

A Semantic Model for Service Composition with Coordination Time Delays

N. Kokash, B. Changizi and F. Arbab¹ * **

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract. The correct behavior of a service composition depends on the appropriate coordination of its services. According to the idea of channel-based coordination, services exchange messages through channels without any knowledge about each other. The Reo coordination language aims at building connectors out of basic channels to implement arbitrarily complex interaction protocols. The activity within a Reo connector consists of two types of communication, each of which incurs a delay: internal coordination and data transfer. Semantic models have been proposed for Reo that articulate data transfer delays, but none of them explicitly considers coordination delays. More importantly, these models implicitly assume that (1) internal coordination and data transfer activities take place in two separate phases, and (2) data transfer delays do not affect the coordination phase. These assumptions prevent maximal concurrency in data exchange and distort the evaluation of end-to-end delays in service composition models. In this paper, we introduce a novel compositional automata-based semantic model for Reo that explicitly represents both internal coordination and data transfer aspects in channel-based connectors. Furthermore, we map the proposed model to the process algebra $mCRL2$, which allows us to generate state spaces for connectors with time delays and analyze them automatically.

1 Introduction

Provisioning of end-to-end client-perceived Quality of Service (QoS) in concurrent systems is a well-known problem that has been attracting attention of researchers and software engineers over the past few decades. The problem acquired even more attention with the advent of service-oriented computing where systems are composed out of loosely-coupled services of different vendors to realize complex value-added business processes. The quality of what a service-based system offers is derived from the quality of its constituent parts, the quality of the so-called “glue code” that coordinates the execution of the individual services, and the characteristics of the underlying infrastructure such as, e.g., the physical location of servers and the characteristics of the communication networks connecting them.

* Corresponding author, email: Natallia.Kokash@cwi.nl

** Supported by IST COMPAS FP7-ICT-2007-1 project, contract number 215175

One of the ways to coordinate autonomous services is to use connectors. Reo [1] is a model for coordination of software components or services wherein complex connectors are constructed out of simple primitives called channels. By composing basic channels, arbitrarily complex interaction protocols can be implemented. A distinctive characteristic of Reo is the propagation of synchrony: with the help of connectors composed of synchronous channels, one can define transactional protocols where all participating services should be ready to provide or consume messages simultaneously. This facility is very useful as it enables models that are both concise and compositional, but it also makes the problem of describing the operational semantics of Reo a non-trivial task. The most basic semantic model that currently exists for Reo relies on constraint automata [2]. In this model, states represent configurations of data stored in the buffers of Reo networks, while transition labels are composed of (i) sets of channel ends where dataflow is observed simultaneously, and (ii) data constraints necessary to trigger such transitions. Constraint automata represent a theoretical basis for most of the available validation and verification tools for Reo, which are integrated in a framework known as the Eclipse Coordination Tools (ECT)¹.

Several extensions for Reo and its initial semantics have been proposed to capture the notions of timed, context-sensitive, probabilistic and stochastic behavior. However, none of these models accounts for the possible delays that the channels need to transfer data. The existing approaches that aim at extending Reo with QoS information [3,4] assume that delays in channels do not affect the operational semantics of Reo connectors. Nevertheless, as we argue in this paper, this assumption can limit the degree of concurrency in the presence of transactions with different durations and lead to imprecise estimations of end-to-end communication delays in service compositions.

To fix this problem, we introduce a more expressive semantic model for Reo, called *action constraint automata*, which distinguished several actions performed internally by each channel to manifest its behavior. By observing the start and the end of a multiparty communication as well as the start and the end of actual dataflow in each channel, we include more information into the model describing the behavior of a circuit. This approach eventually helps us to compute the total delay in a circuit given the delays for each individual channel. In this paper, we introduce the action constraint automata model, illustrate its application to dataflow modeling in Reo, and discuss the tool support achieved by mapping action constraint automata into the process algebra mCRL2 as well as the integration of the mCRL2 toolset within ECT.

The remainder of this paper is organized as follows. In Section 2, we explain the basics of Reo. In Section 3, we give examples that motivate our work. In Section 4, we introduce the action constraint automata-based semantic model for Reo. In Section 5, we use this new automata to give semantics to our motivating examples. In Section 6, we discuss the translation of this model to the process algebra mCRL2, which enables the application of the mCRL2 toolset for

¹ <http://reo.project.cwi.nl/>

the verification of Reo circuits. Finally, in Section 7, we conclude the paper and outline our future work.

2 Background

Reo is a coordination language in which components and services are coordinated exogenously by channel-based connectors [1]. Connectors are essentially graphs where the edges are user-defined communication channels and the nodes implement a fixed routing policy. Channels in Reo are entities that have exactly two ends, also referred to as ports, which can be either source or sink ends. Source ends accept data into, and sink ends dispense data out of their channel. Although channels can be defined by users, a set of basic Reo channels with predefined behavior suffices to implement rather complex coordination protocols. Among these channels are (i) the **Sync** channel, which is a directed channel that accepts a data item through its source end if it can instantly dispense it through its sink end; (ii) the **LossySync** channel, which always accepts a data item through its source end and tries to instantly dispense it through the sink end. If this is not possible, the data item is lost; (iii) the **SyncDrain** channel, which is a channel with two source ends that accept data simultaneously and loses them subsequently; (iv) the **AsyncDrain** channel, which accepts data items only through one of its two source channel ends at a moment in time and loses it; and (v) the **FIFO** channel, which is an asynchronous channel with a buffer of capacity one. Additionally, there are channels for data manipulation. For instance, the **Filter** channel always accepts a data item at its source end and synchronously passes or loses it depending on whether or not the data item matches a certain predefined pattern or data constraint. Finally, the **Transform** channel applies a user-defined function to the data item received at its source end and synchronously yields the result at its sink end.

Channels can be joined together using nodes. A node can be a source, a sink or a mixed node, depending on whether all of its coinciding channel ends are source ends, sink ends or a combination of both. Source and sink nodes together form the boundary nodes of a connector, allowing interaction with its environment. Source nodes act as synchronous replicators, and sink nodes as non-deterministic mergers. A mixed node combines these two behaviors by atomically consuming a data item from one of its sink ends at the time and replicating it to all of its source ends.

