Simulation-Based Performance Analysis of Channel-Based Coordination Models

C. Verhoef, C. Krause, O. Kanters, R. van Der Mei

Simulation-based Performance Analysis of Channel-based Coordination Models

C. Verhoef[1⋆], C. Krause[2⋆⋆], O. Kanters[1] and R. van der Mei[1,3]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
[2] Hasso Plattner Institute (HPI), University of Potsdam, Germany
[3] Vrije Universiteit Amsterdam (VUA), The Netherlands

Abstract. Quantifying the performance of component-based or service-oriented systems is a complex task, e.g., it is non-trivial to calculate the end-to-end quality of service of a composite Web service. An established approach to reason about such systems in general is the use of coordination models, which can provide a formal basis for both their verification and implementation. An example of such a model is the channel-based coordination language *Reo* and its probabilistic extension *Stochastic Reo*. However, all existing performance analysis approaches for Stochastic Reo are restricted to the use of exponential distributions. To this end we introduce a transition structure, which enables a simulation approach for performance evaluation in Reo, enabling the use of arbitrary distributions and predefined probabilistic behaviors. Our approach supports steady-state and transient analysis and, moreover, scales much better than the existing automata-based algorithms.

1 Introduction

Non-functional requirements, such as reliability, security and performance are becoming of increasing importance in many branches of component-based and service-oriented software engineering. Particularly the quantitative aspects inherent in the performance evaluation of composite and distributed systems introduce major challenges. Even if the quality of service (QoS) properties of every individual service and connection is known, it is far from trivial to reason about the end-to-end QoS of the composed system. This is due to the fact that synchronization constraints as well as buffering and routing policies between the different parties in a network can have an impact not only on its qualitative behavioral properties, but also on its overall performance. In the worst case, a 'bad' performance, e.g. if a service takes too long to respond to a request, can even have an influence on the functional properties of the system. However, in this paper we consider rather typical questions of performance evaluation, such as: Where are the bottlenecks in the network? What is the expected delay and the maximum throughput? How much time does it take until a certain event happens? What is the expected utilization of a buffer?

⋆ Corresp. author, e-mail: C.G.Verhoef@cwi.nl. Supported by NWO project Cooper.
⋆⋆ Supported by the research school in 'Service-Oriented Systems Engineering' at HPI.

Building software compositionally out of a set of primitive components or services is a key task in software engineering in general. The coordination paradigm provides concepts to properly describe the allowed interactions between the active entities in a system. A specific coordination approach is considered in the channel-based coordination language *Reo* [1], in which compositionally built components connectors are used as coordination artifacts. Connectors in Reo can be seen as a kind of 'glue code' which coordinate the interactions among a set of components or services from outside. To enable performance evaluation of component connectors, *Stochastic Reo* [2] provides an extension that allows to annotate connectors with stochastic performance properties. Specifically, communication channels in Stochastic Reo are annotated with processing delays. Moreover, to reason about the end-to-end QoS of a connector, its boundary nodes are annotated with data arrival rates, modeling the interaction with its environment. In this way, Stochastic Reo provides detailed information about the performance of the primitive buildings blocks on the one hand, and the external world on the other.

The existing techniques for performance evaluation in Stochastic Reo are all based on analytical methods and essentially follow the same recipe. An automata-based model is used to describe the semantics of every primitive channel in a connector. By composing all these automata, a behavioral model for the whole connector is built. Then, using the stochastic annotations of the channels and boundary nodes, a probabilistic performance model, specifically: a continuous-time Markov chain is generated. Finally, the Markov model is fed into a tool for probabilistic analysis, such as PRISM [3] or Matlab. This approach was taken in [4] using *Quantitative Constraint Automata* (QCA), in [2] using *Quantitative Intentional Automata* (QIA) and in [6] using *Stochastic Reo Automata* (SRA). An implementation of the QIA-based approach is described in [7]. However, all of these approaches to performance evaluation in Reo have two main limitations: (i) they are all restricted to the use of exponential distributions, and (ii) they suffer from the state space explosion problem, because the automaton / continuous-time Markov chain for the whole system has to be computed in advance.

Complementary to the existing analytical methods, we consider a simulation approach for performance analysis in Stochastic Reo, which enables the use of *arbitrary* (not just exponential) distributions for describing stochastic properties of channels and components. Our approach is based on the coloring semantics [8] of Reo, which enables a step-wise execution scheme (cf. [9]). Thus, state spaces can be generated on-the-fly during the simulation without requiring to keep track of the execution history. Therefore, our approach scales much better than the existing automata-based techniques, which require to compute the whole state space *before* the actual analysis starts. The coloring semantics which we use in our approach, supports context-dependent primitives, such as the *LossySync* channel (cf. [8]). Moreover, it allows to model the availability of I/O requests at the boundary of a connector, which is a key ingredient to reason about the end-to-end performance of a connector.

