Compositional Workflow Modeling with Priority Constraints

Behnaz Changizi[a,*], Natallia Kokash[b], Farhad Arbab[c], Leonid Makhnist[d]

[a] *Leiden Institute of Advanced Computer Science, Niels Bohrweg 1, Leiden, The Netherlands*
[b] *Peoples' Friendship University of Russia (RUDN University), 6 Miklukho-Maklaya St, Moscow, 117198, Russian Federation*
[c] *Centrum Wiskunde & Informatica, Science Park 123, Amsterdam, The Netherlands*
[d] *Brest State Technical University, Department of Higher Mathematics, Moskovskaya 267, 224017 Brest, Republic of Belarus*

Abstract

Priority is an important concept in Business Process Management (BPM), useful in the context of workflow patterns such as, e.g., cancelable and compensable tasks within business transactions. Unfortunately, the presence of priority in workflows makes them difficult to be analyzed formally by automated validation and verification tools. In the past, we demonstrated that the Reo coordination language can be successfully used for modeling, automatic validation and model checking of process models. In this paper, we propose a constraint-based approach to formalize priority in Reo. We introduce special channels to initiate, propagate, and block priority flows, define their semantics as constraints, and model priority propagation as a Constraint Satisfaction Problem (CSP). The semantic extension we propose in this paper enables workflow analysis in presence of priority constraints.

Keywords: Workflow modelling, Formal semantics, Transaction, Priority, Constraints, Coordination

*Corresponding author

**The publication has been prepared with the support of the "RUDN University Program 5-100" and funded by RFBR according to the research projects No. 12-34-56789 and No. 12-34-56789 (recipient: Natallia Kokash, formal semantics)

Email addresses: behnaz.changizi@gmail.com (Behnaz Changizi), kokosh-nv@rudn.ru (Natallia Kokash), farhad.arbab@cwi.nl (Farhad Arbab), hm@bstu.bu (Leonid Makhnist)

1. Introduction

Business Process Management (BPM) is an operational management approach that focuses on improving business processes. A business processes, i.e., a collection of important activities in an organization, is represented in the form of a workflow, an orchestrated and repeatable pattern of activities amenable to automated analysis and control. BPM systems [1, 2, 3, 4, 5] are widely used to provide means for automated process analysis such as model validation, transformation, simulation, visualization of Key Performance Indicators (KPIs), and reporting.

Despite the variety of BPM systems, their formal analysis commonly relies on Petri Nets [6, 7]. Business Process Modeling Notation (BPMN), a de-facto standard in business process modeling, has been mapped to Petri Nets to explain its semantics and enable model verification [5, 8]. The choice of Petri Nets as foundation for BPM over other formal methods, often more expressive or specialized [9, 10], is not surprising: hardly any model is as simple, intuitive, and naturally supports task traceability. However, while Petri net-based models enable automated process analysis, they lack a few desirable characteristics:

1. They cannot naturally represent semantics of component-based or service-based processes. Ideally, we would like to plug semantic models for individual components (often integrated dynamically at run time) to the semantic models of existing processes in a compositional way.
2. The classical Petri Nets are not expressive enough and often are extended (e.g., with colors [11], reset and inhibitor arcs [12], priority [13, 14] transitions) to enable meaningful process analysis. Such extensions change the semantics of the model and generate incompatible dialects of process-specification languages adopted by various tools.

An alternative formalization to express the semantics of BPMN models is the Reo coordination language [15]. Reo has been used to formalize semantics of BPMN, UML Activity and Sequence Diagrams [16], Business Process Execution Language (BPEL) fragments [17], represent transactional workflows [18], and model service orchestrations [19] and choreographies [20]. Reo allows composition of components and services in an intuitive way, hence, addressing the first issue mentioned above.

The open-ended nature of Reo allows us to introduce channels with specific properties. Similarly to Petri Nets, introducing new primitives in Reo often requires its semantics to be revised. Several dozen variations of semantic models for Reo have been proposed [21]. They vary from rather simple ones that cover basic Reo behavior (e.g., Constraint Automata (CA) [22]) to more complex models that cover specific behavioral aspects, e.g., context-sensitivity [23]. In some of these models, deriving the overall semantics of a system is computationally expensive. This hampers using the language for analyzing large real-world systems.

In [24], we proposed to model the semantics of Reo as a Constraint Satisfaction Problem (CSP). We defined data flow in a Reo network in the form of mathematical expressions on data observed at various nodes or ports. The main advantage of such representation is the possibility to use existing highly optimized constraint solvers [25, 26] to infer the behavior of a network given the semantics of its constituent parts.

Priority flow is an important aspect of process modeling not easily supported by existing formal semantics. Analyzing compensation and error handling requires a mechanism to express priority of flow alternatives over others. In this paper, we develop a theoretical foundation for priority flow modelling in Reo where the behavioral semantics of Reo circuits extended with priority-aware modelling primitives is expressed in terms of CSP. The paper revises and extends our initial ideas about applying Reo with priority flow control channels to model service orchestrations [27]. In this revision, we refine definitions, dedicate more space for case study analysis, and supplement the theory with an open-source implementation of a tool to generate CSP solutions for a given Reo circuit with priority flow.

This paper is organized as follows: In Section 2, we briefly describe the Reo coordination language. In Section 3, we introduce priority flow in Reo and proceed with constraint-based semantics for it. In Section 4, we extend our approach to support numeric priorities. In Section 5, we show the application of our constraint-based approach via two classes of connectors: a) priority-aware, and b) connectors with a large number of states. In Section 6, we overview related work. Finally, in Section 7, we conclude the paper and outline future work.

2. Reo

In BPM, an organizational process is not simply defined by the performed activities, but to a large extent by coordination among entities performing these activities: components, services, agents, stakeholders, etc. Hence, the behavior of a software system realizing a business process is not only defined by the functionality of its parts, but also by the "glue code" controlling their interaction. Writing and maintaining glue code is a tedious task, especially in complex systems wherein the size and rigidity of the glue code tend to grow over time. Coordination languages offer a manageable alternative for designing and maintaining the "glue code".

Reo [15, 28] is a channel-based coordination language for composition of software components and services. Using a small and open-ended set of predefined and user-defined constructs, Reo supports modeling of complex coordination behavior. The primitive constructs in Reo are *channels* and *nodes*. The composition of channels and nodes yields *connectors*. A channel is an atomic connector with two *ends* and a *constraint* that relates the flow of data at these ends. Channel ends are either *source* ends that read data into the channel or *sink* ends that write the data out. Channels can connect to each other through nodes.

There are two types of channel ends; therefore, three types of nodes can exist: *source nodes* where only source ends coincide, *sink nodes* where only sink ends coincide, and *mixed nodes* where both source and sink ends coincide. The mixed nodes of a connector are internal to the connector and not accessible for external data exchange. The source and sink nodes of a connector, collectively called its *boundary nodes* or *ports*, are used to connect to (the ports of) components to exchange data. A source node atomically replicates an incoming data items into all of its coincident channel ends, whenever they are all ready to accept. A sink node non-deterministically selects a data item out of one of its coincident channel ends and delivers it as its outgoing data item, leaving all other data items in its coincident channels intact. The behavior of a mixed node is an atomic combination of the behavior of a source node and that of a sink node: whenever all of its coincident source channels ends are ready to accept data items, it selects a data item out of one of its non-deterministically chosen coincident sink channel ends, and atomically replicates it into all of its source channel ends.

Sync  channel has a source and a sink end. It accepts data from its source iff its sink can dispense it simultaneously.

LossySync  channel has a source and a sink end. It reads a data item from its source and writes it simultaneously to its sink. If the sink end is not ready to accept the data item, the channel loses it.

SyncDrain  channel has two source ends and no sink end. It reads data from its two ends and discards it iff the ends are ready to interact simultaneously.

FIFO  channel has a source end, a sink end, and capacity for only one data item. If it is empty, the channel accepts a data item from its source end and buffers it. If it is full, it is ready to dispense data through its sink end. Because its one-capacity buffer is always either full or empty, both ends of the channel cannot interact simultaneously.

In addition to the channels, we use the standard set of Reo nodes and shortcuts for routing components:

Replicator  node has one source end and one or more sink ends. It replicates data coming from its source to its sinks simultaneously.

Merger  node has one or more source ends and one sink end. It chooses one of its ready to interact source ends non-deterministically, receives a data item through this end, and writes it to its sink end simultaneously.

Router  is a node representing a connector shortcut with equivalent behavior. It has one source end and a number of sink ends. It accepts a data item from its source and simultaneously replicates it on one of its non-deterministically chosen sink, which is ready to accept data.

CrossProduct  node has a number of source ends and a sink end. It accepts a data item from each source, forms a tuple of them in the counter-clock-wise order with respect to its sink, where it writes the tuple, simultaneously.

Two unique features of Reo are *compositionality*, which can be understood as the ability to derive the behavior of a circuit given behavioral semantics of its parts, and *propagation of synchrony*, which helps to instantly propagate an event on a Reo port to the rest of the circuit, i.e., like a switch in an electrical network can start or stop the flow in the network, an input/output event on

a Reo port can enable or disable control/data flow in (the synchronous parts of) a Reo circuit.

In [29, 18], the authors illustrated how to use Reo for modelling complex workflow patterns. Reo process models provide a rigorous foundation for process simulation. While Reo models do not give the answer to all challenges associated with process simulation [30], they enable accurate distributed process simulation with attention to synchronous vs asynchronous flow, blocking events, conditional data-based branching, and time delays.