The basic set of Reo channels can be extended to enable modeling of specific features of service communication. Apart from functional aspects, channels can differ at the level of their non-functional characteristics. In quantitative Reo [3], channels are characterized by a set of associated QoS parameters such as data transfer delays or cost.

In this paper, we consider Reo channels in presence of internal coordination and data transfer delays. Roughly, by internal coordination delay we mean the time that it takes a channel to decide whether or not it can accept to satisfy the I/O request at its ends. By data transfer delay we mean the time needed

to deliver a data item accepted by the source end of a channel to its sink end for the Sync channels, from the channel source end to its buffer and from the buffer to its sink end for the FIFO channels, or accept and destroy a data item for SyncDrain, AsyncDrain and LossySync channels.

An informal description of Reo given above is rather incomplete and ambiguous. The semantics of any Reo connector can be understood only in terms of a specific semantic model and its appropriate translation into that model.

The most basic model expressing the semantics of Reo formally is constraint automata [2]. Transitions in a constraint automaton are labeled with sets of ports that fire synchronously, as well as with data constraints on these ports. The constraint automata-based semantics for Reo is compositional, meaning that the behavior of a complex Reo circuit can be obtained from the semantics of its constituent parts using the product operator. Furthermore, the hiding operator can be used to abstract from unnecessary details such as dataflow on the internal ports of a connector.

Definition 1 (Constraint Automaton (CA)). *A constraint automaton $\mathcal{A} = (S, \mathcal{N}, \rightarrow, s_0)$ consists of a set of states S , a set of port names \mathcal{N} , a transition relation $\rightarrow \subseteq S \times 2^{\mathcal{N}} \times DC \times S$, where DC is the set of data constraints over a finite data domain $Data$, and an initial state $s_0 \in S$.*

We write $q \xrightarrow{N:g} p$ instead of $(q, N, g, p) \in \rightarrow$. Table 1 shows our graphical notation for the basic Reo channels and nodes together with their constraint automata semantics. The behavior of any Reo circuit can be obtained by computing the product of these automata which can be defined using the notion of a port synchronization function [5].

Definition 2. *Let $\mathcal{A}_1 = (S_1, \mathcal{N}_1, \rightarrow_1, s_0^1)$, $\mathcal{A}_2 = (S_2, \mathcal{N}_2, \rightarrow_2, s_0^2)$ be two constraint automata with disjoint sets of port names \mathcal{N}_1 and \mathcal{N}_2 , respectively. A port synchronization function $\gamma: \mathcal{N} \rightarrow \mathcal{N}_1 \times \mathcal{N}_2$ is defined as $\gamma(n) = (\gamma_1(n), \gamma_2(n))$ through the pair of injective functions $\gamma_1: \mathcal{N} \rightarrow \mathcal{N}_1$ and $\gamma_2: \mathcal{N} \rightarrow \mathcal{N}_2$ that map port names from a new set \mathcal{N} into port names from the sets \mathcal{N}_1 and \mathcal{N}_2 .*

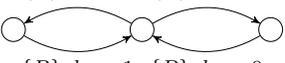
In the above definition, “new set” means $\mathcal{N} \cap (\mathcal{N}_1 \cup \mathcal{N}_2) = \emptyset$. Observe that the disjointness of \mathcal{N}_1 and \mathcal{N}_2 and the injectivity of the total functions γ_1 and γ_2 confine the cardinality of \mathcal{N} such that $0 \leq |\mathcal{N}| \leq \min(|\mathcal{N}_1|, |\mathcal{N}_2|)$. The exact definition of γ_1 or γ_2 , then, uniquely defines \mathcal{N} . Intuitively, $\gamma(n) = (x, y)$ represents a renaming of $x \in \mathcal{N}_1$ and $y \in \mathcal{N}_2$ to the same common element $n \in \mathcal{N}$. In the context of the port synchronization function γ , we write \mathcal{N}'_1 for $\mathcal{N}_1 \setminus \gamma_1[\mathcal{N}]$ and \mathcal{N}'_2 for $\mathcal{N}_2 \setminus \gamma_2[\mathcal{N}]$. If, for subsets $N_1 \subseteq \mathcal{N}_1$, $N_2 \subseteq \mathcal{N}_2$, it holds that $\gamma_1^{-1}[N_1] = \gamma_2^{-1}[N_2]$ we write

$$N_1 \mid_{\gamma} N_2 = (N_1 \cap \mathcal{N}'_1) \cup \gamma_1^{-1}[N_1] \cup (N_2 \cap \mathcal{N}'_2). \quad (1)$$

This means that $N_1 \mid_{\gamma} N_2$ is the union $N_1 \cup N_2$ with the parts of N_1 and N_2 that are identified via γ_1 and γ_2 replaced by the shared names $\gamma_1^{-1}[N_1] = \gamma_2^{-1}[N_2]$.

Also, for a constraint g , we write $\gamma(g)$ for the formula obtained by replacing the port names in $\gamma_1[\mathcal{N}] \subseteq \mathcal{N}_1$ and $\gamma_2[\mathcal{N}] \subseteq \mathcal{N}_2$ by the corresponding name in \mathcal{N} .

Table 1. Graphical notation and semantics for channels and nodes

Primitive	Notation	Constraint automaton
Sync	$A \longrightarrow B$	$\circ \rightleftarrows \{A, B\} \ d_A = d_B$
LossySync	$A \dashrightarrow B$	$\{A\} \rightleftarrows \{A, B\} \ d_A = d_B$
SyncDrain	$A \longrightarrow \bullet B$	$\circ \rightleftarrows \{A, B\}$
AsyncDrain	$A \bullet \longleftarrow B$	$\{B\} \rightleftarrows \{A\}$
FIFO	$A \boxed{\longrightarrow} B$	$\{A\} \ d_A = 1 \ \{A\} \ d_A = 0$  $\{B\} \ d_B = 1 \ \{B\} \ d_B = 0$
Filter	$A \rightsquigarrow B$	$\{A\} \neg \text{expr}(d_A) \rightleftarrows \{A, B\} \ \text{expr}(d_A) \wedge d_A = d_B$
Transform	$A \dashrightarrow B$	$\circ \rightleftarrows \{A, B\} \ d_B = f(d_A)$
Merger	$\begin{array}{c} A \\ B \end{array} \searrow C$	$\{A, C\} \ d_A = d_C \rightleftarrows \{B, C\} \ d_B = d_C$
Replicator	$A \searrow \begin{array}{c} B \\ C \end{array}$	$\circ \rightleftarrows \{A, B, C\} \ d_A = d_B = d_C$