We have implemented our simulation approach for Reo in a sophisticated graphical tool, as part of the Eclipse Coordination Tools (ECT) [10]. Connectors can be specified using a graphical editor in ECT. By annotating these graphical connector models with stochastic information, our simulator generates a large number of performance statistics. Our tool supports both steady-state and transient analysis and can be applied to connectors built using all standard and even user-defined Reo channels. To analyze specific behaviors of the modeled system, a number of tools are available to the user, such as automatic deadlock and livelock detection, visualization of the connector colorings, and charts for the behavior of simulation results during the simulation. Various stopping conditions can be specified for the simulation. Our simulator generates a number of statistical outputs depending on the chosen type of simulation, for an overview we refer to Section 4.1.

Related work. Model-based methodologies to assess performance of distributed software systems can be categorized [11] in: queuing networks, state/transition-based analysis, and software performance engineering. A survey of the available results in the theory of queuing networks is given in [12]. The Method of Layers in [13], models the responsiveness of composite services using closed queuing networks using Mean Value Analysis. Stochastic rendezvous networks are introduced in [14] for performance evaluation of distributed systems with synchronization. Software Performance Engineering is suggested in [15] to enable the integration of performance analysis into the software development process. Simulation of stochastic graph transformation systems is described in [16]. In [17] a methodology for simulation of embedded systems is presented. Yacoub et al. [18] focus on reliability analysis for component-based systems. In [19] a reasoning technology to simulate and verify pure Web services is defined. In [20] Generalized Stochastic Petri Nets (GSPN) are proposed for performance analysis of multiprocessor systems. Performance evaluation is done by generating continuous-time Markov chains [21]. Haas provides an overview of simulation techniques for GSPNs [22]. GreatSPN is a simulation tool for performance evaluation of distributed systems using GSPNs [23]. Compared to GSPNs, Reo has a strong notion of synchronization, which, just like the notion of context dependency, propagates through connectors, both not supported by GSPNs. Due to this, traditionally automata based models are used as semantical models for Reo.

Acknowledgments. We are grateful to Farhad Arbab and anonymous reviewers for their insightful comments.

Organization. Section 2 gives a brief overview of (Stochastic) Reo. We define the operational semantics underlying our simulation and introduce our transition system in Section 3. In Section 4 we present our simulation-based stochastic analysis procedure. Our simulation tool is described in Section 5. We present two case studies in Section 6. Section 7 contains conclusions and future work.

2 Channel-based coordination with Reo

The simulation approach we present here targets the channel-based coordination language Reo [1]. Channels in Reo are entities that have exactly two ends, which can be either *source* or *sink* ends. Source ends accept data into, and sink ends dispense data out of their channel. Reo allows directed channels as well as *drain* and *spout* channels, which have respectively two source and two sink ends. Channels may impose constraints on the dataflow at their ends. For instance, the communication through channels can be (a)synchronous and (un)buffered.

For the scope of this paper, we consider a fixed set of channels, summarized in Table 1. The *Sync* channel consumes data items at its source end and dispenses them at its sink end. The I/O operations are performed synchronously and without any buffering. Thus, the channel blocks if the party at the sink end is not ready to receive data. The *LossySync* channel behaves in the same way, except that it does not block the party at its source end. Instead, the data item is consumed and destroyed by the channel if the receiver is not ready to accept it. The *SyncDrain* channel is also synchronous, but it differs in the fact that it has two source ends through which it consumes and destroys data items synchronously. The *FIFO* channel is a directed, asynchronous channel with a buffer of size one.

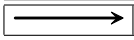
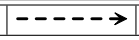
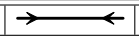
Sync	*LossySync*	*SyncDrain*	*FIFO*
			

Table 1. Some basic Reo channels

Channels in Reo can be joined together using nodes, which read data items from sink ends and write data items to source ends of channels that coincide in it. Nodes in Reo behave as non-deterministic mergers on the sink ends and as (synchronous) replicators on the source ends. This means that a node non-deterministically reads a data item from one of the incoming sink ends and replicates it to all outgoing source ends without buffering it.

2.1 Building connectors

In Reo, channels and nodes are joined together to build so-called *connectors* which resemble electronic circuits. These connectors are used as *glue code* between components or services and essentially enforce a communication protocol between them. This coordination of components or services is performed from outside and without their knowledge, which is also referred to as *exogenous* coordination.

An important aspect of Reo is the fact that nodes do not buffer data items and therefore allow synchrony to propagate through the connector. For instance, a sequence of n *Sync* channels joined together using nodes has the same qualitative behavior as a single *Sync*. Note also that Reo allows an arbitrary mixing of synchrony and asynchrony.

Example 1. We consider a simple instant messenger application, depicted in Fig. 1. Two *Client* components exchange messages via a connector. Messages are exchanged via *FIFO* channels and are, thus, buffered. When leaving the buffer again, the messages are synchronously replicated by the node behind the *FIFO* and sent to both clients. This can succeed only when both clients are ready to accept data, i.e. when there are pending read requests at both *in* ports. In a nutshell, this connector ensures that the clients get –as an acknowledgment– a copy of their own message when the other client has successfully received it.