3. Priority flow

Organizations run multiple processes which often compete for resources (e.g., operators, processing time). Even within a single process, different tasks and cases may compete for resources. Setting priority to tasks and cases helps organizations to resolve operational conflicts. For example, it is common to prioritize delayed process cases or, on the contrary, cancel stalled operations to improve overall KPIs. Although important, priority is rarely captured by simulation tools due to the lack of theoretical foundation for priority-aware modelling.

As a simple motivating example, we refer to BPMN, which prioritizes exception flow over normal flow. To enable verification and accurate simulation of Reo process models with exception flow, we propose to extend the basic set of Reo channels with the following channel types:

PrioritySync  is a synchronous channel that behaves like Sync but imposes priority on its flow. The data propagates through the connector (unless it is blocked), and can influence the non-deterministic choices in the containing connector by favoring flow alternatives that incorporate its ends.

BlockSourceSync  is similar to the Sync channel but blocks the propagation of priority from its source end towards its sink end.

BlockSinkSync  is similar to the Sync channel but stops propagation of priority from its sink end towards its source end.

BlockSync  represents a combination of BlockSourceSync and BlockSinkSync and stops the propagation of priority in both ways.

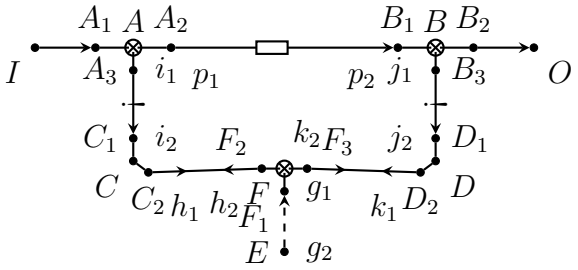


Figure 1: Decomposed Reo circuit to model process cancellation

Figure 1 shows an assembly of a Reo circuit to model process cancellation. The normal flow expects to pass through the input node I and output node O. The circuit includes two PrioritySync channels to prioritize the divergence of flow in the Router-type nodes A or B upon the arrival of the cancellation request at port E.

To enable correct derivation of behavioral semantics for Reo with priority, we introduce the concepts of *innate* and *acquired* priority. Both ends of PrioritySync have *innate* priority. When an end with *innate* priority connects to another end that has no priority, the new end will obtain *acquired* priority. When one end of a synchronous type channel (e.g., Sync, LossySync, SyncDrain) has *acquired* priority, the other end has *innate* priority.

However, in the case of non-synchronous channels (e.g., FIFO, AsyncDrain) and priority blocking channels (i.e., BlockSourceSync, BlockSinkSync), their ends can only have *acquired* priority. We augment the constraint-based framework for Reo [24] to capture priority and the priority propagation mechanism.

Motivated by the *constraint-based* nature of Reo, we propose to define the behavior of Reo channels as algebraic constraints that alter a set of variables. In the rest of this paper, we omit data constraints when defining behavior of Reo elements. Data constraints are irrelevant for priority flow and were thoroughly covered in [24].

Let $\mathcal{N}$ and $\mathcal{M}$ be global sets of ends and state memory variables, respectively. Let $n \in \mathcal{N}$ and $m \in \mathcal{M}$. A free variable v ranges over Booleans $\{\top, \bot\}$ and has one of the following forms:

- $\tilde{n}$ shows presence or absence of flow on n;
- $n^{\triangleright}$ indicates the reason for lack of flow on n originating from the prim-

itive or the context (of this primitive);

- $n^{!\bullet}$, $n^{!\circ}$ model priority flow denoting whether n has *acquired* or *innate* priority. An end n has priority iff $n^{!\bullet} \vee n^{!\circ} = \top$;
- $\mathring{m}, \mathring{m}'$ denote whether the state memory variable m is defined in the source and the target states of the transition, respectively.

A constraint ψ encoding the behavior of a Reo network is defined as:

$$\psi ::= \top \mid v \mid \neg\psi \mid \psi \wedge \ \psi,$$

where

$$v ::= \tilde{n} \mid n^{!\bullet} \mid n^{!\circ} \mid n^{\triangleright} \mid \mathring{m} \mid \mathring{m}'.$$

Solutions to ψ are given by assignment of values in $\{\bot, \top\}$ to variable sets defined above. The satisfaction rules for a solution $\langle\delta\rangle$ are given by satisfaction of formulae in propositional logic.

Definition 1 (Reo Constraint Satisfaction Problem (RCSP)). *A Reo Constraint Satisfaction Problem (RCSP) is a tuple*

$$\Psi = \langle \mathcal{N}, \mathcal{M}, M_0, \mathcal{V}, C \rangle,$$

where:

- $\mathcal{N}$ *is a finite set of ends;*
- $\mathcal{M}$ *is a finite set of state memory variables;*
- $M_0 \subseteq \mathcal{M}$ *is a set of state memory variables that define the initial configuration of a network;*
- $\mathcal{V}$ *is a set of variables* $v ::= \tilde{n} \mid n^{\triangleright} \mid \mathring{m} \mid \mathring{m}' \mid n^{!\circ} \mid n^{!\bullet}$ *for* $n \in \mathcal{N}$ *and* $m \in \mathcal{M}$.
- $C = \{c_1,\ c_2,\ ...,\ c_m\}$ *is a finite set of constraints, where each* c_i *is a constraint given by the grammar* ψ *involving a subset of variables* $V_i \subseteq \mathcal{V}$.

We denote the set of all solutions for Ψ as $\mathfrak{S}_\Psi$.

The constraint encoding for a composition of Reo networks is obtained as a composition of their respective encodings:

Definition 2 (Composition $\odot$). *The composition of two RCSPs*

$$\Psi_1 = \langle \mathcal{N}_1, \mathcal{M}_1, M_{0,1},\ \mathcal{V}_1,\ C_1 \rangle$$

and

$$\Psi_2 = \langle \mathcal{N}_2,\ \mathcal{M}_2,\ M_{0,2},\ \mathcal{V}_2,\ C_2 \rangle$$

is defined as:

$$\Psi_1 \odot \Psi_2 = \langle \mathcal{N}_1 \cup \mathcal{N}_2,\ \mathcal{M}_1 \cup \mathcal{M}_2,\ M_{0,1} \cup M_{0,2},\ \mathcal{V}_1 \cup \mathcal{V}_2,\ C_1 \wedge C_2 \rangle.$$

To propagate no-flow reasons via joint networks, we introduce the following axioms:

Axiom 1 (Join). *When a source end c joins a sink end k,*

$$\neg\tilde{c} \Leftrightarrow \neg\tilde{k} \Leftrightarrow (c^{\triangleright} \vee k^{\triangleright}).$$

The *priority join* axiom below guarantees that there is a source of propagation for priority and that priority gets propagated.

Axiom 2 (Priority join). *When a source end c joins a sink end k:*

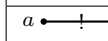
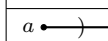
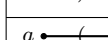
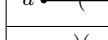
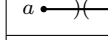
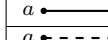
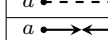
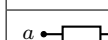
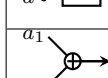
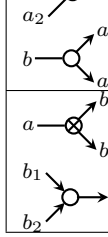
$$(c^{!^{\circ}} \vee c^{!^{\bullet}} \Leftrightarrow k^{!^{\circ}} \vee k^{!^{\bullet}}) \wedge (c^{!^{\circ}} \wedge k^{!^{\circ}} \Leftrightarrow c^{!^{\bullet}} \vee k^{!^{\bullet}}).$$

The primitives that perform non-deterministic choices between alternative flows should respect priority. We need to exclude cases where flow via an end with no priority is chosen over the flow via a prioritized end ready to perform I/O operations. The *non-deterministic choice* axiom carries out this requirement:

Axiom 3 (Non-deterministic choice). *Let $N \subseteq \mathcal{N}$ be a set of ends from which a Reo primitive chooses one for communication. The following guarantees that an end $y \in N$ with no priority has flow only if no prioritized end $x \in N$ is ready to interact:*

$$(\neg\tilde{x} \wedge (x^{!^{\circ}}\ \vee\ x^{!^{\bullet}}) \wedge \tilde{y} \wedge \neg(y^{!^{\circ}}\ \vee\ y^{!^{\bullet}})) \Rightarrow \neg x^{\triangleright}$$