Definition 3. For two constraint automata $\mathcal{A}_1 = (S_1, \mathcal{N}_1, \rightarrow_1, s_0^1)$ and $\mathcal{A}_2 = (S_2, \mathcal{N}_2, \rightarrow_2, s_0^2)$ and the port synchronization function $\gamma : \mathcal{N} \rightarrow \mathcal{N}_1 \times \mathcal{N}_2$ with $\gamma_1 : \mathcal{N} \rightarrow \mathcal{N}_1$ and $\gamma_2 : \mathcal{N} \rightarrow \mathcal{N}_2$, the constraint automaton $\mathcal{A}_1 \bowtie_{\gamma} \mathcal{A}_2$, called the γ -synchronous product of \mathcal{A}_1 and \mathcal{A}_2 , is given by $\mathcal{A}_1 \bowtie_{\gamma} \mathcal{A}_2 = (S_1 \times S_2, \mathcal{N}', \rightarrow, \langle s_0^1, s_0^2 \rangle)$ where $\mathcal{N}' = \mathcal{N}'_1 |_{\gamma} \mathcal{N}'_2$ and the transition relation \rightarrow is determined by the following rules:

$$\frac{s_1 \xrightarrow{N_1, g_1}_1 t_1 \quad N_1 \subseteq \mathcal{N}'_1}{\langle s_1, s_2 \rangle \xrightarrow{N_1, g_1} \langle t_1, s_2 \rangle} \quad \frac{s_2 \xrightarrow{N_2, g_2}_2 t_2 \quad N_2 \subseteq \mathcal{N}'_2}{\langle s_1, s_2 \rangle \xrightarrow{N_2, g_2} \langle s_1, t_2 \rangle} \quad (2)$$

and

$$\frac{s_1 \xrightarrow{N_1, g_1}_1 t_1 \quad s_2 \xrightarrow{N_2, g_2}_2 t_2 \quad \gamma_1^{-1}(N_1) = \gamma_2^{-1}(N_2)}{\langle s_1, s_2 \rangle \xrightarrow{N_1 |_{\gamma} N_2, \gamma(g_1 \wedge g_2)} \langle t_1, t_2 \rangle} \quad (3)$$

In the above setting, for a port $n \in \mathcal{N}$, the idea is that the ports $x = \gamma_1(n) \in \mathcal{N}_1$ and $y = \gamma_2(n) \in \mathcal{N}_2$ synchronize. Thus, either x and y both have flow or x and y both have no flow, expressed as n having flow or no flow, respectively. The resulting automaton, the so-called synchronized product automaton $\mathcal{A}_1 \bowtie_{\gamma} \mathcal{A}_2$, follows the flow of \mathcal{A}_1 and \mathcal{A}_2 , based on the first two rules for the transition relation, but requires the flow on its ports in \mathcal{N} to be agreed upon by both \mathcal{A}_1 and \mathcal{A}_2 .

Constraint automata in their basic form are not expressive enough to capture all interesting behavior in Reo. In particular, they cannot express the behavior of so-called context dependent channels. A basic example of such a channel is a

LossySync channel that loses a data item only if the environment or subsequent channels are not ready to consume it. Numerous models have been proposed to overcome this and other problems. However, due to space limits we cannot provide their detailed discussion in this paper.

The problem of expressing the behavior of Reo circuits is orthogonal to the problem of estimating the end-to-end quality of the communication protocol that they implement. The existing semantic models, most notably, constraint automata, have been extended with the information to capture the QoS characteristics of the channels and their composition metrics [3,4]. However, these extensions assume that the QoS characteristics do not affect the behavior of a circuit and simply assign QoS labels to the transitions of the basic automata models. In the next section, we argue that data transfer delays are important for circuit behavior and accommodating them properly requires an appropriate formal model.

3 Motivation

As mentioned before, currently variants of constraint automata extended with labels representing QoS characteristics are used to give formal semantics to QoS-aware Reo. These automata are defined with the help of Q-algebra introduced initially by Chothia and Kleijn [6]. A Q-algebra is an algebraic structure $R = (C, \oplus, \otimes, ||, \mathbf{0}, \mathbf{1})$ where C is the domain of R and represents a set of QoS values. The operation \oplus induces a partial order on the domain of R and is used to define a preferred value of a QoS dimension, \otimes is an operator for the sequential channel composition, while $||$ is an operator for the parallel composition. For example, the Q-algebra corresponding to the circuit execution time is defined as follows: $(\mathbb{R}_{\geq} \cup \{\infty\}, \min, +, \max, \infty, 0)$. Taking into account this definition, a Quantitative CA (QCA) [3] is an extended CA $\mathcal{A} = (S, S_0, \mathcal{N}, E, R)$ where the transition relation E is a finite subset of $\cup_{N \in \mathcal{N}} S \times \{N\} \times DC(N) \times C \times S$ and $R = (C, \oplus, \otimes, ||, \mathbf{0}, \mathbf{1})$ is a labeled Q-algebra with domain C .

QCA were introduced to enable the estimation of QoS of compound circuits given the QoS parameters of their constituent channels. However, as our example shown in Figure 1(a) illustrates, this model does not allow us to precisely compute the data transfer delays in synchronous regions. According to the definition of the QCA and Q-algebra for the execution time, the delay for the barrier synchronization connector equals $\max(t_1, t_2, t_3, t_4, t_5)$ while the real data transfer time cannot be smaller than the sum of the delays of the two pairs of Sync channels composed sequentially. Assuming that the whole transaction does not finish before the SyncDrain channel destroys the data consumed through its source ports, we conclude that the delay equals $\max(t_1 + \max(t_2, t_3), t_4 + \max(t_3, t_5))$.

As shown in [4], data transfer delays in Reo circuits can be computed given the information about its topology and the presumed dataflow semantics. However, currently there are no automata-based semantic models for Reo that supports such a computation in the compositional manner.