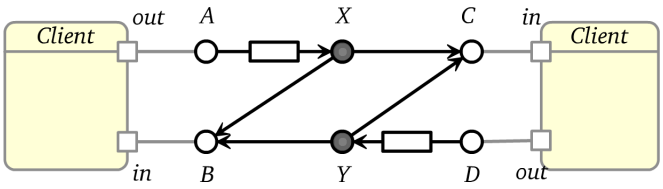


Fig. 1. Instant messenger application modeled in Reo

2.2 Stochastic Reo

Stochastic Reo is an extension of Reo annotated with stochastic properties. In particular, we distinguish between the following two quantitative aspects in Reo:

- **Channel delays**: Every channel has one or more associated delays represented by a set of random variables. Such a delay models how long it takes for a channel to transfer or process a data item. For instance, a $LossySync_{A\to B}$ has two associated delays 'dAB' and '$dALost$', respectively for successful dataflow through the channel, and losing data in the channel if B is not ready to receive data. A $FIFO_{A\to B}$ has two associated delays: 'dAF' and 'dFB'. The former represents the delay for the dataflow from A into the buffer. The latter models the dataflow out of the channel. *Sync* and *SyncDrain* channels have only one delay, i.e., for successful dataflow.
- **Arrivals at nodes**: I/O operations are performed at the boundary nodes of a connector through which it interacts with its environment (depicted as empty circles). We assume the time between consecutive arrivals of read and write requests at the boundary nodes depends on their associated stochastic processes. For instance, 'dA' and 'dB' in the connector in Fig. 1 represent the associated arrival processes of 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.

Note that arrivals at nodes are considered only for boundary nodes, e.g. A, B, C, D, but not X, Y in Fig.2. Internal nodes are used for synchronous dataflow only and merely pump data in the connector, without interaction with the environment. Therefore, internal nodes have neither an associated arrival rate, nor a delay.

2.3 Distributions

In our simulation approach and particularly in the simulation tool which we present in Section 5, we support a number of distribution types, some of them being general stochastic distributions, while others being special constructs for steering the simulation process. The types of supported distributions and their parameters are listed in Table 2. The value after the parameters between the brackets indicates the type of the parameter, where b = Boolean, i = integer, r = real, and s = string.

Distribution	*Param 1*	*Param 2*	*Param 3*
Beta	α (r)	β (r)	
Binomial	n (i)	p (r)	
Chi^2	k (i)		
Constant (*Con*)	value (r)		
Exponential (*Exp*)	λ (r)		
F	d_1 (r)	d_2 (r)	
Gamma	k (r)		
Lognormal	μ (r)	θ (r)	
Poisson	λ (r)		
Triangular (*Tri*)	low (r)	high (r)	avg (r)
Uniform	low (r)	high (r)	
Weibull	k (r)		
IfNeeded			
Always			
Trace	path (s)	loop (b)	

Table 2. Supported distributions

Channel	Delay 1	Delay 2
$FIFO_{A\to X}$	$Exp(2)$	$Exp(1)$
$Sync_{X\to C}$	$Tri(5,10,7)$	–
$FIFO_{D\to Y}$	$Exp(2)$	$Con(0)$
$Sync_{Y\to B}$	$Con(0)$	–
$Sync_{X\to B}$	$Exp(1)$	–
$Sync_{Y\to C}$	$Exp(2)$	–

Table 3. Example channel delays

Node	Arrivals
A	$Exp(1)$
B	$Exp(10)$
C	$Exp(1/2)$
D	$Exp(1/8)$

Table 4. Example node arrival rates

Example 2. For the instant messenger example, we consider the channel delay and node arrival parameters chosen such that analysis is not trivial, given in Table 3 and 4, respectively. We assume exponential distributions for the request arrivals at all boundary nodes and for most of the channel delays. However, we assume that the dataflow between the buffer of $FIFO_{D\to Y}$ to the boundary node B can be performed without any delay ($Con(0)$). Moreover, the delay of $Sync_{X\to C}$ is approximated using a triangular distribution.

3 Coloring semantics with states

In our simulation approach, we use the so-called *coloring semantics* [8] of Reo, introduced by Clarke et al. to properly model context-dependent behavior as required for instance for the *LossySync* channel. The basic idea of the coloring semantics is to associate flow and no-flow colors to channel ends. As shown in [8] one *flow* and two *no-flow* colors are sufficient to model context-dependency. Essentially, the two different *no-flow* colors are used to distinguish between absence and presence of an I/O request. Table 5 depicts the names and graphical notations of the flow and the two no-flow colors, as used in this paper.

Color name	Symbol
flow	
no-flow-provide-reason	
no-flow-require-reason	

Table 5. Colors

Sync	*Merger*
(S1) (S2) (S3) (S4)	(M1) (M2) (M3) (M4)

Table 6. Example colorings

The color *flow* represents ordinary dataflow at a channel end. The two *no-flow* colors are used to encode a direction of the reason for the fact that no dataflow is possible. Intuitively, *no-flow-provide-reason* models the fact that the receiving or sending party is not ready to perform an I/O operation. Conversely, *no-flow-require-reason* says that the party is ready to receive or send data, but is not allowed to perform the operation. At the boundary of a connector, the two *no-flow* colors can be interpreted as lack of dataflow – either because of a missing, or in spite of a present I/O request.