Table 1: Constraint encoding of Reo with priority

Channel	Constraints
	$(\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge a^{!\bullet} \wedge b^{!\bullet}$
	$(\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg b^{!\bullet}$
	$(\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg a^{!\bullet}$
	$(\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge \neg a^{!\bullet} \wedge \neg b^{!\bullet}$
	$(\tilde{a} \Leftrightarrow \tilde{b}) \wedge \neg(a^{\triangleright} \wedge b^{\triangleright}) \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee ((a^{!\circ} \Rightarrow b^{!\bullet}) \wedge (b^{!\circ} \Rightarrow a^{!\bullet})))$
	$(\tilde{b} \Rightarrow \tilde{a}) \wedge \neg a^{\triangleright} \wedge \neg\tilde{a} \Rightarrow b^{\triangleright} \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee ((a^{!\circ} \Rightarrow b^{!\bullet}) \wedge (b^{!\circ} \Rightarrow a^{!\bullet})))$
	$(\tilde{a}_1 \Leftrightarrow \tilde{a}_2) \wedge \neg(a_1^{\triangleright} \wedge a_2^{\triangleright}) \wedge ((\neg a^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b^{!\bullet} \wedge \neg b^{!\circ}) \vee (a^{!\circ} \Rightarrow b^{!\bullet}) \wedge (b^{!\circ} \Rightarrow a^{!\bullet}))$
	$(\tilde{a} \Rightarrow \neg\mathring{m} \wedge \mathring{m}') \wedge (\tilde{b} \Rightarrow \mathring{m} \wedge \neg\mathring{m}') \wedge (\neg\tilde{a} \wedge \neg\tilde{b}) \Rightarrow (\mathring{m} \Leftrightarrow \mathring{m}') \wedge (\neg\mathring{m} \Rightarrow b^{\triangleright}) \wedge (\mathring{m} \Rightarrow a^{\triangleright}) \wedge (\neg a^{!\bullet} \wedge \neg b^{!\bullet})$
	$\tilde{b} \Leftrightarrow (\tilde{a_1} \wedge \tilde{a_2}) \wedge \neg\tilde{b} \Rightarrow ((\neg b^{\triangleright} \wedge a_1^{\triangleright} \wedge a_2^{\triangleright}) \vee (\neg a_1^{\triangleright} \wedge a_2^{\triangleright} \wedge b^{\triangleright}) \vee (\neg a_2^{\triangleright} \wedge a_1^{\triangleright} \wedge b^{\triangleright})) \wedge ((\neg b^{!\bullet} \wedge \neg a_1^{!\bullet} \wedge \neg a_2^{!\bullet} \wedge \neg b^{!\circ} \wedge \neg a_1^{!\circ} \wedge \neg a_2^{!\circ}) \vee ((b^{!\circ} \Rightarrow (a_1^{!\bullet} \wedge a_2^{!\bullet}) \wedge (a_1^{!\circ} \vee a_2^{!\circ}) \Rightarrow b^{!\bullet})))$
	$\tilde{a} \Leftrightarrow (\tilde{b_1} \vee \tilde{b_2}) \wedge \neg(\tilde{b_1} \wedge \tilde{b_2}) \wedge \tilde{a} \Leftrightarrow (\neg a^{\triangleright} \vee \neg(b_1^{\triangleright} \vee b_2^{\triangleright})) \wedge [(\neg a^{!\bullet} \wedge \neg b_1^{!\bullet} \wedge \neg b_2^{!\bullet} \wedge \neg a^{!\circ} \wedge \neg b_1^{!\circ} \wedge \neg b_2^{!\circ}) \vee (\tilde{a} \Rightarrow ((a^{!\circ} \Rightarrow (b_1^{!\bullet} \vee b_2^{!\bullet})) \wedge (b_1^{!\circ} \vee b_2^{!\circ}) \Rightarrow a^{!\bullet}) \wedge (\tilde{b_1} \Rightarrow (a^{!\circ} \Rightarrow b_1^{!\bullet} \wedge b_1^{!\circ} \Rightarrow a^{!\bullet}) \wedge ((\neg b_1^{!\circ} \wedge \neg b_1^{!\bullet} \wedge \neg\tilde{b_2} \wedge (b_2^{!\circ} \vee b_2^{!\bullet})) \Rightarrow \neg b_2^{\triangleright})) \wedge (\tilde{b_2} \Rightarrow (a^{!\circ} \Rightarrow b_2^{!\bullet} \wedge b_2^{!\circ} \Rightarrow a^{!\bullet} \wedge (\neg b_2^{!\circ} \wedge \neg b_2^{!\bullet} \wedge \neg\tilde{b_1} \wedge (b_1^{!\circ} \vee b_1^{!\bullet}) \Rightarrow \neg b_1^{\triangleright})))))]$

In our previous work [24], we described the behavioral constraints that each Reo primitive imposes on a network as a CSP. We now extend these constraints with priority capturing variables.

If variable $p^{!\bullet} = \top$, the end p has *innate* priority. A PrioritySync channel with source end a and sink end b, where both ends have *innate* priority, yields $a^{!\bullet} \wedge b^{!\bullet}$. A primitive end can also obtain *innate* priority via propagation. For instance, if one end of a Sync channel has *acquired* priority, which means it is prioritized because a primitive connected to it propagates priority, then the other end will have *innate* priority. We denote *acquired* priority for a primitive end p as: $p^{!\circ} \wedge \neg p^{!\bullet}$. The priority capturing constraint for a Sync channel with source end a and sink end b can be specified as follows:

$$\neg(a^{!\circ} \vee a^{!\bullet} \vee b^{!\circ} \vee b^{!\bullet}) \vee (a^{!\circ} \wedge \neg a^{!\bullet} \wedge b^{!\bullet}) \vee (a^{!\bullet} \wedge b^{!\circ} \wedge \neg b^{!\bullet}).$$

The assertion $\neg p^{!\bullet}$ blocks the priority propagation on p. Though, p can still have *acquired* priority through a potential connecting primitive when $p^{!\circ} = \top$.

Table 1 shows the constraint encoding of Reo channels and nodes in presence of priority flow. The solutions to the CSP expressing the behavior of a Reo network encode possible flow through its nodes. Since a network may

later connect to another network, the constraints should account for priority imposed by potential future connections. This information can be discarded when analyzing the behavior of a network in isolation. To exclude such cases, we should restrict the possible values of boundary ends. This is achieved with the help of the *grounding* axiom:

Axiom 4 (Grounding). *Let $B \subseteq \mathcal{N}$ be a set of boundary nodes in a Reo network. We rule out the solutions that are only present for further expansion of the network by:*

$$\forall b \in B : b^{!^\circ} \Rightarrow b^{!^\bullet}.$$

Solutions of the RCSP represent semantics of the corresponding Reo network, but they are specified as equations, which are much harder to interpret than an equivalent automata-based semantics. To tackle this issue, we introduce a new form of automata-like semantics for Reo, which we call Reo Labeled Transition System (RLTS). The purpose of the RLTS is to compactly represent solutions of RCSPs for visualization, model checking and simulation.

Definition 3 (RLTS). *A Reo Labeled Transition System (RLTS) is a tuple*

$$\mathcal{L} = (\mathcal{N},\ \mathcal{M},\ Q,\ \rightarrow,\ q_0),$$

where:

- $\mathcal{N}$ *is a set of ends,*
- $\mathcal{M}$ *is a set of state memory variables,*
- Q *is a (finite) set of states of the form* $\langle M \rangle$*, where* M *is the set of state memory variables that are valid in the given state,*
- $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times Q$ *is a transition relation, wherein* N*,* R*, and* I *in* $(q,\ N,\ R,\ I,\ p) \in \rightarrow$ *represent the ends that have flow, those without flow for which the reason for no flow is the end not being ready for interaction, and the ends with priority.*
- $q_0 \in Q$ *is the initial state.*

We write $q \xrightarrow{N,R,I} p$ instead of $(q, N, R, I, p) \in \rightarrow$.

Note that $n \notin N$ does not always mean $n \in R$ as the reason for data flow can be the network (then, n requires a reason for no flow). For $n \in I, n \notin R \Leftrightarrow n \in N$.

Given a Reo network, its RCSP can be obtained by traversing the network and forming the conjunction of the constraint encodings of its primitives.

The procedure to solve an RCSP is presented in Algorithm 1. It takes a Reo connector and its RCSP and outputs the solutions set and the initial state of the connector. The algorithm initializes the global variables that keep the states of FIFO channels (*fifoStates*), the states to explore (*toExplore*), and the visited states (*visited*) (lines 2, 3). While *toExplore* is not empty, Ψ is updated with the current state and its Conjunctive Normal Form (CNF)

```
1  Input: A Reo network R and its RCSP Ψ
2  Output: Solutions for Ψ
3  fifoStates ← initial states of FIFOs from Ψ;
4  state_0 ← {⟨ fifoStates ⟩}; toExplore ← {state_0};
5  visited ← {}; solutions ← {};
6  while (toExplore ≠ {}) do
7      state ← toExplore.pop(); visited ← visited ∪ {state};
8      cnf ← updateStateAndMakeCNF(Ψ, state);
9      solutions_B ← solve(cnf);
10     for sol_B ∈ solutions_B do
11         state'← next state of FIFOs extracted from sol_B;
12         if state' ∉ visited and state' ∉ toExplore then
13             toExplore ← toExplore ∪ {state'};
14         end
15         solutions ← solutions ∪ {⟨state, sol_B, state'⟩};
16     end
17     output ← {solutions, state_0};
18 end
```

Algorithm 1: Finding solutions for a given RCSP

is produced for computing the solutions of the Boolean predicates (lines 4, 5) The $\langle state' \rangle$ indicates the new state of the connector and if it is not already explored or queued to be processed, it gets added to the list of states to be explored (lines 6-9). Then, the solutions set is updated with the current solution (line 13). The final output is the set of solutions and the initial state.

