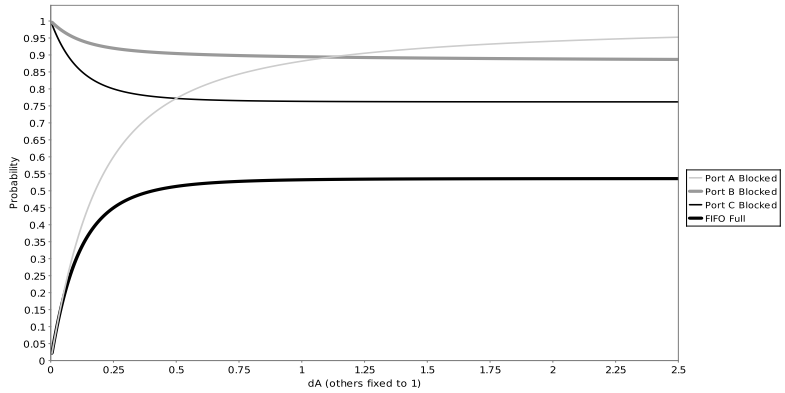

Fig. 12. Blocking probabilities of ordering circuit in Figure 7

than that of node B. However all probabilities reach a certain threshold after a while, because of the fixed stochastic values.

7 Related work

The research in formal specification of a system with quantitative aspects encompasses many developments such as Stochastic Process Algebras (SPAs) [11], Stochastic Automata Networks (SANs) [15, 24, 22], Stochastic Petri nets (SPNs) [16, 23]. SPA is a model for both qualitative and quantitative specification and analysis with a compositional and hierarchical framework, and has algebraic laws (or so called static laws) and expansion laws which express a parallel composition in terms of its operators. In SPA the interpretation of the parallel composition is a vexed one, which allows various interpretations such as Performance Evaluation Process Algebra (PEPA) [17], Extended Markovian Process Algebra (EMPA) [7, 8]. SPA describes '*how*' each process behaves, but (Stochastic) Reo directly describes '*what*' communication protocols connect and coordinate the processes in a system, in terms of primitive channels and their composition. Therefore, QIA and (Stochastic) Reo explicitly model the pure coordination and communication protocols including the impact of real communication networks on software systems and their interactions. Compared to SPA, our approach more naturally leads to a formulation using queueing models like SPNs.

SPN is a directed, weighted, and bipartite graph with an associate exponentially distributed firing delay on each transition. SPN is widely used for modelling concurrency, synchronization, and precedence, and is conducive to both top-down and bottom-up modelling. Stochastic Reo shares the same properties with SPN and natively supports composition of synchrony and exclusion together with asynchrony, which is not possible in Petri nets. The topology of connectors in (Stochastic) Reo is inherently dynamic, and it accommodates mobility.

Moreover, (Stochastic) Reo supports a liberal notion of channels and is more general than data-flow models and Petri nets, which can be viewed as specialized channel-based models that incorporate certain specific primitive coordination constructs.

SAN consists of a couple of stochastic automata which act independently. In other words, it supports a modular approach. Hence the state of SAN at time t is expressed by the states of each automaton at time t. The concept of a collection of individual automata helps modelling distributed and parallel systems more easily. SAN might be viewed as SPA. However, SPA is concerned with structural properties such as compositionality and equivalence, and mapping of the specification onto Markov Chains for the computation of performance measures. On the other hand, the original purpose of SAN is to provide an efficient and convenient methodology for computing performance measures rather than a means of deriving algebraic properties of complex systems. The interactions in SAN are rather limited to patterns like synchronizing events or operating at different rates. Compared with the SAN approach, the expressiveness of (Stochastic) Reo makes it possible to model different interaction patterns involving both asynchronous and synchronous communications.

In general, the reachability graphs or MCs derived from the above formalisms have a large state space that prohibits the computation of a solution. In case of SAN the state space explosion problem is relieved by a modular approach to modelling and efficient numerical treatment of the generator matrix [22]. Ameliorating the state explosion problem in other models is still ongoing research, and we are also concerned with the efficient solution technique for the MC derived from Stochastic Reo. The compositional nature of Reo encourages a modular design approach that can, as in the case of SAN, help the state explosion problem. Moreover, the Markov Chains generated by our translation method from Stochastic Reo and QIA consistently show certain interesting structural properties that can be exploited for modular solution and composition through re-scaling. We are currently investigating these alternative solution techniques.