Valid behaviors of channels are described as colorings of their respective ends. Table 6 depicts the colorings of the *Sync* the *Merger* primitive. The latter is used for modeling nodes in Reo. For the colorings of other primitives such as the *FIFO* channel we refer to [8]. Note that the colors are always read from the perspective of the primitive. For instance, in coloring (S2) of the *Sync* the party at the right end provides a reason for no flow, whereas the source end on the left requires a reason. This models the behavior where data is available at the source end but the receiver at the sink end is not ready to accept data. Similarly, in coloring (S3) there is no flow, because there is no data available at the source end. Finally, coloring (S4) models the situation where no data is available and the receiver is also not ready to accept any data. Similarly, the colorings of the *Merger* primitive in Table 5 show the valid dataflows through sink nodes and how reasons for no dataflow are being propagated.

Valid colorings of primitives are joined together and give rise to valid colorings of the whole connector (see [8] for details).

Example 3. Fig. 2 depicts an example coloring of the instant messenger application. The coloring is based on the following state of the connector: $FIFO_{A\to X}$ is full, $FIFO_{D\to Y}$ is empty, there are read requests at the boundary nodes B and C, and no write requests at A and D. This particular coloring models a dataflow action from the full $FIFO_{A\to X}$ to both clients, i.e., a synchronized message delivery and acknowledgment.

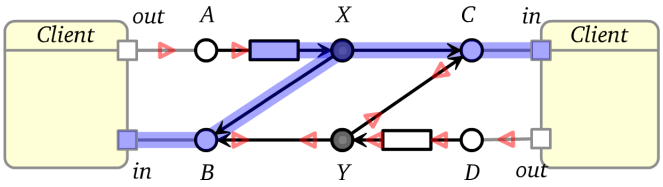


Fig. 2. A coloring of the instant messenger application

3.1 Coloring transition system

Colorings describe only dataflow events, but not the state of primitives or the whole connector. Therefore, we now incorporate a notion of state into the coloring model, which gives rise to a transition structure defined in the following. Let $\mathcal{C}olor$ be a fixed set of flow colors, as defined in Table 5.

Definition 1 (coloring transition system). *A* coloring transition system $C = (N, B, Q, \Longrightarrow)$ *consists of a set of nodes* N*, a set of boundary nodes* $B \subseteq N$*, a set of states* Q *and a set of coloring transitions* $\Longrightarrow \subseteq Q \times \mathcal{C}olor^N \times Q$*.*

We often write $q \Longrightarrow_c q'$ for a transition where $c \in \mathcal{C}olor^N$ is a coloring. Colorings model dataflows, which is why we also refer to transitions as *dataflow transitions* or just *dataflows*.

However, this model does not reflect the interaction of the connector with its environment. Specifically, boundary nodes receive requests from their components. Therefore, we model the state of boundary nodes explicitly as:

- $\mathcal{S}tates = \{empty, waiting, busy\}$

A boundary node is *empty* when there is no I/O request pending, *waiting* when the node received an I/O request pending for processing, and *busy* when it is sending or receiving data. We model the state change of boundary nodes on request arrivals using the map $\mathcal{A}rrival : \mathcal{S}tates \to \mathcal{S}tates$ defined as follows:

- $\mathcal{A}rrival = \{empty \mapsto waiting, waiting \mapsto waiting, busy \mapsto busy\}$

In the following, we relate the state of the boundary nodes with the coloring semantics. Specifically, we define a transition structure where colorings are being enabled based on the presence/absence of requests. Moreover, we model the start and the end of dataflows as distinct events. This is important to measure, e.g., the duration of dataflows and the waiting time of requests at boundary nodes.

Definition 2 (induced intensional coloring transition system). *Given a coloring transition system* $C = (N, B, Q, \Longrightarrow)$*. The induced* intensional coloring transition system *is a tuple* $\mathcal{C} = (\mathcal{Q}, \to, \overset{start}{\Longrightarrow}, \overset{end}{\Longrightarrow})$ *where:*

- $\mathcal{Q} = Q \times \mathcal{S}tates^B \times 2$ *is a set of states where a state* $q \in \mathcal{Q}$ *consists of a state* $q_\bullet \in Q$ *together with* $(q_n)_{n \in B} \in \mathcal{S}tates^B$ *and* $q_\sim \in \{true, false\}$
- $\to \subseteq \mathcal{Q} \times B \times \mathcal{Q}$ *is a set of request arrival transitions*
- $\overset{start}{\Longrightarrow}, \overset{end}{\Longrightarrow} \subseteq \mathcal{Q} \times \mathcal{C}olor^N \times \mathcal{Q}$ *are sets of dataflow start and end transitions*

where the transition relations are defined by the following rules:

$$\frac{\exists n \in B : q'_n = \mathcal{A}rrival(q_n) \quad \forall m \neq n \in B : q'_m = q_m \quad q'_\bullet = q_\bullet \quad q'_\sim = q_\sim}{q \to_n q'} \tag{1}$$