We use $s(v)$ to denote the value assigned to variable v in the solution s, and define the following operations on a solution for RCSP

$$\Psi = \langle \mathcal{N},\ \mathcal{M},\ M_0,\ \mathcal{V},\ C \rangle :$$

- $source(s) = \langle \{m | m^{\circ} \in \mathcal{M} : s(m^{\circ}) = \top\} \rangle$,
- $target(s) = \langle \{m | m'^{\circ} \in \mathcal{M} : s(m'^{\circ}) = \top\} \rangle$,
- $flow(s) = \{n | n \in \mathcal{N} : s(\tilde{n}) = \top\}$,
- $reasonGiving(s) = \{n | n \in \mathcal{N} : s(n^{\triangleright}) = \top\}$,
- $priority(s) = \{n | n \in \mathcal{N} : (s(n^{!\circ}) \vee s(n^{!\bullet})) = \top\}$.

We say $s \backsim q \xrightarrow{N,R,I} p$, where $q = source(s)$, $N = flow(s)$, $R = reasonGiving(s)$, $I = priority(s)$, and $p = target(s)$.

The solutions for RCSP can be shown in the form of an LTS produced with the help of the following function:

Definition 4 (Visualization). *The visualization function γ on the set of solutions $\mathfrak{S}_\Psi$ for RCSP*

$$\Psi = \langle \mathcal{N}_\Psi,\ \mathcal{M}_\Psi,\ M_{\Psi_0},\ \mathcal{V},\ C \rangle$$

yields RLTS

$$\mathcal{L} = (\mathcal{N}_\mathcal{L},\ \mathcal{M}_\mathcal{L},\ Q,\ \rightarrow,\ q_0),$$

where

- $\mathcal{N}_\mathcal{L} = \{n | s(\tilde{n}) = \top,\ n \in \mathcal{N}_\Psi, s \in \mathfrak{S}_\Psi\}$,
- $\mathcal{M}_\mathcal{L} = \{m | s(m^\circ) = \top \vee s(m'^\circ) = \top,\ m \in \mathcal{M}_\Psi, s \in \mathfrak{S}_\Psi\}$,
- $Q = \bigcup_{s \in \mathfrak{S}_\Psi} \{source(s), target(s)\}$,
- $s \backsim q \xrightarrow{N,R,I} p \mid s \in \mathfrak{S}_\Psi$,
- $q_0 = source(s_0)$.

Now we define composition operator for Reo with priority and show that the semantics of the composed circuit can be derived from the semantics of its constituent parts.

Definition 5 (Composition $\boxdot$). *We define the composition of*

$$\mathcal{L}_1 = (\mathcal{N}_1,\ \mathcal{M}_1, Q_1, \rightarrow_1,\ q_{0_1})$$

and

$$\mathcal{L}_2 = (\mathcal{N}_2,\ \mathcal{M}_2,\ Q_2,\ \rightarrow_2,\ q_{0_2})$$

as

$$\mathcal{L}_1 \boxdot \mathcal{L}_2 = (\mathcal{N}_1 \cup \mathcal{N}_2,\ \mathcal{M}_1 \cup \mathcal{M}_2,\ \rightarrow,\ q_{0_1} \times q_{0_2})$$

where $\rightarrow$ is defined as:

$$\frac{q_1 \xrightarrow{N_1,R_1,I_1}{}_1 t_1,\ q_2 \xrightarrow{N_2,R_2,I_2}{}_2 t_2,\ N_1 \cap \mathcal{N}_2 = N_2 \cap \mathcal{N}_1,\ R_1 \cap \mathcal{N}_2 = R_2 \cap \mathcal{N}_1,\ I_1 \cap \mathcal{N}_2 = I_2 \cap \mathcal{N}_1}{q_1 \times q_2 \xrightarrow{N_1 \cup N_2, R_1 \cup R_2, I_1 \cup I_2} t_1 \times t_2},$$

$$\frac{q_1 \xrightarrow{N_1,R_1,I_1}_1 t_1,\ q_2 \xrightarrow{N_2,R_2,I_2}_2 t_2,\ N_1 \cap \mathcal{N}_2 = \emptyset}{q_1 \times q_2 \xrightarrow{N_1,R_1,I_1} t_1 \times t_2},$$ *and its symmetric rule.*

Theorem 1 (Compositionality). *Let* Ψ_1 *and* Ψ_2 *be two RCSPs, we show that* $\gamma(\Psi_1 \odot \Psi_2) = \gamma(\Psi_1) \boxdot \gamma(\Psi_2)$.

Proof 1. *Let* $\gamma(\Psi_1) = (\mathcal{N}_1, \mathcal{M}_1, Q_1, \rightarrow_1, q_{0_1})$, $\gamma(\Psi_2) = (\mathcal{N}_2, \mathcal{M}_2, Q_2, \rightarrow_2, q_{0_2})$, *and* $\gamma(\Psi_1 \odot \Psi_2) = (\mathcal{N}, \mathcal{M}, Q, \rightarrow, q_0)$. *It is trivial to see that* $\mathcal{N} = \mathcal{N}_1 \cup \mathcal{N}_2$, $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$, $Q = Q_1 \times Q_2$, $q_0 = q_{0_1} \times q_{0_2}$.

Assume $\exists s \in \mathfrak{S}(\Psi_1 \odot \Psi_2)$, $s_1, \in \mathfrak{S}_1$, $s_2 \in \mathfrak{S}_2$, $t_1 : q_1 \xrightarrow{N_1,R_1,I_1}_1 p_1$, $t_2 : q_2 \xrightarrow{N_2,R_2,I_2}_2 p_2$ *s.t.* $s_1 \backsim t_1$ *and* $s_2 \backsim t_2$, *but* $\nexists\ t : q \xrightarrow{N,R,I} p \in \rightarrow$ *s.t.* $s \backsim t$. *Therefore,* $N_1 \cap \mathcal{N}_2 \neq N_2 \cap \mathcal{N}_1 \wedge N_1 \cap \mathcal{N}_2 \neq \emptyset$ *or* $(N_1 \cup N_2) \cap (R_1 \cup R_2) \neq \emptyset$. *The latter is impossible. For the former, either* $n \in N_1, n \notin N_2$ *or* $n \in N_2, n \notin N_1$, *which is impossible as it induces* $s(n) = \top \wedge s(n) = \bot$.

Similarly, we can show it is impossible to have $t \in \gamma(\Psi_1 \odot \Psi_2)$, *when there is no* $s \in \mathfrak{S}$ *s.t.* $s \backsim t$.

Table 2: Encoding of numeric priority constraints for Reo

Channel	Constraints	Channel	Constraints
a —!→ b	$a^{!\bullet} \geq P \wedge b^{!\bullet} \geq P$	a —)→ b	$b^{!\bullet} = 0$
a —)(→ b	$a^{!\bullet} = 0 \wedge b^{!\bullet} = 0$	a —(→ b	$a^{!\bullet} = 0$

Channel	Constraints
a ⇢ b	$(a^{!\bullet} = 0 \wedge a^{!\circ} = 0 \wedge b^{!\bullet} = 0 \wedge b^{!\circ} = 0) \vee ((a^{!\circ} > 0 \Rightarrow (a^{!\circ} = b^{!\bullet})) \wedge (b^{!\circ} > 0 \Rightarrow (b^{!\circ} = a^{!\bullet}))) \wedge (b^{!\bullet} > 0 \Rightarrow \tilde{b})$
a → b a →← b	$(a^{!\bullet} = 0 \wedge a^{!\circ} = 0 \wedge b^{!\bullet} = 0 \wedge b^{!\circ} = 0) \vee ((a^{!\circ} > 0 \Rightarrow (a^{!\circ} = b^{!\bullet})) \wedge (b^{!\circ} > 0 \Rightarrow (b^{!\circ} = a^{!\bullet})))$
a —▭→ b	$a^{!\bullet} = 0 \wedge b^{!\bullet} = 0$
a_1, a_2 ⊕→ b b → a_1, a_2	$((a^{!\bullet} = 0 \wedge a^{!\circ} = 0 \wedge b^{!\bullet} = 0 \wedge b^{!\circ} = 0) \vee ((b^{!\circ} > 0 \Rightarrow (a_1^{!\bullet} = b^{!\circ} \wedge a_2^{!\bullet} = b^{!\circ})) \wedge (a_1^{!\circ} > 0 \Rightarrow (a_2^{!\bullet} = a_1^{!\circ} \wedge b^{!\bullet} = a_1^{!\circ}) \wedge (a_2^{!\circ} > 0 \Rightarrow (a_1^{!\bullet} = a_2^{!\circ} \wedge b^{!\bullet} = a_2^{!\circ})))))$
a → b_1, b_2 b_1, b_2 → a	$((a^{!\bullet} = 0 \wedge a^{!\circ} = 0 \wedge b^{!\bullet} = 0 \wedge b^{!\circ} = 0) \vee (\tilde{b_1} \Rightarrow (b_1^{!\circ} > 0 \Rightarrow (b_1^{!\circ} = a^{!\bullet}))) \wedge (\tilde{b_2} \Rightarrow (b_2^{!\circ} > 0 \Rightarrow (b_2^{!\circ} = a^{!\bullet}))) \wedge ((max(b_1^{!\circ}, b_1^{!\bullet}) > max(b_2^{!\circ}, b_2^{!\bullet})) \Rightarrow ((\tilde{b_2} \wedge \neg \tilde{b_1}) \Rightarrow \neg b_1^{\triangleright})) \wedge ((max(b_2^{!\circ}, b_2^{!\bullet}) > max(b_1^{!\circ}, b_1^{!\bullet})) \Rightarrow ((\tilde{b_1} \wedge \neg \tilde{b_2}) \Rightarrow \neg b_2^{\triangleright})))$

4. Numeric priority flow

In BPM, often more than one priority level is required. Here we extend the behavioral semantics of Reo with priority channels to support numeric priorities.