QCA and Quantitative Reo deal with various kinds of non-functional aspects of the system's behaviour and provide a computational and reasoning model with Q-algebra, as used for selection and composition of services/components [19]. The QoS aspects concerned in QCA, such as delays, costs, and resource, depend on the internal details of the system, and accordingly ignore the influence of the environment. However, the performance of a system depends not only on its internal details but also on how it is used in its environment like the the frequency and distribution of request arrivals, and QCA do not concern these stochastic aspects. QIA and Stochastic Reo cover both the internal details of a system as well as the influence of the environment, and hence support a comprehensive approach for specification and performance analysis of a system.

8 Conclusion and Future work

In this paper, we propose Stochastic Reo and QIA by adding quantitative support in our coordination model and its operational semantics to account for the influence of the environment on the performance of a coordination protocol (i.e., connector). We provide an approach to translate QIA into CTMC for performance analysis when the performance properties are distributed exponentially in QIA. The Reo and automata editors in the Eclipse Coordination Tools [1] have been extended to support Stochastic Reo and QIA, and the automatic derivation of the QIA semantics of Stochastic Reo circuits. We have implemented the translation from QIA to CTMC, and also the generation of the input files for PRISM, and have incorporated them within this platform.

As future work, we want to consider non-exponential distributions, for example, by considering phase-type distributions [21] as an approximation of non-exponential distributions or using (generalized) semi-Markov processes [25] as a target model of the translation. We have found that the CTMCs that are derived from Reo circuits frequently contain a pattern of essentially feed-forward clusters of states. We are investigating methods to exploit these patterns for more efficient compositional solution techniques. A recent semantic model for Reo captures the context-dependent behaviour of Reo connectors in a very small automata model [10]. We expect that using these automata as a basis can provide a more abstract model with significantly smaller numbers of states and transitions compared to the QIA. This can make translating a Reo connector to an MC considerably more efficient.

Acknowledgment

The work reported in this paper is supported by a grant from the GLANCE funding program of NWO, through project CooPer (600.643.000.05N12); project SYANCO (DN 62-613) funded by the DFG-NWO bilateral program; and by the European IST-33826 STREP project CREDO. The authors are indebted to the members of SEN3 for helpful discussions on various aspects of this work. Specifically, the authors are thankful for the assistance of Christian Koehler and Ziyan Maraikar for their cooperation in the implementation of the QIA tools.

References

1. Eclipse Coordination Tools. http://reo.project.cwi.nl/.
2. Probabilistic model checker. http://www.prismmodelchecker.org/.
3. F. Arbab. Reo: a channel-based coordination model for component composition. *MSCS*, 14(3):329–366, 2004.
4. F. Arbab, T. Chothia, S. Meng, and Y.-J. Moon. Component Connectors with QoS Guarantees. In *COORDINATION*, pages 286–304, 2007.
5. F. Arbab and J. J. M. M. Rutten. A Coinductive Calculus of Component Connectors. In *WADT*, pages 34–55, 2002.

6. C. Baier, M. Sirjani, F. Arbab, and J. J. M. M. Rutten. Modeling component connectors in Reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.
7. M. Bernardo and R. Gorrieri. Extended Markovian Process Algebra. In *CONCUR*, pages 315–330, 1996.
8. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theor. Comput. Sci.*, 202(1-2):1–54, 1998.
9. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
10. M. Bonsangue, D. Clarke, and A. Silva. Automata for context-dependent connectors. Submitted.
11. M. Calzarossa and S. Tucci, editors. *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, volume 2459 of *LNCS*. Springer, 2002.
12. W.-K. Ching and M. K. Ng. *Markov Chains: Models, Algorithms and Applications*. Springer, 2005.
13. T. Chothia and J. Kleijn. Q-Automata: Modelling the Resource Usage of Concurrent Components. *Electr. Notes Theor. Comput. Sci.*, 175(2):153–167, 2007.
14. D. Costa. *Formal Models for Context Dependent Connectors for Distributed Software Components and Services*. Phd thesis, 2009.
15. P. Fernandes, B. Plateau, and W. J. Stewart. Efficient Descriptor-Vector Multiplications in Stochastic Automata Networks. *J. ACM*, 45(3):381–414, 1998.
16. B. R. Haverkort, R. Marie, G. Rubino, and K. S. Trivedi, editors. *Performability Modelling: Techniques and Tools*. Wiley, 2001.
17. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
18. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Symbolic Model Checker. In *Computer Performance Evaluation/TOOLS*, pages 200–204, 2002.
19. S. Meng and F. Arbab. QoS-Driven Service Selection and Composition. In *ACSD*, pages 160–169. IEEE Computer Society, 2008.
20. R. D. Nicola, G. L. Ferrari, U. Montanari, R. Pugliese, and E. Tuosto. A Process Calculus for QoS-Aware Applications. In *COORDINATION*, pages 33–48, 2005.
21. C. O'Cinneide. Characterization of phase-type distributions. *Stochastic Models*, 6(1):1–57, 1990.
22. B. Plateau and W. J. Stewart. Stochastic automata networks: product forms and iterative solutions, RR-2939. Technical report, INRIA Research Report, 1996.
23. R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.
24. W. J. Stewart, K. Atif, and B. Plateau. The numerical solution of stochastic automata networks. *EOR*, 86(3):503–525, 1995.
25. H. Younes and R. Simmons. Solving Generalized Semi-Markov Decision Processes using Continuous Phase-Type Distributions. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 742–747. California AAAI Press, 2004.