$$\frac{\begin{array}{lll} q_\bullet \Longrightarrow_c q'_\bullet & & c(n) = \rightarrow\!\!-\!\!\Rightarrow\ q_n = q'_n = empty \\ q_\sim = false & \forall n \in B : & c(n) = -\!\!\leftarrow\!\!\Rightarrow\ q_n = q'_n = waiting \\ q'_\sim = true & & c(n) = \text{——}\Rightarrow\ q_n = waiting \wedge q'_n = busy \end{array}}{q \overset{start}{\Longrightarrow}_c q'} \tag{2}$$

$$\frac{p \overset{start}{\Longrightarrow}_c p' \rightarrow^* q \qquad \begin{array}{ll} q_\sim = true & \\ q'_\sim = false & \end{array} \qquad \forall n \in B\colon \begin{array}{l} q_n = busy \Leftrightarrow q'_n = empty \\ q_n \neq busy \Leftrightarrow q'_n = q_n \end{array}}{q \overset{end}{\Longrightarrow}_c q'} \tag{3}$$

In an intensional coloring transition system (ICTS), we distinguish between request/data arrival transitions (1), dataflow start (2), and dataflow end (3) transitions. Moreover, the state space of an ICTS is enriched with the states of the boundary nodes and a global dataflow flag. This operational semantics is the basis of our simulation approach.

4 Simulation-based stochastic analysis

In this section, we show how to construct a discrete event simulator engine (DES) [24] for Stochastic Reo, which can be used for performance evaluation of connectors. The core idea of simulation in general is to generate a large number of sample path sequences, which are used as a characterization of the system behavior. Formally, a sample path is a realization of a (stochastic) process $X(t)$ of transitions between states over time. In a DES, states change at discrete points in time, rather than continuously with time. An advantage of simulation over algorithmic approaches, such as QIA [2], is that all kinds of stochastic distributions can be used for specifying channel delays and request arrivals at nodes, in particular the ones given in Table 2. As underlying stochastic semantic model for our approach we use a generalized semi-Markov process (GSMP), a classical model for discrete event stochastic systems [25].

Definition 3 (generalized semi-Markov process). *A* generalized semi-Markov process *is a stochastic process $X(t)$ with state space S generated by a stochastic timed automaton $\mathcal{A}$ defined as $\mathcal{A} = (S, E, F(x), T(x,e), p_0, P)$, with E a set of events, $F(x)$ the set of feasible events at state $x \in S$, $T(x,e)$ the state transition function with x the current state and event $e \in E$, p_0 the probability mass for the initial state, and P the probability function for all events.*

Lemma 1. *Let $\mathcal{C} = (\mathcal{Q}, \rightarrow, \overset{start}{\Longrightarrow}, \overset{end}{\Longrightarrow})$ be an ICTS. This induces a minimal GSMP $\mathcal{A} = (S, E, F(x), T(x,e), p_0, P)$ such that the states are given by $S = \mathcal{Q}$, events are $E = \{request_b \mid b \in B\} \cup \{start_c \mid c \in \mathcal{C}olor^N\} \cup \{stop_c \mid c \in \mathcal{C}olor^N\}$, and the transitions T are given by the union of $\rightarrow$, $\overset{start}{\Longrightarrow}$ and $\overset{end}{\Longrightarrow}$.*

Proof. The semi-Markov property holds, because for a transition $s \xrightarrow{e} s'$ the next state s' is depending only on the current state $s \in S$ and event $e \in E$. □

Note that the probability function P and the initial probability mass p_0 are derived from the channel delays and request arrival distributions specified by the user. Thus, mapping the semantical ICTS model to a GSMP enables the use of discrete event simulation for performance analysis of connectors.

Since we are not limited to use only continuous distributions, to model delays and inter-arrival times, multiple events could take place at the same time. In such a case, the correct DES process order of the event sequence is crucial. Therefore, we enforce that *dataflow* events take precedence over *request arrival* events. Furthermore, in this case, multiple possible dataflows, i.e. colorings, can be activated. A *scheduler* then selects one dataflow based on a given execution policy, such that only one dataflow is active at a time to ensure proper synchronization.

4.1 Simulation and analysis

We distinguish between two types of simulation: *steady-state* analysis, and *transient* analysis. Moreover, we consider a number of stopping criteria, i.e., *maximum simulation time*, *maximum number of events*, *deadlocks*, *livelocks*, and *observed states*. The latter offers the possibility to end the simulation in a specific state, which is particularly important for transient analysis.

Channel delays and node inter-arrival times. As described in Section 2.2, we associate a number of stochastic delays to every channel, and request inter-arrival times to boundary nodes. The derived GSMP allows the distributions to be general stochastic distributions, as in Table 2. Besides the standard distributions there are some special constructs. *IfNeeded* and *Always* can be used to model inter-arrival times without specifying a particular distribution, but depending on the current state of the connector. *IfNeeded* ensures that a boundary node always is in the *empty* or *busy*, but never *waiting*. Thus, request are spawned on demand. *Always* ensures that a node is never in the *empty* state. Whenever the node is finished with a dataflow, it immediately switches to *waiting*. Moreover, predefined inter-arrival times and channel delays can be specified as a *Trace*.