To distinguish priority levels in the flow, we allow two predefined variables $n^{!\circ}$ and $n^{!\bullet}$ representing acquired and innate priority at the port n, to range over the set $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ (of natural numbers with 0), with an assumption that larger numbers represent higher priority and 0 means no-priority. Each PrioritySync channel comes with a user defined priority value, which propagates through its ends. For propagation of higher priority over lower priority, we constrain priority variables to be greater than or equal to their initial values.

Table 2 defines extra constraints for Reo primitives to handle multi-level priority flow. The new constraint-based encoding of the Replicator and Router nodes in this table are constructed in accordance with Axiom 3.

Similarly to RLTS, we define a Numeric Priority (NP) RLTS to define the behavior of Reo models in presence of priority flow levels.

Definition 6 (NP-RLTS). *A Numeric Priority Reo Labeled Transition Sys-*

tem (NP-RLTS) is a tuple

$$\mathcal{L} = (\mathcal{N},\ \mathcal{M},\ Q,\ \rightarrow,\ q_0),$$

where:

- $\mathcal{N}$ *is a set of ends,*
- $\mathcal{M}$ *is a set of state memory variables,*
- Q *is a (finite) set of states of the form* $\langle M \rangle$*, where* M *is the set of state memory variables that are valid in the given state,*
- $s \backsim q \xrightarrow{N,R,I} p \mid s \in \mathfrak{S}_\Psi$*,*
- $\rightarrow \subseteq Q \times 2^{\mathcal{N}} \times 2^{\mathcal{N}} \times \mathcal{N} \mapsto \mathbb{N} \times Q$ *is a transition relation, wherein* N*,* R*, and* f_I *in* $(q,\ N,\ R,\ f_I,\ p) \in \rightarrow$ *are the ends having flow, those without flow for which the reason for no flow is the end not being ready for interaction, and a partial map of nodes with priority to their priority values, respectively.*
- $q_0 \in Q$ *is the initial state.*

Similarly to RLTS, we write $q \xrightarrow{N,R,f_I} p$ *instead of* $(q,\ N,\ R,\ f_I,\ p) \in \rightarrow$*. For all* $q \xrightarrow{N,R,f_I} p$*:* $f(n) > 0, n \notin N \Leftrightarrow n \in R$*. We redefine priority*$(s)$ *as* $\{(n,p) | n \in \mathcal{N}_\Psi : s(n^{!^\circ}) = p \vee s(n^{!^\bullet}) = p\}$*.*

Definition 7 (Extended Visualization). *The visualization function* γ *on the set of solutions* $\mathfrak{S}_\Psi$ *for NP-RCSP*

$$\Psi = \langle \mathcal{N}_\Psi,\ \mathcal{M}_\Psi,\ M_{\Psi_0},\ \mathcal{V},\ C \rangle$$

yields NP-RLTS

$$\mathcal{L} = (\mathcal{N}_L,\ \mathcal{M}_L,\ Q,\ \rightarrow,\ q_0),$$

where

- $\mathcal{N}_\mathcal{L} = \{n | s(\tilde{n}) = \top,\ n \in \mathcal{N}_\Psi, s \in \mathfrak{S}_\Psi\}$*,*
- $\mathcal{M}_\mathcal{L} = \{m | s(m^\circ) = \top\ \vee\ s(m'^\circ) = \top,\ m \in \mathcal{M}_\Psi, s \in \mathfrak{S}_\Psi\}$*,*
- $Q = \bigcup_{s \in \mathfrak{S}_\Psi} \{source(s),\ target(s)\}$*,*
- $s \backsim q \xrightarrow{N,R,f_I} p \mid s \in \mathfrak{S}_\Psi$*,*
- $q_0 = source(s_0)$*.*

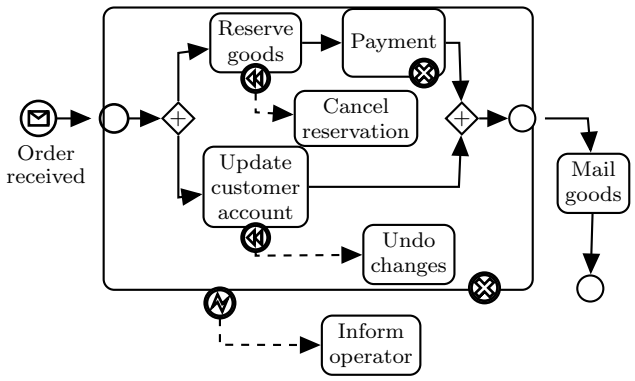


Figure 2: Sales process, a BPMN model with priority flow

5. Case study

Here we consider a practical example of priority flow modelling using Reo and give an indication of the performance of our approach.

Figure 2 depicts a sales process. The process starts when a purchase order is received and proceeds to the reservation of the items ordered by the customer and processing the payment, simultaneously updating her account. If the customer cancels the order or the payment encounters a problem, a *cancellation* event is triggered to enable the compensation flow which reverses the effects of the activities performed so far. If an *error* event occurs, all tasks inside the transaction stop, and the boundary error catch event redirects the flow to "inform operator" task. Finally, if no problem occurs, the ordered items are shipped and the purchase process case successfully ends.

To analyze the presented BPMN process, we convert it to a Reo network. The core mapping is presented in [29, 16]. In a nutshell, we map *tasks* to FIFO_1 channels, *message* and *events* (such as *cancel* and *error* events) to writer components simulating the incoming flow from the environment. A diverging *parallel* gateway is mapped to a Replicator nodes, while a converging *parallel* gateway is mapped to a CrossProduct node. The *sequence* flow is modelled by synchronous channels. The mapping of *exception* and *error* handling flows are more complex and are presented in [18]. In this example, the *error* handling flow has the highest priority, while the *exception* handling has the medium priority, and the *success* flow has no priority. Router nodes are employed to alternate among these three types of flow.

Figure 3 shows a Reo network that simulates this process. The process

starts when a token is produced by the writer W_2, implying that a purchase order has been received. In BPMN, when a *start* event is triggered, a new instance of the process is instantiated. A Reo network in Figure 3 instead would simply read a new token. It is possible to introduce a circuit that prevents new entries before the existing token has left the network (meaning that the process case has been completed), however, we omit it here to avoid over-complicating the case study and simply assume that the circuit needs to handle one request at the time.

To reference a node end, we append a number index to the node name (e.g., A_1). For brevity, we may also group the ends belonging to the same node, e.g., $A_{1,2}$ (referring to ends A_1 and A_2). We refer to a channel using the name of the nodes connected to its ends (e.g., BC). Similarly, we append a number index to a channel name to denote a channel end (e.g., BC_1).

The end A_1 (marked as 1 next to the port A in Figure 3) reads a token from the writer W_2 and delivers it to the replicator node B, which duplicates the token and forwards two copies to the FIFO_1 channels BC and BE. The token from BC continues to the FIFO_1 channel CD, modelling the payment activity. If the payment succeeds, the flow from CD and BE channels merge and a token enters the FIFO_1 channel FG and gets consumed by the reader R_3.

If the payment fails, performed activities need to be compensated. A token from W_1 simulates a payment failure and triggers the cancellation scenario. As part of this scenario, the PrioritySync channel IJ imposes a priority of *one* on the failure associated flow. The node J replicates the failure token into the LossySync channels JM and JU. Depending on which of FIFO_1 channels BC or CD is full, the connected LossySync channels lose the incoming tokens or pass them to the adjacent SyncDrains to consume the tokens from the full FIFO_1.

After the sales process has been initiated, the replicator node J sends tokens into the channels JK and JN, which simulate *cancel reservation* and *undo changes* tasks, respectively. The error flow, triggered by the writer W_3, is structurally similar to the failure flow, but it has higher priority of 2 imposed by the PrioritySync channel SQ.

To obtain the NP-RLTS for the presented circuit, we form the RCSP of the network by recording constraints for each and everyone of its primitives, solve the RCSP, and extract transitions from the set of solutions, as described in Algorithm 1.