A Algorithm for Deriving Delay-sequences

Some firing transitions in QIA have multiple synchronized events, identified as a set of firing node names, in their labels, together with a set of delay informa-

tion, D, that gives the (stochastic) performance properties of each firing event. In continuous time scale, only a single event can (be observed to) happen at a time. Therefore, the translation of QIA into a CTMC requires breaking down every QIA transition with atomic multi-event firing into its valid sequences of micro-step single-event transitions. The contents of the delay information sets are used to guide the decomposition of a multi-event transition into its corresponding sequence of single-event transitions. For this purpose, we generate a delay-sequence from the contents of the delay information set, and then spread and split the multi-event firing into micro-step single-event transitions along with their relevant elements in the delay-sequence. For a sequential delay-sequence we enumerate each element, and for a parallel delay-sequence we consider the interleaving of all elements. We give here the algorithm that receives D, a set of delay information, for a firing and returns its corresponding delay-sequence d whose syntax complies with (1) in Section 5. We use the following functions to define this algorithm. For brief explanation of them, we use capital letters for the elements in a delay information set, and let $\langle\rangle$ be an empty delay-sequence and $\mathbb{A}$ be a given finite set of delay information. Moreover we assume that each delay-sequence contains a specific delay information at most once.

- Concatenation $\hat{}$ adds the second delay-sequence to the end of the first delay-sequence:

$$d_1 \hat{} \ d_2 = d_1 d_2$$

- $contains(d, B)$ checks if the delay information B is already included in the delay-sequence d:

$$contains(d,\ B) = \begin{cases} true & \text{if } B \in d \\ false & \text{if } B \notin d \end{cases}$$

- $removes(A, d)$ deletes the delay information A from the delay-sequence d if A exists in d :

$$remove(A,\ \langle\rangle) = \langle\rangle$$
$$remove(A,\ \langle B\rangle \hat{} d) = \begin{cases} d & \text{if } A = B \\ \langle B\rangle \hat{} remove(A,\ d) & \text{if } A \neq B \end{cases}$$

- $getNext(\mathscr{P}, \mathbb{A})$ with $\mathscr{P} \in 2^{\mathbb{A}}$ returns a set of delay information whose relevant events follow the events relevant to the elements in $\mathscr{P}$:
  $getNext(\emptyset,\ \mathbb{A}) = \emptyset$
  $getNext(\{A, B, \cdots, C\},\ \mathbb{A})$
  $$= \begin{cases} \{D\} \cup getNext(\{B, \cdots, C\},\ \mathbb{A}) & \text{if } \exists D \in \mathbb{A} \ \text{ s.t. } O_A = I_D \\ \qquad\qquad \text{where } A = (I_A, O_A, r_A),\ D = (I_D, O_D, r_D) & \\ getNext(\{B, \cdots, C\},\ \mathbb{A}) & \text{otherwise} \end{cases}$$
- $sorting(\mathscr{P}, d)$ with $\mathscr{P} \in 2^{\mathbb{A}}$ sorts the elements in $\mathscr{P}$ according to the occurrence order in the delay-sequence d:
  $sorting(\emptyset,\ d) = \langle\rangle$

$$sorting(\{A, B, \ldots, C\},\ d)$$
$$= \begin{cases} \langle A \rangle \hat{}\, sorting(\{B, \cdots, C\},\ d) & if\ \forall D \in \{B, \cdots, C\} \\ & s.t.\ d = d_1; (A|d_2); d_3; (D|d_4); d_5 \\ \langle\rangle & otherwise \end{cases}$$