QoS measures. Among others, the following QoS measures can be computed during the simulation. The *channel utilization*, *channel locked utilization*, and *dataflow utilization* represent the percentage of time a channel is busy handling requests, locked for further processing, and the time a certain dataflow is activated, respectively. Request arrival statistics for boundary nodes include the *expected node state* and *request observation state*. The latter is the probability for the node being in a certain state during a request arrival. The *expected waiting time* measure is the expected waiting time at each boundary node. The *conditional waiting time* is the waiting time after a request arrived at a node. For *FIFO* channels, the expected *buffer utilization* can be calculated. For *LossySync* channels, the expected *loss ratio of requests* is an interesting measure. For nodes, the expected *merger direction* gives further insight about the internal routing of data in the connector. Global QoS measures of interest include the *steady state probabilities* of the connector and the *dataflow probabilities*. The latter is the probability of a specific coloring being active.

End-to-end delay A special role plays the *expected end-to-end delay* between a given start to another end boundary node of a connector. We compute the end-to-end delay of dataflows using a recursive depth-first traversal through all channels and nodes with active dataflow. Based on the active dataflow, we calculate the longest dataflow path through the connector, from the given start to the given end point. This uniquely determines the duration of the dataflow and, thus, the point in time where the dataflow is finished. A detailed algorithm for computing the end-to-end delay is given in [26].

5 Tool support

We have implemented the presented simulation approach for Reo in discrete event simulation tool as part of the Eclipse Coordination Tools [10]. All distribution types given in Table 2 and all QoS measures described in Section 4.1, including end-to-end delays, are supported by this tool. The current scheduler implementation selects a dataflow randomly, with even distribution, thus does not prioritize. For all statistics, the expectation, standard deviation, the coefficient of variation, and confidence interval are calculated. ECT includes a graphical editor for specifying connector models. These graphical connector models are annotated with stochastic information which is sufficient for performing the stochastic simulation with our tool. The simulator is integrated with the graphical environment of ECT, as shown in the screenshot in Fig. 3, and generates a number of charts and diagrams.

Our simulation tool supports both steady-state and transient analysis. Steady-state analysis is only possible if the system actually reaches steady-state, which is not guaranteed in simulation-based analysis. Therefore, we have implemented a number of tools to facilitate convergence checking. Specifically, the tool generates charts that show how the different QoS measure develop over time during the simulation runs. Furthermore, the tool computes the number of observations, result histograms, and supports automatic deadlock and livelock detection.

All analysis results are available in the user interface, and can can be additionally exported for subsequent analysis with other tools. Dataflows, i.e., colorings, are visualized graphically which provides an intuitive way to investigate the dataflow statistics.

Depending on the size of the modeled system, state spaces can grow very fast. However, the implementation of the coloring semantics in ECT supports step-wise execution. Our simulation tool uses this functionality for an on-the-fly generation of the state space, thus, enabling simulation without prior computation of the whole state space.

6 Case studies

In the following, we present two case studies for our simulation approach. We perform steady-state simulations with the ending condition of 10,000,000 events,

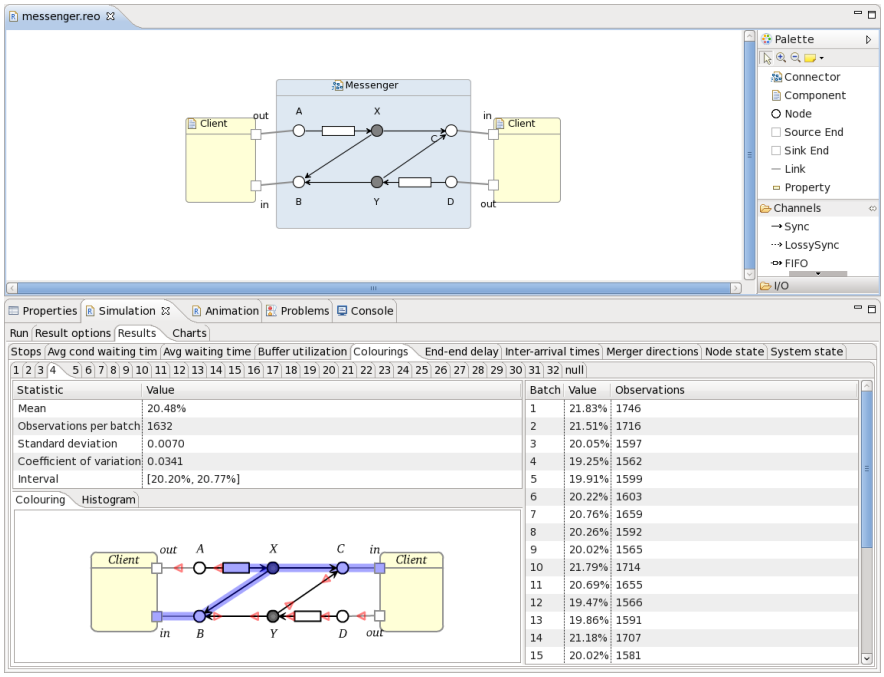


Fig. 3. Simulation-based stochastic analysis in the Eclipse Coordintation Tools

and a warm-up period of 10,000 events. A comprehensive case study of an industrial software system is described in [27].