A priority-respecting routing between normal and compensation flows

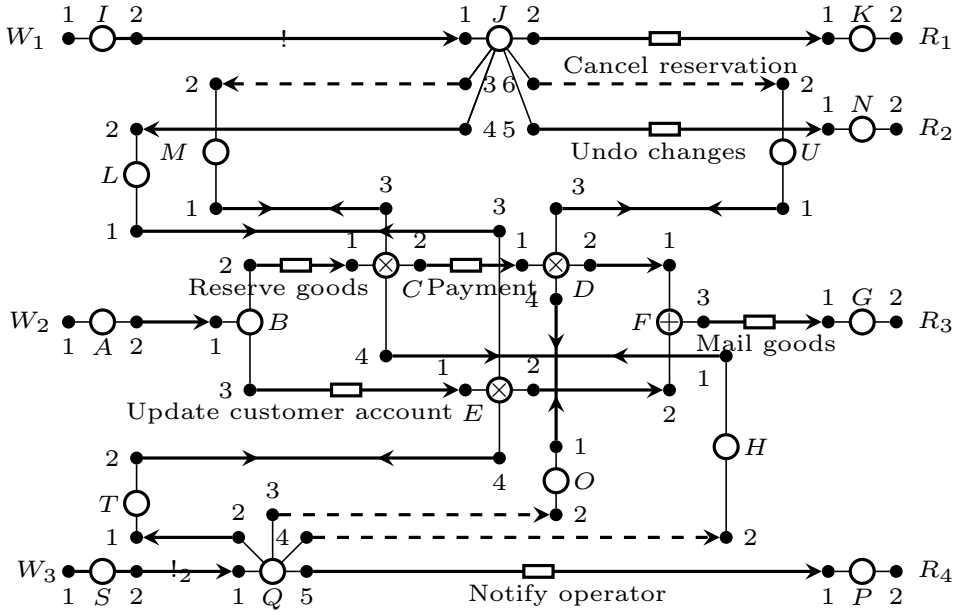


Figure 3: Sales process, Reo model with priority flow

corresponds to the following condition:

$$(\{BE\} \in source(t) \wedge (C_1 \in flow(t) \vee E_1 \in flow(t))) \Rightarrow$$
$$((W_3 \notin reasonGiving(t) \Leftrightarrow W_3 \in flow(t)) \wedge (W_3 \in reasonGiving(t) \wedge$$
$$W_1 \notin reasonGiving(t)) \Leftrightarrow W_1 \in flow(t)).$$

Informally, it states that at any state wherein BE holds, W_3 has flow unless it provides a reason for no-flow, and if W_3 provides a reason for no flow, W_1 has flow, unless it provides a reason for no-flow.

To illustrate the effect of priority flow modelling, we first generate RLTS of the network in absence of priority, i.e., when the router node E chooses one of its outgoing flows non-deterministically and the normal flow can continue executing regardless of a payment failure. Such priority-agnostic RLTS is shown in Figure 4. Since the property solely mentions the ends C_1, E_1, W_1, and W_3 on the transitions originating from the states where the FIFO_1 channel BE is full, we omit transitions that do not include either of these ends and skip end labels on remaining transitions (Reo hiding operator [15, 31] allows us to abstract from the flow presence on any chosen node or node end (port) without compromising the correctness of the model). The aforemen-

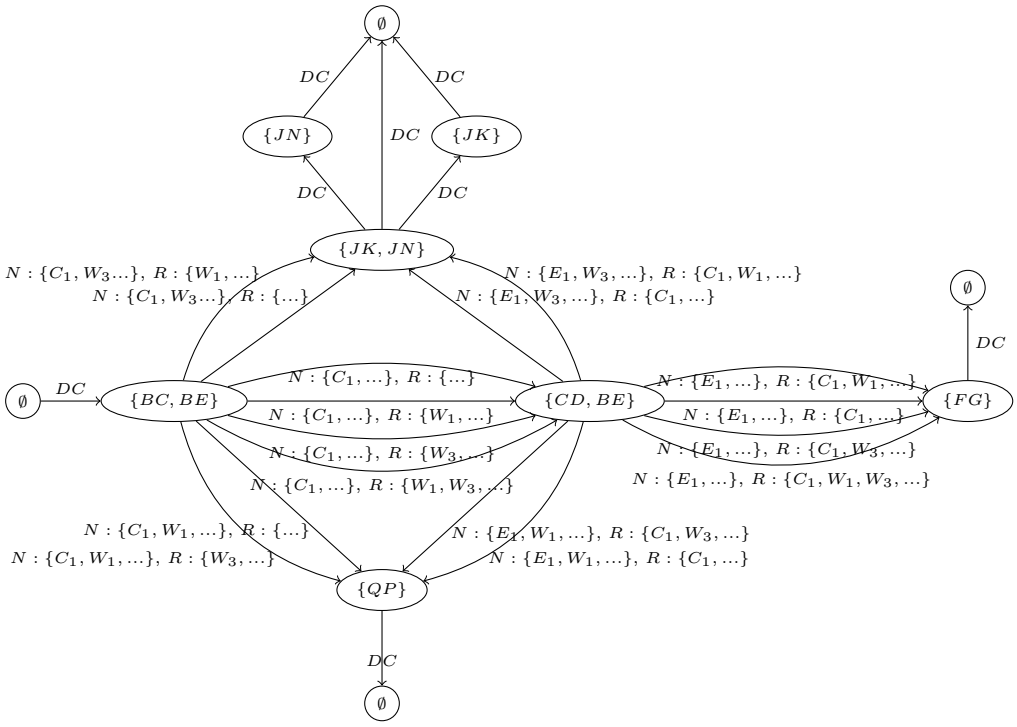


Figure 4: Sales process, RLTS

tioned priority condition does not hold on this RLTS as it contains transitions originating from the states $\{BC, BE\}$ and $\{CD, BE\}$, wherein either $W_3 \notin R \wedge W_3 \notin N$ or $W_3 \in R \wedge W_3 \notin N \wedge W_1 \notin R$ (recall that R is the set of ends providing a reason for no-flow and N is the set of ends with data flow).

Here we show how considering priority constraints rules out these transitions. We reason about one of the transitions (the transition from $\{BC, BE\}$ to $\{CD, BE\}$ with $N = \{C_1, ...\}$, $R = \{W_1, ...\}$) as follows:

$$0 : \frac{\exists t \in \rightarrow : C_1 \in N(t), W_1 \in R(t), W_3 \notin N(t), W_3 \notin R(t)}{\exists\ s \in \mathfrak{S}(\Psi) \mid s \Rightarrow \tilde{C_1} \wedge \neg \tilde{W_3} \wedge {W_1}^{\triangleright} \wedge \neg {W_3}^{\triangleright}}$$

$$1 : \frac{0\&\Psi_{PrioritySync_1}(SQ_{1,2}) \text{ \& priority join}}{C_3^{!\circ} = 1},$$

$$2 : \frac{0\&\Psi_{PrioritySync_2}(IJ_{1,2}) \text{ \& priority join}}{C_4^{!\circ} = 2},$$

$$3 : \frac{2; \Psi_{router}(C_{1,2,3,4})}{\tilde{C_1} \wedge \neg \tilde{C_4} \Rightarrow {C_4}^{\triangleright}}, \quad 4 : \frac{0 \text{ \& join}}{\neg \tilde{C_4}}, \quad 5 : \frac{\text{coloring \& join}}{{C_4}^{\triangleright} \Rightarrow {W_3}^{\triangleright}},$$

$$6 : \frac{0 \text{ \& } 3 \text{ \& } 4 \text{ \&} 5}{{W_3}^{\triangleright}}, \quad 7 : \frac{0 \text{ \& } 5}{\perp}.$$

This disproves the existence of the aforementioned transition. Consequently, when $\{BC\}$ and $\{BE\}$ are full, the request from W_1 is never ignored. In the above reasoning process, *join* and *priority join* refer to the conditions defined by Axiom 1 and Axiom 2, respectively.

5.1. Performance evaluation

The execution time of Algorithm 1 depends on the number of states in the RCSP and the time to solve the RCSP. We evaluate the performance of our framework and compare it with the performance of existing approaches on computing Reo semantics using an *n-Sequencer* circuit. The circuit consists of n sequentially connected FIFO_1 channels; at each step it propagates a token from a full FIFO_1 channel forward and simultaneously outputs it via of the output nodes $(A, B, C, ...)$. Figure 5 shows the version of the circuit for $n = 7$, i.e., a *7-Sequencer*.

Although the final number of states in the *(n+1)-Sequencer* is equal the number of states in the *n-Sequencer* plus 1, adding a new FIFO_1 channel doubles the number of states in the decomposed system and increases the

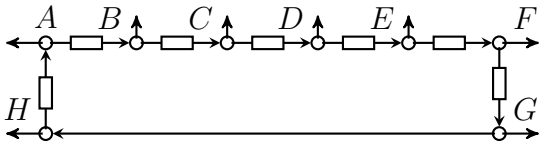
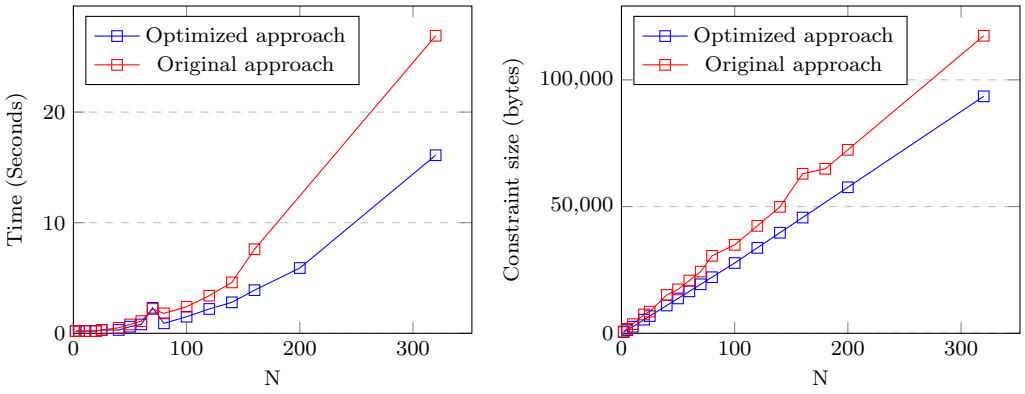