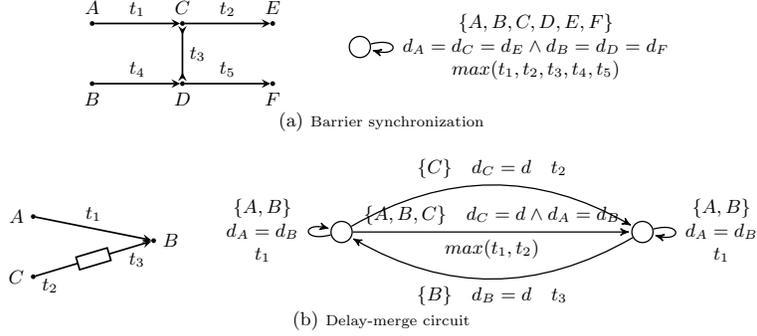


Fig. 1. Motivating examples

Another drawback of the constraint automata semantics for Reo is that it forces the synchronization of independent concurrent transactions with different durations. This problem arises from the fact that only one transition on constraint automaton can be enabled at the same time. Assuming that data transfer through a synchronous region of a circuit is not instantaneous as in the basic Reo model, dataflow on some other parts of the circuit can be initiated during this time. For example, the constraint automaton for the circuit shown in Figure 1(b) implies that while the FIFO channel accepts data through the port C , no other transition can be triggered. Imagine that the delay t_2 is much bigger than t_1 . This means that the circuit will not transfer data through the channel $\text{Sync}(A, B)$ until the port C finishes to accept data. However, the circuit should allow data transfer through the Sync channel at any time when the port B is not occupied. As this example illustrates, (Q)CA do not show all possible behaviors of Reo with data transfer delays.

4 Action Constraint Automata

In this section, we introduce a new model, called *action constraint automata*, that provides a valid semantic model for Reo coordination networks in the presence of time delays. This model is essentially a labeled transition system (LTS) with data and synchronization constraints. However, in contrast to constraint automata in their classic form, we distinguish several kinds of actions which are triggered on channel ports to signal the state changes of the channel. Formally, an action constraint automaton is defined as follows:

Definition 4 (Action Constraint Automaton (ACA)). *An action constraint automaton $\mathcal{A} = (S, \mathcal{N}, \rightarrow, s_0)$ consists of a set of states S , a set of action names \mathcal{N} derived from a set of port names \mathcal{M} and a set of admissible action types \mathcal{T} , a transition relation $\rightarrow \subseteq S \times 2^{\mathcal{N}} \times DC \times S$, where DC is the set of data constraints over a finite data domain $Data$, and an initial state $s_0 \in S$.*

We introduce an injective function $act : \mathcal{M} \times \mathcal{T} \rightarrow \mathcal{N}$ to define action names for each pair of a port name and an action type observed on the port. For example, the function $act(m, \alpha) = \alpha \bullet m \mid m \in \mathcal{M}, \alpha \in \mathcal{T}$, where \bullet is a standard lexical concatenation operator, can be used to obtain a set of unique action names given sets of distinctive Reo port names and types of observable actions.

Analogously to the constraint automata, we define the action synchronization function $\gamma : \mathcal{N} \rightarrow \mathcal{N}_1 \times \mathcal{N}_2$ through a pair of injective functions $\gamma_1 : \mathcal{N} \rightarrow \mathcal{N}_1$, $\gamma_2 : \mathcal{N} \rightarrow \mathcal{N}_2$ from a new set of action names \mathcal{N} into \mathcal{N}_1 and \mathcal{N}_2 . Given such function, we can define the product operator for ACA.

Definition 5 (Product of Action Constraint Automata). *For two action constraint automata $\mathcal{A}_1 = (S_1, \mathcal{N}_1, \rightarrow_1, s_0^1)$ and $\mathcal{A}_2 = (S_2, \mathcal{N}_2, \rightarrow_2, s_0^2)$ and the action synchronization function $\gamma : \mathcal{N} \rightarrow \mathcal{N}_1 \times \mathcal{N}_2$ with $\gamma_1 : \mathcal{N} \rightarrow \mathcal{N}_1$ and $\gamma_2 : \mathcal{N} \rightarrow \mathcal{N}_2$, the action constraint automaton $\mathcal{A}_1 \boxtimes_{\gamma} \mathcal{A}_2$, called the γ -synchronization product of \mathcal{A}_1 and \mathcal{A}_2 , is given by $\mathcal{A}_1 \boxtimes_{\gamma} \mathcal{A}_2 = (S_1 \times S_2, \mathcal{N}'_1 \mid_{\gamma} \mathcal{N}'_2, \rightarrow, \langle s_0^1, s_0^2 \rangle)$ where the transition relation \rightarrow is determined by the following rules:*

$$\frac{s_1 \xrightarrow{N_1, g_1} t_1 \quad N_1 \subseteq \mathcal{N}'_1}{\langle s_1, s_2 \rangle \xrightarrow{N_1, g_1} \langle t_1, s_2 \rangle} \quad \frac{s_2 \xrightarrow{N_2, g_2} t_2 \quad N_2 \subseteq \mathcal{N}'_2}{\langle s_1, s_2 \rangle \xrightarrow{N_2, g_2} \langle s_1, t_2 \rangle} \quad (4)$$

and

$$\frac{s_1 \xrightarrow{N_1, g_1} t_1 \quad s_2 \xrightarrow{N_2, g_2} t_2 \quad \gamma_1^{-1}(N_1) = \gamma_2^{-1}(N_2)}{\langle s_1, s_2 \rangle \xrightarrow{N_1 \mid_{\gamma} N_2, \gamma(g_1 \wedge g_2)} \langle t_1, t_2 \rangle} \quad (5)$$

Transitions where the set of actions N is non-empty are called *visible*, while transitions with the empty action-set are called *hidden*. In a hidden transition, none of the actions is visible and the data constraints appear as unknown from outside. We denote hidden transitions by the label τ . Such transitions can be witnessed only by the change of a state in an automaton. Taking this into account, the hiding operator on ACA is defined as follows:

Definition 6 (Action hiding). *The action hiding operator takes as input an ACA $\mathcal{A} = (S, \mathcal{N}, \rightarrow, s_0)$ and a non-empty set of actions $K \subseteq \mathcal{N}$. The result is an ACA $hide(\mathcal{A}, K) = (S, \mathcal{N} \setminus K, \rightarrow, s_0)$ where*

- $q \xrightarrow{N', g'} p$ iff there exists a transition $q \xrightarrow{N, g} p$ such that $N \setminus K \neq \emptyset$ and $g' = \bigvee_{\delta \in DA(K)} g[d_A / \delta.A \mid A \in K]$, where $g[d_A / \delta.A \mid A \in K]$ denotes the syntactic replacement of all occurrences of d_A in g for $A \in K$ with $\delta.A$.
- $q \xrightarrow{\tau} p$ iff there exists a transition $q \xrightarrow{N, g} p$ such that $N \setminus K = \emptyset$.