6.1 Case 1: Instant Messenger

In this example we investigate the instant messenger example, introduced in Section 2.1. As distinct from existing performance evaluation techniques for Reo our approach allows us to analyze in detail the impact of the configuration, as specified in Table 3, on the behavior of the instant messenger.

Using our simulator, we found an asymmetry between the two dataflow regions of the message delivery parts, caused by the configuration. For *Client* 1, this is the dataflow represented by the coloring in Fig. 2. For *Client* 2 it is the symmetric dataflow. Using the *dataflow utilization* statistic, we found out that in 54.0% of the time, dataflow for the message delivery of *Client* 1 is active, versus only 3.6% of the time for the message delivery of *Client* 2. We can also look to the dataflows from another perspective, i.e., whether the clients are both sending, both receiving, one is sending and one is receiving, or both are idle. The results are shown in Table 7.

When we look to the *merging directions* of node B, 64.1% of the data arrives from the $Sync_{X \to B}$ (acknowledgment message from *Client* 1) and only 35.9% arrives from the direction of node Y. Due to the symmetrical structure of the

Sending	Receiving	Probability
–	–	32.9%
–	X	57.3%
X	–	9.4%
X	X	0.3%

Table 7. Dataflow probabilities

$\mu_{D,fifo}$	$A \to C$		$D \to B$	
	Delay	σ	Delay	σ
0.125	10.867	0.110	14.734	0.200
2.000	7.677	0.039	8.343	0.139
50.000	7.642	0.029	8.361	0.117

Table 8. End-to-end delays

connector, the merging directions for node C are the same. From the *buffer utilization* statistics, we derive that the buffer between A and X is full 92.8% of the time, compared to 59.5% for the buffer between D and Y. Using the *expected node states*, blocking probabilities of all boundary nodes can be inspected (the percentage time the boundary node is waiting or busy). The probabilities are, for node A: 87.1%, B: 98.0%, C: 59.7%, and D: 41.9%. The very high blocking probability of node B can be explained by the very high arrival rate of requests at B and the high delay of the dataflow of the message delivery part of *Client* 1.

In Table 8, the effect on the end-to-end delay from A to C and from D to B is shown, varying the delay $\mu_{D,fifo}$ between D and the buffer of $FIFO_{D \to Y}$. When we decrease the delay from 0.5 to 0.02 (rate 2.0 and 50.0), the decrease in delay between A and C is very small (7.677 vs. 7.642). If we increase the delay from 0.5 to 8.0 there is, as one would expect, a major increase in the end to end delay from D to B. Interestingly, the delay from A to C increases as well. This is due to the fact that if the dataflow between D and the *FIFO* buffer is active, no other dataflow can happen at the same time and the waiting time of requests at node A increases and therefore also the end-to-end delay between A and C.

6.2 Case 2: Production line decision making

In this example, we model a production line in Reo, as shown in Fig. 4. It uses 1 permanent server on the right-hand side, and whenever there are 3 jobs in the queue, modeled by a sequence of *FIFO* channels, one additional server is started. Whenever a job is assigned to the queue, it will wait until it has been serviced by server 1, so it will never go to server 2. We vary the service rate of the base server and keep all other parameters constant to investigate the impact on the queue length.

For the arrival rate we have chosen a Weibull distribution with $k = 1.5$. Both servers have a log-normal distribution with $\mu = 0$ and $\sigma = 1$. We vary the μ of the first server. The average queue length of the queue before the permanent server is shown in Figure 5. The average inter-arrival times at boundary node A is around 0.9, so when the average server duration of the base server exceeds this time, the server is not capable of handling all request. Because of this, the queue will fill up and the second server will be used to help the first server. When the average service duration is around 0.9, the average queue length increases rapidly, until it converges to the maximum queue size. When the service time becomes large enough, almost all of the requests will be redirected to server 2 or blocked if server 2 is also not available.

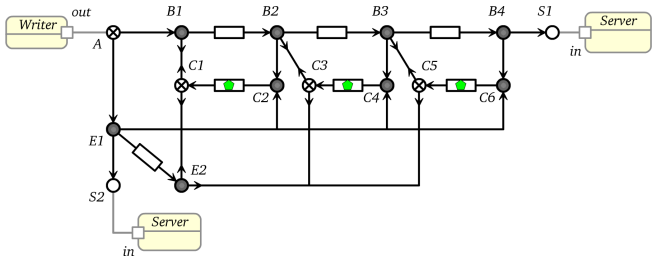


Fig. 4. Reo connector for a production line

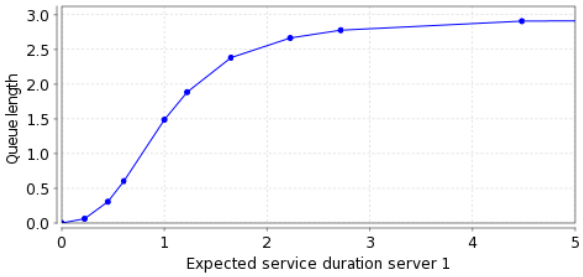


Fig. 5. Average queue length for production line

7 Conclusions and future work

We introduced a performance evaluation approach for Reo based on a new transition system and discrete event simulation. Our approach is more powerful then the existing techniques for performance analysis in Reo in two respects: (i) it allows the use of arbitrary distributions, and (ii) scales much better due to an on-the-fly state-space generation. We implemented our approach in a tool that supports both steady-state and transient analysis.