– $commonD(d_1, d_2)$ returns a set of delay information that appear in both d_1 and d_2:

$$commonD(d_1,\ d_2) = \{A \in \mathbb{A} |\ contains(d_1,\ A) \wedge contains(d_2,\ A)\}$$

– $parallel(d, A)$ returns a set of delay information in delay sequence d_2 if $A|d_2$ is a subsequence of d:

$$parallel(d_1; (A|d_2); d_3,\ A) = \{C \in \mathbb{A} |\ contains(d_2,\ C)\}$$

– $sub1(d, A)$ returns a subsequence of d that starts from the head of d and ends right before A appears:

$$sub1(d_1; (A|d_2); d_3,\ A) = d_1$$

– $sub2(d, A, B)$ returns a subsequence of d that appears between A and B, and A must occurs before B:

$$sub2(d_1; (A|d_2); d_3; (d_4|B); d_5,\ A,\ B) = d_3$$

– $sub3(d, A)$ returns a subsequence of d that starts right after A and ends at the end of d:

$$sub3(d_1; (A|d_2); d_3,\ A) = d_3$$

ExtractDelaySequence(D)

$D := \{delay_1, \cdots, delay_n\}$ s.t. $delay_i = (I_i, O_i, r_i)$ where $r_i \geq 0$
$d := \langle\rangle$
$Init := \{delay_i | \exists delay_i \in D,\ I_i \cap \bigcup_{k \in\ N_1^n \setminus \{i\}} O_k = \emptyset\}$, $d_i := \langle\rangle,\ \cdots,\ d_{|Init|} := \langle\rangle$
for $i = 1$ to $|Init|$ **do**
  $d_i := d_i \hat{} \langle Init[i] \rangle$
  $Pre := \{Init[i]\},\ Post = getNext(Pre, D)$
  while $Post \neq \{\}$ **do**
    for $k = 1$ to $|Post|$ **do**
      if $contains(d_i, Post[k])$ **then**
        $remove(Post[k], d_i)$
      end if
    end for
    $d_i := d_i \hat{} \langle\ ;\ (\ Post[1]\ | \cdots |\ Post[|Post|]\ ) \rangle$
    $Pre := Post,\ Post := getNext(Pre, D)$
  end while
end for
for $i = 1$ to $|Init|$ **do**
  for $m = 1$ to $|Init|$ **do**
    $Com := commonD(d_i, d_m)$
  end for
end for
$Sort := \langle\rangle \hat{}_{1 \leq i \leq |Init|} sorting(Com, d_i)$
if $Com = \emptyset$ **then**
  $d := d \hat{} \langle (d_1 | d_2 |\ \ldots\ | d_{|Init|}) \rangle$
else
  $FPar = \{delay | \exists delay \in \bigcup_{1 \leq k \leq |Init|} (parallel(d_k, Sort[1])\}$
  $FSub_1 := sub1(d_1, Sort[1])$
  $\vdots$
  $FSub_{|Init|} := sub1(d_{|Init|}, Sort[1]))$
  $d := d \hat{} \langle (FSub_1\ | \ldots |\ FSub_{|Init|}) \rangle \hat{} \langle ; (FPar[1] | \ldots | FPar[|FPar|]) \rangle$
  for $j = 2$ to $|Com|$ **do**
    $MPar = \{delay | \exists delay \in \bigcup_{1 \leq l \leq |Init|} parallel(d_l, Sort[j])\}$
    $MSub_1 := sub2(d_1, Sort[j-1], Sort[j])$
    $\vdots$
    $MSub_{|Init|} := sub2(d_{|Init|}, Sort[j-1]), Sort[j])$
    $d := d \hat{} \langle ; (MSub_1\ | \ldots |\ MSub_{|Init|}) \rangle \hat{} \langle ; (MPar[1] | \ldots | MPar[|MPar|]) \rangle$
  end for
  $LSub_1 := sub3(d_1, Sort[|Sort|])$
  $\vdots$
  $LSub_{|Init|} := sub3(d_{|Init|}, Sort[|Sort|])$
  $d := d \hat{} \langle ; (LSub_1\ | \ldots |\ LSub_{|Init|}) \rangle$
end if
return d