A port hiding can be achieved by hiding of all actions observed on this port. In turn, a node hiding is the result of the hiding of all ports coincident on the node.

Note that constraint automata represent a subclass of action constraint automata with only one action observed on each port. This action represents the fact that the data flow through this port. The synchronization function used in the definition of constrain automata implies a renaming of joint channel/node ports while here it is used for renaming of actions that are observed simultaneously.

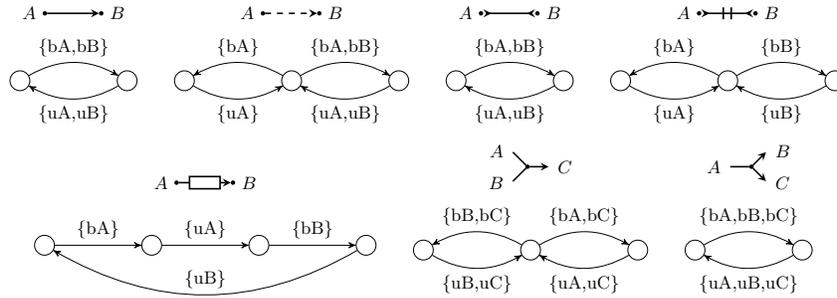


Fig. 2. Semantics of channels and nodes with port blocking

5 Dataflow Modeling

In this section, we introduce an ACA-based model for representing the semantics of Reo with data transfer delays.

Since some time is required by a channel for its internal coordination and to transfer data, it may happen that the channel is still busy while other requests arrive at the source ports of the circuit. There is no reason why the channels that are not busy at the moment should not process the arrived requests. However, as our motivating examples have shown, CA do not allow such behavior. To provide a more expressive model for Reo that fixes the aforementioned problem, we consider two actions, namely, a ‘block’ action and its dual an ‘unlock’ action which are used to establish port communication within a single transaction and release channel ports involved in such a transaction, respectively.

Table 5 shows the semantics of the basic Reo channels with presumable data transfer delays in terms of ACA for the set of action types $\mathcal{T}_1 = \{b, u\}$, where b stands for the ‘block’ and u stands for the ‘unlock’ actions. Since our focus is on synchronization constraints, we omit the data constraints in this figure to simplify the presentation. In their initial states, channels do not accept or dispense data. To show that the Sync channel with the source end A and the sink end B is ready to accept and dispense data, ‘block’ actions bA and bB occur simultaneously. After the data transfer is finished, the channel returns to its initial state when both ports are released and ‘unlock’ actions uA and uB are observed. For the LossySync channel the behavior is similar with the exception that the data can be lost after entering the channel. In this case, only the channel source port A is involved in the communication. For the SyncDrain channel we require that ports are blocked and unblocked simultaneously, while for the AsyncDrain only one of the source ends A or B can be involved in the communication at each particular moment in time. Finally, for the empty FIFO channel, first the data is stored in the buffer through the source port A , then the buffer is emptied through the sink port B .

Figure 3 shows the semantics of the delay-merge circuit and the barrier synchronization of Figure 1 obtained using the product and hiding operators on the

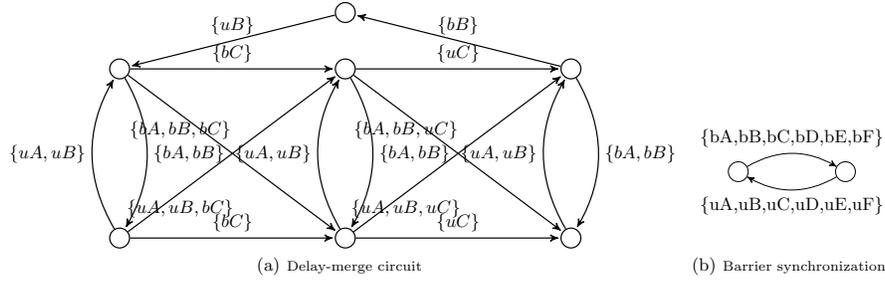


Fig. 3. Semantics of the delay-merge circuit with port blocking

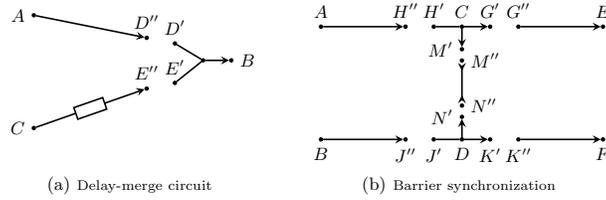


Fig. 4. Motivating examples: decomposed circuits

ACA with the set of action types \mathcal{T}_1 . Since in our definitions of CA and ACA we require all port names to be different, we apply these operators to the channels and nodes in the decomposed versions of these circuits shown in Figure 4. Observe that after blocking the port C in the delay-merge circuit, the system can trigger transitions defined by the actions $\{bA, bB\}$ and $\{uA, uB\}$. This means that the Sync channel can transfer data while the FIFO channel is writing data into its buffer. Thus, our new ACA semantic model resolves the problem of synchronization of independent concurrent transactions with different durations manifested by the CA model.

The automaton for the barrier synchronization consists of two states that represent the situations where all channels in the synchronous region are free and where all channels are transferring data. The circuit can stay in the second state for the duration of time needed to finish the data transfer through all five synchronous channels. However, the model does not show the actual flow of data within the circuit. Thus, we cannot use this model for computing time delays in synchronous circuits given delays for its individual channels.