As future work, we plan to support the use of convergence of statistics as stopping criteria and to add automatic sensitivity analysis. To gain more insight in the precise distribution of the results of statistics, keeping all information of every single observation, and a detailed distribution plot, will be helpful. Another promising extension is to link current automata-based models directly to the simulator state-space. Thereby, it will be possible to define statistics and stopping criteria for different semantical models.

References

1. Arbab, F.: Reo: A channel-based coordination model for component composition. Mathematical Structures in Computer Science **14** (2004) 329–366
2. Arbab, F., Chothia, T., Mei, R., Meng, S., Moon, Y.J., Verhoef, C.: From coordination to stochastic models of QoS. In: Proc. of COORDINATION'09, Berlin, Springer-Verlag (2009) 268–287

3. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic Symbolic Model Checker. In: Proceedings of TOOLS'02. LNCS, Springer-Verlag (2002) 200–204
4. Arbab, F., Chothia, T., Meng, S., Moon, Y.: Component connectors with QoS guarantees. In: Proc. of COORNATION'07. LNCS 4467, Springer (2007) 286–304
5. Chothia, T., Kleijn, J.: Q-Automata: Modelling the Resource Usage of Concurrent Components. ENTCS **175**(2) (2007) 153–167
6. Moon, Y.J., Silva, A., Krause, C., Arbab, F.: A compositional semantics for stochastic Reo connectors. In: Proc. of FOCLASA'10. (2010) 93–107
7. Arbab, F., Meng, S., Moon, Y., Kwiatkowska, M., Qu, H.: Reo2MC: a tool chain for perf. anal. of coordination models. In: Proc. of FSE, ACM (2009) 287–288
8. Clarke, D., Costa, D., Arbab, F.: Connector colouring I: Synchronisation and context dependency. Science of Computer Programming **66**(3) (2007) 205–225
9. Proença, J.: Deployment of Distributed Component Based Systems. PhD thesis, Leiden University, The Netherlands (2011)
10. ECT: Eclipse Coordination Tools. http://reo.project.cwi.nl/ (2011)
11. Gijsen, B., van der Mei, R., van den Berg, J.: An Integrated Performance Modeling Approach for Distributed Applications and ICT Systems. In: CMG-CONFERENCE. Volume 2., Computer Measurement Group; 1997 (2003) 471–482
12. Boxma, O., Daduna, H.: Sojourn times in queueing networks. Stochastic Analysis of Computer and Communication Systems (1990) 401–450
13. Rolia, J., Sevcik, K.: The method of layers. Software Engineering, IEEE Transactions on **21**(8) (2002) 689–700
14. Woodside, M., Neilson, J., Petriu, D., Majumdar, S.: The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. Computers, IEEE Transactions on **44**(1) (2002) 20–34
15. Smith, C.: Performance Engineering of Software Systems. Addison-Wesley (1990)
16. Torrini, P., Heckel, R., Ráth, I.: Stochastic simulation of graph transformation systems. In: FASE. (2010) 154–157
17. Ledeczi, A., Davis, J., Neema, S., Agrawal, A.: Modeling methodology for integrated simulation of embedded systems. ACM Transactions on Modeling and Computer Simulation (TOMACS) **13**(1) (2003) 82–103
18. Yacoub, S., Cukic, B., Ammar, H.: A scenario-based reliability anal. approach for component-based software. Reliability, IEEE Trans. on **53**(4) (2004) 465–480
19. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: Proc. of the 11th int. conf. on WWW, ACM (2002) 77–88
20. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems (TOCS) **2**(2) (1984) 93–122
21. Haverkort, B.R., Marie, R., Rubino, G., Trivedi, K.S., eds.: Performability Modelling: Techniques and Tools. Wiley (2001)
22. Haas, P.: Stochastic petri nets: Modelling, stability, simulation. Springer (2002)
23. Chiola, G., Franceschinis, G., Gaeta, R., Ribaudo, M.: GreatSPN 1.7: graphical editor and analyzer for timed and SPNs. Perf. Eval. **24**(1-2) (1995) 47–68
24. Fishman, G.: Principles of discrete event simulation. John Wiley, New York (1978)
25. Glynn, P.W.: On the role of generalized semi-markov processes in simulation output analysis. In: Proc. WSC '83, IEEE Press (1983) 39–44
26. Kanters, O., Verhoef, C., Schut, M.: QoS analysis by simulation in Reo. Vrije Universiteit Amsterdam, The Netherlands (2010)
27. Moon, Y., Arbab, F., Silva, A., Stam, A., Verhoef, C.: Stochastic Reo: A case Study. In preparation. (2011)