Figure 5: 7-Sequencer



(a) Time (s), build single solution

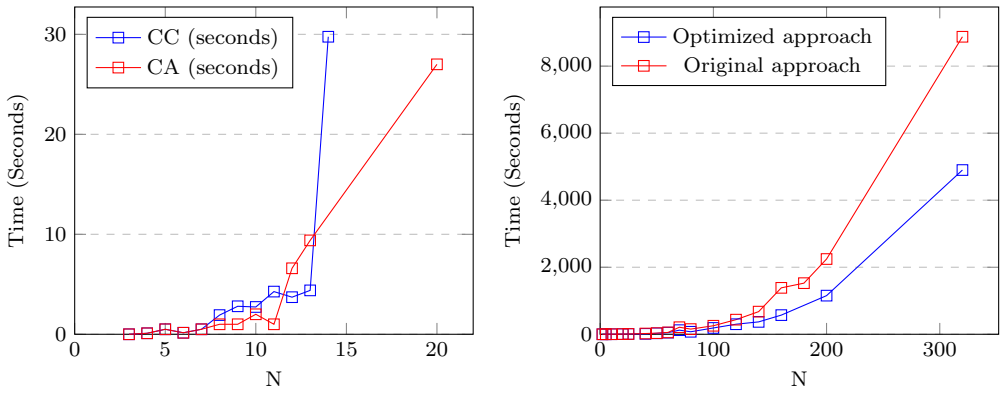
(b) Size of the RCSP

Figure 6: Effect of RCSP optimization for n-Sequencer

complexity of the RCSP. This makes the network a convenient choice for performance bench-marking. Since we compare our RCSP-based approach for computing the semantics of Reo given the semantics of its constituent parts with the analogous functionality of the existing frameworks, we do not include priority in this case study (incorporating priority does not affect the number of states in the model and influences only the size of the constraints).

Our implementation [32] relies on an algebra system *REDUCE* [33] which we use to compute and solve the CNF given a set of RCSP constraints. We provide results for both straightforward encoding of RCSP and an optimized encoding where the number of variables in RCSP has been reduced by substituting equivalent variables with a single variable.

Figure 6(a) presents the average time to compute a single solution for the RCSP for various sizes of the *n-Sequencer*. Figure 6(b) shows the relation between n and the size of the corresponding CNF. Figure 7(a) shows the time consumed to calculate the Connector Coloring (CC) semantics [34] and the CA semantics [22] for the *n-Sequencer* circuit using the Extensible



(a) Time (s), building CA and CC table (b) Time (s), calculating the NP-RLTS

Figure 7: Performance evaluation of building semantic models for n-Sequencer

Coordination Tools (ECT) [35]. Figure 7(b) shows the total time required to compute all solutions of RCSP for the circuit. The benchmarks have been performed on Mac Book Pro with 2.8 GHz Intel Core i7 and 16 GB MHz DDR3 memory.

The tools computing CC and CA-based semantics fail with the stack overflow error for $n = 16$ and $n = 21$, respectively. The construction of LTS semantics via the mCRL2 mapping [36] can handle much larger circuits with depth-first traversal optimization, provided the size of the final state space does not grow exponentially with n (the graph representing Reo circuit is traversed and joint end labels are replaced with one label at each step). The approach presented in this paper is a generic solution that does not take into account the circuit topology and shows the sign of slowdown for circuit instances with $n > 300$. Hence, our tool for deriving Reo semantics with the help of constraint-solvers can handle larger models than the existing tools can. The effect of the optimization is more significant for larger values of n.

6. Related work

Similar to the BPMN, Reo belongs to the class of graph-based workflow languages [37], which define control flow through explicit control links between activities. In block-structured languages, such as BPEL [38], control flow is defined by using typical programming language block-structures such as `if` and `while`. Workflow patterns [39] focus on the expressiveness of the control flow constructs and do not explicitly distinguish between graph-based

and block-structured modelling. Reo has been used for formalizing process models specified both in graph-base [29] and in block-structured formats [40]. An overview of formalisms used in the process and workflow modelling area is extensively presented in [5, 41, 42]. In this section, we overview approaches that aim at definition of behavioral semantics of process models with priority flow.

A prominent class of formal graph-based languages with priority support is the Petri-net based workflow nets [43, 44]. Priority flow in Petri nets is managed with the help of inhibitor arcs [12] and transition priorities [45]. Inhibitor arcs allow a transition to fire only if the adjacent place is empty. In Petri nets with transition priorities, a priority function maps transitions into non-negative natural numbers representing their priority level. Given a set of enabled transitions, the transitions with higher priority fire before the transitions with lower priority.

Best and Koutny [14] introduced Petri-net based semantics to priority systems defined by a bounded Petri-net and priority relation pair. The semantic model is expressed in the form of a Petri-net with additional places and arcs, and by splitting the transitions of the original net. Bause [13] proposed Petri net analysis by introducing dynamic priorities to decrease the number of reachable markings to tackle space-explosion problem. Concurrent Petri nets [46] were proposed to model *preemptible* systems. Preemption relates to controlling the execution of the processes composing a concurrent system by temporarily interrupting a task with the intention of resuming it at a later time. In [47], priorities on transitions are employed to transform live and unbounded Petri-nets into live and bounded nets, which is useful for proving process soundness.

Petri nets form an inherently asynchronous model. In most cases, capabilities of the model are sufficient for simulating mainstream business processes and service-based interaction. Reo instead has advantages when dealing with composition of complex synchronous patterns, which can easily be used to model atomic transactions. While not every application involves such patterns, we see vast opportunities in using Reo for modelling processes with massive synchronous interactions (such as e.g., biological processes), provided the supporting tools can handle large networks. In this context, our work on translating Reo semantics into problems tractable by existing constraint solvers is a prominent step towards wider adaptation of the framework.

Our priority model preserves main characteristics of Reo such as compositionality and propagation of synchrony. The presented formalism is compati-

ble with other extensions of Reo which rely on constraint solving to build the semantic model of the circuit [48]. Moreover, the use of Reo priority channels can be bound and applied to a relevant scope. Hence, the computational overhead will be minimal if the priority flow applies only to localized process fragments such as cancellable or compensatable activities/subprocesses in BPMN.

Cleaveland and Hennessy [49, 50] developed an operational semantics for processes having actions that take priority over other actions. They defined equivalence notions for the process algebra with prioritized actions and indicated that it should contain associated combinators such as prioritization and de-prioritization and that certain other characteristics such as *strongly prioritized* actions are desirable.

Reo differs from process algebras as its main focus is on interaction rather than action synchronization. As was shown in [36], its semantics can be defined using process algebras, but the mapping to notations that do not preserve compositionality neglects the key modeling advantage of Reo. Our extension allows modellers to capture semantics of processes with prioritized actions and reason about their composition.

Stursberg and Hillmann [51] study distributed discrete-event systems (DES) with priority structures, represented as automata. Subsystems with high priorities are supplied with the output of subsystems with lower priority, which helps to improve the performance of control synthesis for the DESs. Custom algorithms are proposed for synthesis of systems with various configurations. Our work on network composition using Reo with priority channels coupled with a constraint solver to derive automata-like semantics for the composed system can potentially be applied for controller synthesis.

In [24], a framework is presented to encode semantics of Reo networks as CSP with predicates in the form of binary propositions and numerical constraints. An advantage of this method is handling data constraints symbolically and, hence, mitigating the state explosion problem of automata models. Our current work directly extends the aforementioned framework to handle priority constraints.

Among other formal semantics of Reo, connector coloring comes with a limited notion of priority based on the context information. The context information affects otherwise non-deterministic data-flow choices. In [52], an automata-based semantics is proposed, which associates a preference for each transitions. A transition of lower preference is fired iff no more preferred transition can occur.

RLTS is comparable with Reo automata [53], a context-dependent formal semantic of Reo. A transition in Reo automata is labeled with a *guard*, which is a Boolean predicate in disjunctive normal form expressing positive and negative information about presence or absence of I/O requests, and a *firing* set that models the occurring I/O operations in the transition. The second set in RLTS transitions (the set of ends that provide reason for no flow) correspond to the negated elements of the guards in Reo automata, while the set of ends with flow relates to both the firing set and the positive elements of the guards.

7. Conclusions and future work

In this paper, we addressed the problem of priority flow modelling using the Reo coordination language. We extended the unified constraint-based semantics of Reo with binary and numeric priority constraints, showed correctness of our approach for the binary case and evaluated the performance of the algorithm for solving the RCSP to derive the semantics of a Reo network given the behavior of its constituent elements. We also illustrated the use of our framework for modeling business processes with priority flow.

As part of our ongoing work, we are using this framework to encode other aspects of the semantics of Reo, specifically, timed behavior. A promising area for future work is to use our framework for constraint-based model checking of Reo networks with priority.

References

[1] M. Havey, Essential Business Process Modeling, O'Reilly Media, Inc., 2005.

[2] R. Dijkman, J. Hofstetter, J. Koehler, Business Process Model and Notation, Springer, 2011.

[3] W. M. P. van der Aalst, A. H. M. ter Hofstede, M. Weske, Business Process Management: A Survey, in: Proceedings of the International Conference on Business Process Management (BPM), Vol. 2678 of LNCS, Springer, 2003, pp. 1–12.

[4] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Springer, 2013.

[5] R. Lu, S. Sadiq, A Survey of Comparative Business Process Modeling Approaches, in: Proceedings of the International Conference on Business Information Systems (BIS), Vol. 4439 of LNCS, Springer, 2007, pp. 82–94.