To solve this problem, in addition to the ‘block’ and ‘unblock’ actions, we introduce ‘start’ and ‘finish’ actions which are used to represent the start and the end of dataflow through a blocked channel port. Thus, we use the set of action types $\mathcal{T}_2 = \{b, s, f, u\}$, where b stands for the ‘block’, s for the ‘start’, f for the ‘finish’ and u for the ‘unblock’ action types. The sequence of the aforementioned four actions is observed on each Reo port. Before the start of each transition, ports participating in this transition must be blocked. Then, the data transfer

starts. After some time t , which represents the delay in the channel, the ‘finish’ action occurs to signal that the data transfer is over. Finally, the ‘unlock’ action releases the channel port, subsequent to which it can be coopted to perform another communication. The time between a ‘block’ action and a subsequent ‘start’ action on the same port represents the overhead necessary for the set-up of the internal coordination before the data transfer can happen. Analogously, the time between a ‘finish’ and an ‘unlock’ represents the overhead of dismantling the data transfer set-up. Table 2 shows the semantics of the basic Reo channels with explicit modeling of internal coordination and dataflow within each channel. After blocking actions have occurred in the Sync channel, both its ports start to accept data. This is represented by the simultaneous occurrence of the actions sA and sB . Similarly, after the data transfer is finished, actions fA and fB are observed. For the SyncDrain channel, as usual, we require that its ports are blocked and unblocked simultaneously, while the actual data transfer through the two ports start and end independently, i.e., all interleavings of action pairs (sA, fA) and (sB, fB) are allowed. In principle, it is also possible to consider more restricting versions of the SyncDrain channel where both source ports must synchronize on starting and/or finishing of their data transfer.

The semantics of the Merger and the Replicator nodes is defined in a similar way. We assume that, in contrast to channels, the data transfer through a node is instantaneous, i.e., dataflow starts and finishes at the same time. For the scenarios where the time for data replication is significant and cannot be neglected, automata with two different transitions to signal the start and the end of dataflow should be used.

By synchronizing ‘finish’ actions observed on sink ends with ‘start’ actions observed on the source ends, we can model sequential flow of data in the synchronous regions. Given two action constraint automata \mathcal{A}_1 and \mathcal{A}_2 for each pair $X \in \mathcal{M}_1, Y \in \mathcal{M}_2$ of joint ports, where X is a sink port, and Y is a source port, the following pairs of actions happen synchronously:

$$\{(act(X, b), act(Y, b)), (act(X, u), act(Y, u)), (act(X, f), act(Y, s))\}.$$

This approach is compliant with the two principles introduced in [4], namely, that (i) a data-flow in a channel takes place from its input port to its output port, and (ii) mixed nodes receive and send data instantaneously.

Figure 5 shows the ACA with the set of action types \mathcal{T}_2 for the delay-merge circuit obtained as a product of ACA for two channels and the merge node with an action synchronization function defined by the following set of mappings:

$$\begin{aligned} \{(bD'', bD') \rightarrow bD, (uD'', uD') \rightarrow uD, (fD'', sD') \rightarrow fD, \\ (bE'', bE') \rightarrow bE, (uE'', uE') \rightarrow uE, (fE'', sE') \rightarrow fE\}. \end{aligned}$$

Observe that, similarly to the previous example, in any state where port C is occupied (blocked, started to or finished with the transfer of data, but not yet unblocked), the Sync channel can be involved in an independent communication.

Table 2. Semantics of channels and nodes with explicit dataflow

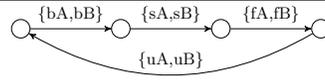
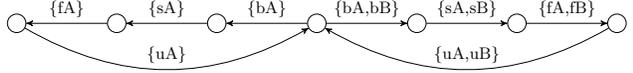
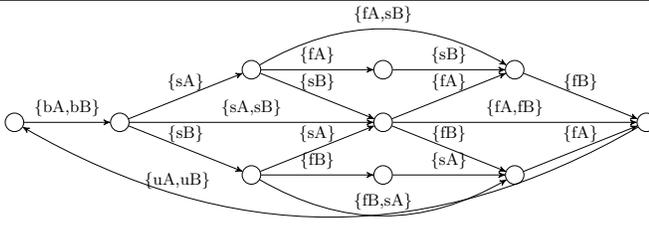
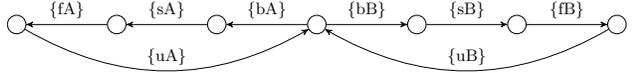
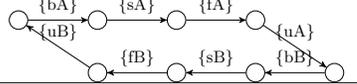
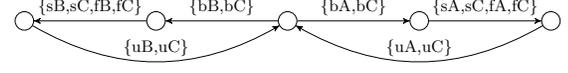
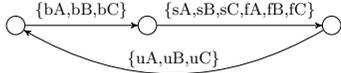
Primitive	Dataflow automaton
$A \longrightarrow B$	
$A \dashrightarrow B$	
$A \longrightarrow B$	
$A \rightleftarrows B$	
$A \square B$	
$A \begin{matrix} \nearrow \\ \searrow \end{matrix} C$	
$A \begin{matrix} \nearrow \\ \searrow \\ \swarrow \end{matrix} B, C$	

Figure 6 shows the ACA for the barrier synchronization circuit obtained using the action synchronization function defined by the following mappings:

$$\begin{aligned}
& \{(bH'', bH') \rightarrow bH, (uH'', uH') \rightarrow uH, (fH'', sH') \rightarrow fH, \\
& (bG', bG'') \rightarrow bG, (uG', uG'') \rightarrow uG, (fG', sG'') \rightarrow sG, \\
& (bM', bM'') \rightarrow bM, (uM', uM'') \rightarrow uM, (fM', sM'') \rightarrow sM, \\
& (bN', bN'') \rightarrow bN, (uN', uN'') \rightarrow uN, (fN', sN'') \rightarrow sN, \\
& (bJ'', bJ') \rightarrow bJ, (uJ'', uJ') \rightarrow uJ, (fJ'', sJ') \rightarrow fJ, \\
& (bK', bK'') \rightarrow bK, (uK', uL'') \rightarrow uK, (fK', sK'') \rightarrow sK\}
\end{aligned}$$

and the set of hidden actions

$$\{fH', sH'', fG'', sG', sM', fM'', sN', fN'', fJ', sJ'', sK', fK''\}.$$

In this model, after blocking all ports, the source ports A and B start to accept data (either separately or simultaneously). Similarly, labels of further transitions show on which ports the dataflow starts and finishes. Observe that the end

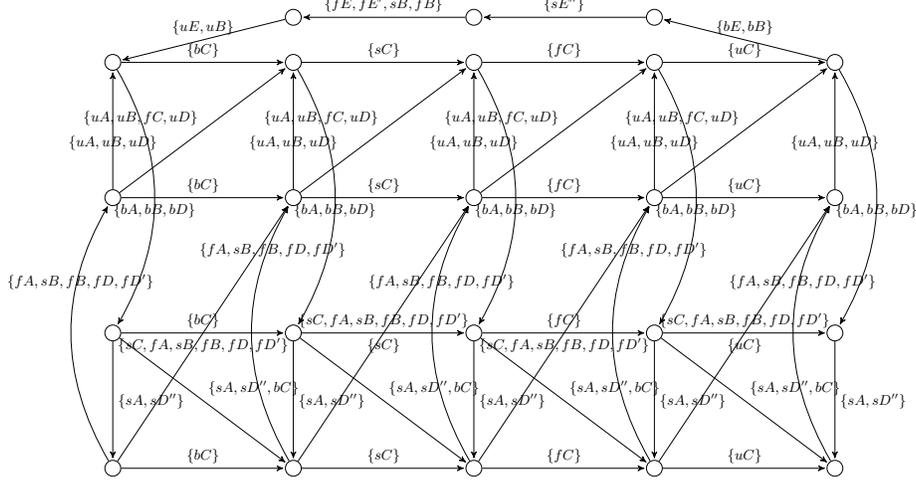


Fig. 5. Semantics of the delay-merge circuit with explicit dataflow

of dataflow on ports preceding the SyncDrain (external ports A and B , and internal ports H and J) coincides with the start of the flow on ports following the SyncDrain (external ports E and F , and internal ports M , N , G and K). Thus, this model is capable of capturing the stepwise dataflow progress through synchronous regions.

Among all the states of these automata we may be interested to locate states in which all channels are idle and free to communicate. Formally, such states are characterized by the condition $\forall A \in \mathcal{M}, act(A, b) \in N[r] \Rightarrow act(A, u) \in N[r]$, where $N[r] = \bigcup N_i \mid s_i \xrightarrow{N_i, d_i} s_{i+1}$ is a set of actions of some automaton run $r = s \xrightarrow{N_0, d_0} s_1 \xrightarrow{N_1, d_1} s_2 \xrightarrow{N_2, d_2} s_3 \dots$. Such states correspond to network configurations defined by the basic CA.

6 Model Analysis and Tool Support

The goal of the introduced semantic model for Reo is to provide a sound mathematical basis for the implementation of analysis tools. The set of potentially useful tools includes but is not limited to converters that generate the automata-based models given graphical Reo circuits, model checking tools able to verify the validity of system properties expressed in some kind of formal logic, simulation engines that allow us to validate and evaluate the performance of a model, and model-based code and test generation tools. The development of such tools from scratch is far from trivial and very time consuming. An alternative approach is to convert our model to a format acceptable by existing analysis tools. To enable model checking of Reo, we generally rely on the mCRL2 framework.

mCRL2 is a specification language based on the process algebra ACP. The basic notion in mCRL2 is the action. Actions represent atomic events and can be

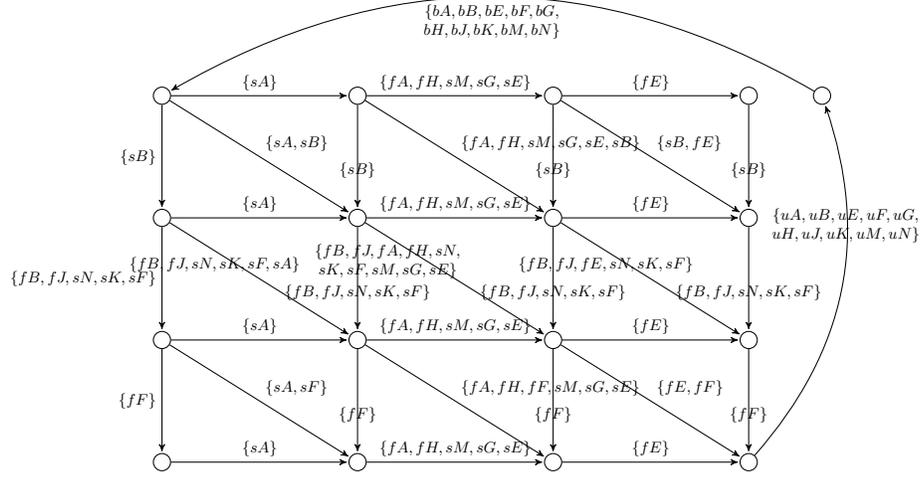


Fig. 6. Semantics of the barrier synchronization circuit with explicit dataflow

parameterized with data. Actions in mCRL2 can be synchronized using the synchronization operator $|$. Synchronized actions are called multiactions. Processes are defined by process expressions, which are compositions of actions and multiactions using a number of operators. The basic operators include (i) *deadlock* or *inaction* δ , (ii) *alternative composition* $p + q$, (iii) *sequential composition* $p \cdot q$, (iv) *conditional* operator or *if-then-else* construct $c \rightarrow p \diamond q$ where c is a boolean expression, (v) *summation* $\sum_{d:D} p$ used to quantify over a data domain D , (vi) *at* operator $a@t$ indicating that multiaction a happens at time t , (vii) *parallel composition* $p \parallel q$ yielding interleavings of the actions in p and q , (viii) *encapsulation* $\partial_H(p)$, where H is a set of action names that are not allowed to occur, (ix) *renaming* operator $\rho_R(p)$, where R is a set of renamings of the form $a \rightarrow b$ and (x) *communication* operator $\Gamma_C(p)$, where C is a set of communications of the form $a_0 | \dots | a_n \mapsto c$, which means that every group of actions $a_0 | \dots | a_n$ within a multiaction is replaced by c . Moreover, the mCRL2 language provides a number of built-in datatypes (e.g., boolean, natural, integer) with predefined standard arithmetic operations and a datatype definition mechanism to declare custom types (called also sorts).