[6] W. M. P. van der Aalst, Business Process Management Demystified: a Tutorial on Models, Systems and Standards for Workflow Management, in: Lectures on Concurrency and Petri Nets: Advances in Petri Nets, Vol. 3098 of LNCS, Springer, 2004, pp. 1–65.

[7] W. Reisig, Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, Springer, 2013.

[8] U. Mutarraf, K. Barkaoui, Z. Li, N. Wu, T. Qu, Transformation of business process model and notation models onto petri nets and their analysis, Advances in Mechanical Engineering 10 (12) (2018) 1687814018808170.

[9] R. Bruni, H. Melgratti, U. Montanari, Theoretical foundations for compensations in flow composition languages, in: Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM, 2005, pp. 209 – 220.

[10] M. Butler, T. Hoare, C. Ferreira, A Trace Semantics for Long-running Transactions, in: Communicating Sequential Processes: The First 25 Years, Vol. 3525 of LNCS, 2005, pp. 133 – 150.

[11] V. Valero, H. Maciá, J. J. Pardo, M. E. Cambronero, G. Díaz, Transforming web services choreographies with priorities and time constraints into prioritized-time colored Petri nets, Science of Computer Programming 77 (3) (2012) 290 – 313.

[12] J. Padberg, Reconfigurable Petri Nets with Transition Priorities and Inhibitor Arcs, in: The Proceedings of the International Conference on Graph Transformation (ICGT), Vol. 9151 of LNCS, Springer, 2015, pp. 104–120.

[13] F. Bause, Analysis of Petri nets with a dynamic priority method, in: Proceedings of the International Conference on Application and Theory of Petri Nets, Vol. 1248 of LNCS, Springer, 1997, pp. 215–234.

[14] E. Best, M. Koutny, Petri net semantics of priority systems, Theoretical Computer Science 96 (1) (1992) 175 – 215.

[15] F. Arbab, Reo: a Channel-Based Coordination Model for Component Composition, Mathematical Structures in Computer Science 14 (2004) 329–366.

[16] B. Changizi, N. Kokash, F. Arbab, A Unified Toolset for Business Process Model Formalization, in: Proceedings of the International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA), ENTCS, 2010, pp. 147–156.

[17] D. Schumm, O. Turetken, N. Kokash, A. Elgammal, F. Leymann, W. van den Heuvel, Business Process Compliance through Reusable Units of Compliant Processes, in: Current Trends in Web Engineering, Vol. 6385 of LNCS, Springer, 2010, pp. 325–337.

[18] N. Kokash, F. Arbab, Formal design and verification of long-running transactions with Extensible Coordination Tools, IEEE Transactions on Services Computing 6 (2) (2013) 186–200.

[19] S. T. Q. Jongmans, F. Santini, M. Sargolzaei, F. Arbab, H. Afsarmanesh, Automatic code generation for the orchestration of web services with Reo, in: Proceedings of the International Conference on Service-Oriented and Cloud Computing (ESOCC), Vol. 7592 of LNCS, Springer, 2012, pp. 1–16.

[20] S. Meng, F. Arbab, Web Services Choreography and Orchestration in Reo and Constraint Automata, in: Proceedings of the ACM Symposium on Applied Computing, ACM, 2007, pp. 346–353.

[21] S. Jongmans, F. Arbab, Overview of Thirty Semantic Formalisms for Reo, Scientific Annals of Computer Science 22 (2012) 201–251.

[22] C. Baier, M. Sirjani, F. Arbab, J. J. M. M. Rutten, Modeling Component Connectors in Reo by Constraint Automata, Science of Computer Programming 61 (2) (2006) 75–113.

[23] D. Clarke, D. Costa, F. Arbab, Connector Colouring I: Synchronisation and Context Dependency, Science of Computer Programming 66 (3) (2007) 205–225.

[24] B. Changizi, N. Kokash, F. Arbab, A Constraint-based Method to Compute Semantics of Channel-based Coordination Models, in: Proceedings of the International Conference on Software Engineering Advances (ICSEA), 2012, pp. 530–539.

[25] R. Barták, M. Salido, F. Rossi, New trends in constraint satisfaction, planning, and scheduling: A survey, Knowledge Eng. Review 25 (2010) 249–279. doi:10.1017/S0269888910000202.

[26] G. Glorian, Nacre, in: C. Lecoutre, O. Roussel (Eds.), Proceedings of the 2018 XCSP3 Competition, 2019, pp. 85–85.
URL http://arxiv.org/abs/1901.01830

[27] B. Changizi, N. Kokash, F. Arbab, Service Orchestration with Priority Constraints, in: Proceedings of the International Conference on Fundamentals of Software Engineering (FSEN), Vol. 11761 of LNCS, Springer, 2019, pp. 194–209.

[28] F. Arbab, Puff, The Magic Protocol, in: Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday, 2011, pp. 169–206.

[29] F. Arbab, N. Kokash, M. Sun, Towards using Reo for Compliance-aware Business Process Modelling, in: Proceedings of the International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), Vol. 17 of LNCS, Springer, 2008, pp. 108–123.

[30] W. Aalst, van der, J. Nakatumba, A. Rozinat, N. Russell, Business process simulation: how to get it right?, BPM Reports, BPM Center, 2008.

[31] N. Kokash, B. Changizi, F. Arbab, A semantic model for service composition with coordination time delays, in: Proceedings of the 12th International Conference on Formal Engineering Methods and Software Engineering, ICFEM'10, 2010, pp. 106–121.

[32] Constreofy, a tool to generate semantics for Reo using constraint satisfaction techniques (2020).
URL https://github.com/behnaaz/constreofy

[33] REDUCE, a portable general-purpose computer algebra system (2008). URL https://sourceforge.net/projects/reduce-algebra/

[34] D. Clarke, D. Costa, F. Arbab, Connector Colouring I: Synchronisation and Context Dependency, ENTCS 154 (1) (2006) 101–119.

[35] Extensible coordination tools (2008). URL http://reo.project.cwi.nl/reo/wiki/Tools

[36] N. Kokash, C. Krause, E. Vink, de, Reo + mcrl2 : a framework for model-checking dataflow in service compositions, Formal Aspects of Computing 24 (2) (2012) 187–216. doi:10.1007/s00165-011-0191-6.

[37] O. Kopp, D. Martin, D. Wutke, F. Leymann, The difference between graph-based and block-structured business process modelling languages, Enterprise Modelling and Information Systems Architectures 4 (2009) 3–13.

[38] M. B. Juric, Business Process Execution Language for Web Services BPEL and BPEL4WS, 2nd Edition, Packt Publishing, 2006.

[39] N. Russell, W. M. van van der Aalst, A. H. M. ter Hofstede, Workflow Patterns: The Definitive Guide, The MIT Press, 2016.

[40] S. Tasharofi, M. Vakilian, R. Zilouchian Moghaddam, M. Sirjani, Modeling web service interactions using the coordination language reo, in: M. Dumas, R. Heckel (Eds.), Web Services and Formal Methods, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 108–123.

[41] H. Groefsema, D. Bucur, A survey of formal business process verification: From soundness to variability, in: Third International Symposium on Business Modeling and Software Design, SciTePress, 2013, p. 198–203.

[42] M. Mongiello, D. Castelluccia, Modelling and verification of bpel business processes, in: 4th Workshop on Model-Based Development of Computer-Based Systems and 3rd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06), 2006, pp. 148–153.

[43] W. van der Aalst, A. H. M. T. Hofstede, Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages, Tech. Rep. DAIMI PB-560, University of Aarhus (2002).

[44] W. M. P. van der Aalst, A. H. M. ter Hofstede, Yawl: Yet another workflow language, Information Systems 30 (4) (2005) 245–275.

[45] G. Balbo, Introduction to Stochastic Petri Nets, in: Lectures on Formal Methods and Performance Analysis, Vol. 2090 of LNCS, Springer, 2001, pp. 84–155.

[46] H. Klaudel, F. Pommereau, A concurrent and compositional petri net semantics of preemption, in: W. Grieskamp, T. Santen, B. Stoddart (Eds.), Integrated Formal Methods, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 318–337.

[47] I. A. Lomazova, L. Popova-Zeugmann, Controlling Petri Net Behavior using Priorities for Transitions, Fundamenta Informaticae 143 (1-2) (2016) 101–112.

[48] D. Clarke, J. Proença, A. Lazovik, F. Arbab, Channel-based Coordination via Constraint Satisfaction, Science of Computer Programming 76 (8) (2011) 681 – 710.

[49] R. Cleaveland, M. Hennessy, Priorities in process algebra, Information and Computation 87 (1990) 58–77.

[50] R. Cleaveland, G. Lüttgen, V. Natarajan, Priority and Abstraction in Process Algebra, Information and Computation 205 (9) (2007) 1426–1458.

[51] O. Stursberg, C. Hillmann, Decentralized Optimal Control of Distributed Interdependent Automata With Priority Structure, IEEE Transactions on Automation Science and Engineering 14 (2) (2017) 785–796.

[52] T. Kappé, F. Arbab, C. L. Talcott, A Compositional Framework for Preference-Aware Agents, in: Proceedings of the The Workshop on Verification and Validation of Cyber-Physical Systems (V2CPS), 2016, pp. 21–35.

[53] M. Bonsangue, D. Clarke, A. Silva, A Model of Context-dependent Component Connectors, Science of Computer Programming 77 (6) (2012) 685–706.