The mCRL2 toolset includes a tool for converting mCRL2 code into a linear process specification (LPS), which is a compact symbolic representation of LTS to speed up subsequent manipulations, a tool for generating explicit LTS from LPS, tools for optimizing and visualizing LTS, and many other useful facilities. For model checking, system properties are specified as formulae in a variant of the modal μ -calculus extended with regular expressions, data and time. In combination with an LPS such a formula is transformed into a parameterized boolean equation system and can be solved with the appropriate tools from the toolset. Analysis at the level of LTS, in particular, deadlock detection or checking

Table 3. mCRL2 encoding for channels and nodes

$\text{Sync} = bA bB \cdot sA sB \cdot fA fB \cdot uA uB \cdot \text{Sync}$ $\text{LossySync} = (bA bB \cdot sA sB \cdot fA fB \cdot uA uB + bA \cdot sA \cdot fA \cdot uA) \cdot \text{LossySync}$ $\text{SyncDrain} = bA bB \cdot ($ $sA \cdot (sB \cdot (fA \cdot fB + fB \cdot fA + fA fB) + fA \cdot sB \cdot fB + sB fA \cdot fB) +$ $sB \cdot (sA \cdot (fA \cdot fB + fB \cdot fA + fA fB) + fB \cdot sA \cdot fA + sA fB \cdot fA) +$ $sA sB \cdot (fA \cdot fB + fB \cdot fA + fA fB)) \cdot uA uB \cdot \text{SyncDrain}$ $\text{AsyncDrain} = (bA \cdot sA \cdot fA \cdot uA + bB \cdot sB \cdot fB \cdot uB) \cdot \text{AsyncDrain}$ $\text{FIFO} = \text{isEmpty}(f) \rightarrow bA \cdot sA \cdot fA \cdot uA \cdot \text{FIFO}(\text{full})$ $\diamond bB \cdot bB \cdot sB \cdot fB \cdot uB \cdot \text{FIFO}(\text{empty})$
$\text{Merger} = (bA bC \cdot sA sC fA fC \cdot uA uC + bB bC \cdot sB sC fB fC \cdot uB uC) \cdot \text{Merger}$ $\text{Replicator} = bA bB bC \cdot sA sB sC \cdot fA fC \cdot uA uC \cdot \text{Replicator}$

of the presence or absence of certain actions, is also possible. A detailed overview can be found at the mCRL2 web site².

We employed the mCRL2 toolset to generate state spaces for graphical Reo circuits and further model check them. mCRL2 models for Reo circuits are generated in the following way [7]: observable actions (i.e., dataflow on the channel ends in the basic CA model) are represented as atomic actions, while data items observed at these ports are modeled as parameters of these actions. Analogously, we introduce a process for every node and actions for all channel ends meeting at the node. A global custom sort *Data* and the mCRL2 summation operator are used to model the input data domain and iterate over it while specifying data constraints imposed by channels.

The availability of the synchronization operator and multications in mCRL2 makes the translation of CA and ACA to the process algebra mCRL2 straightforward: we simply synchronize the joint ports in CA and the simultaneously observed actions in ACA. Table 3 shows the mCRL2 encodings for the basic Reo channels and nodes according to the semantic model introduced in this paper. Since data support in the new translation is analogous to the case of the CA-based translation [7], we omit its discussion here and for simplicity show only the data-agnostic mapping. Note that the expression for the SyncDrain channel in the table is equivalent to

$$\text{SyncDrain} = bA|bB \cdot ((sA \cdot fA)|(sB \cdot fB)) \cdot uA|uB \cdot \text{SyncDrain};$$

However, the use of the parallel operator in mCRL2 is restricted because of the difficulties to linearize processes where such an operator occurs in the scope of the sequential, alternative, summation or synchronization operators.

As in the CA approach, we construct nodes compositionally out of the Merger and the Replicator primitives. Given process definitions for all channels and nodes, a joint process that models the complete Reo connector is built by forming a parallel composition of these processes and synchronizing the actions for the coinciding channel/node ends. Optionally, the mCRL2 hiding operator can

² www.mcr12.org/

be employed for abstracting the flow in internal nodes. Channel/node end synchronization is performed using two of the `mCRL2` operators: communication and encapsulation. For minimizing intermediate state spaces while generating the `mCRL2` specification, we exploit the structure of the circuit and build the process for the whole Reo connector in a stepwise fashion. In [5], we show that the operational semantics of the `mCRL2` specification obtained in this way is equivalent to the CA semantics of the Reo connector. This result applies to ACA as well.

7 Conclusions

In this paper, we discussed the formal semantic models for the channel-based coordination language Reo in the presence of coordination and data transfer delays. We argued that the existing semantic models do not reflect all possible behaviors in such circuits and are not suitable for the computation of end-to-end time delays in Reo circuits. To fix these problems, we proposed a more expressive model, *action constraint automata*, which represent the behavior of a circuit in terms of actions observed on its ports. The new model distinguishes transactional aspects of Reo from dataflow modeling, which is useful for the implementation of animation and simulation tools for Reo as well as the implementation of Reo-based service interaction protocols.

The presented work is a first step toward enabling performance analysis for service compositions and process models specified in Reo. We are going to define the quantitative version of the ACA and develop algorithms for computing time delays in the circuits, which are rather straightforward, but are not discussed here due to space limitation. We also plan to consider circuits with stochastic delays and develop a theory of quality preserving substitutability of channel-based connectors.

References

1. Arbab, F.: Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science* **14** (2004) 329–366
2. Baier, C., Sirjani, M., Arbab, F., Rutten, J.: Modeling Component Connectors in Reo by Constraint Automata. *Science of Computer Programming* **61** (2006) 75–113
3. Arbab, F., Chothia, T., Sun, M., Moon, Y.J.: Component connectors with QoS guarantees. In: *Proc. COORDINATION’ 07*. Volume 4467 of LNCS., Springer (2007) 286–304
4. Arbab, F., Chothia, T., van der Mei, R., Sun, M., Moon, Y., Verhoef, C.: From coordination to stochastic models of QoS. In: *Proc. Coordination’ 09*. Volume 5521 of LNCS., Springer (2009) 268–287
5. Kokash, N., Krause, C., de Vink, E.: Verification of context-dependent channel-based service models. In: *Proc. FMCO 2009*. LNCS, Springer (2010)
6. Chothia, T., Kleijn, J.: *Q-automata: Modelling the resource usage of concurrent components*, Elsevier (2007) 79–94
7. Kokash, N., Krause, C., de Vink, E.: Data-aware design and verification of service composition with Reo and `mCRL2`. In: *Proc. of SAC 2010*, ACM Press (2010) 2406–2413