Science of Computer Programming 110 (2015) 78–103

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reasoning about software reconfigurations: The behavioural and structural perspectives

Nuno Oliveira, Luís S. Barbosa

HASLab, INESC TEC, Universidade do Minho, Braga, Portugal

ARTICLE INFO

Article history:
Received 8 July 2013
Received in revised form 24 May 2015
Accepted 29 May 2015
Available online 26 June 2015

Keywords:
Software reconfiguration
Software architecture
Coordination
Reo

ABSTRACT

Software connectors encapsulate interaction patterns between services in complex, distributed service-oriented applications. Such patterns encode the interconnection between the architectural elements in a system, which is not necessarily fixed, but often evolves dynamically. This may happen in response to faults, degrading levels of QoS, new enforced requirements or the re-assessment of contextual conditions. To be able to characterise and reason about such changes became a major issue in the project of trustworthy software.
This paper discusses what reconfiguration means within coordination-based models of software design. In these models computation and interaction are kept separate: components and services interact anonymously through specific connectors encoding the coordination protocols. In such a setting, of which Reo is a paradigmatic illustration, the paper introduces a model for connector reconfigurations, from both a *structural* and a *behavioural* perspective.

1. Introduction

Complex distributed service-oriented systems require reliable and yet flexible architectures. A clear separation between typical *loci* of computation (*e.g.,* services or components) and the protocols that manage their interaction is at the heart of what are called *exogenous coordination* models [1]. Actually, interaction is mediated by software connectors, such as in Reo [2], which offer powerful "glue-code" to express such interaction protocols, while maintaining the envisaged separation of concerns.

These systems often evolve at runtime, entailing the need for dynamic reconfiguration of their interaction protocols. This is typically motivated by runtime faults, the need to cope with new requirements or the change in contextual conditions, which often degrades the expected levels of quality of service (QoS) to unacceptable levels [3].

Conventionally, an architectural reconfiguration mainly targets the manipulation (*e.g.,* substitution, update or removal) of components, often disregarding the connectors, or otherwise taking them as yet another component [4–8]. In exogenous systems, however, reconfigurations may target the interaction protocols themselves, *i.e.,* the connector's structure, as discussed, for instance, by C. Krause [9]. Reconfigurations may thus substitute, add or remove communication channels, or move communication interfaces between components, in order to restructuring a complex interaction policy.

Connector reconfiguration mechanisms play, in this setting, a major role to express change and adaptation of interaction protocols. Moreover, identifying and understanding the consequences of applying these reconfigurations is an important issue for the correct design of reconfigurable systems.

E-mail address: nunooliveira@di.uminho.pt (N. Oliveira).

http://dx.doi.org/10.1016/j.scico.2015.05.013
0167-6423/

This paper combines and extends our previous work [10,11], to set up a conceptual framework for modelling and reasoning about reconfigurations of software connectors. Connectors are syntactically represented as graphs of communication primitives (*e.g.,* channels), referred to as *coordination patterns*, whose nodes stand for interaction points. Edges are labelled with identifiers and types which characterise their behaviour. The framework defines a number of elementary reconfiguration primitives, as well as how they can be combined to yield 'big-step' reconfiguration patterns able to transform significant parts of an architecture. Two complementary perspectives are introduced to reason about coordination pattern reconfigurations. One is *structural* and independent of whatever semantics is chosen for coordination patterns. It is concerned with requirements that, for example, enforce that during a reconfiguration a specific type of communication primitive remains attached to another specific primitive or a set of primitives. Such properties are specified in a propositional hybrid logic interpreted over the graph underlying the coordination pattern. The second perspective, on the other hand, is *behavioural* and relies on the specific semantics of coordination patterns. It may be of use, for example, to discuss to what extent a reconfiguration preserves the original interaction *behaviour* of a configuration.

It is important to emphasise, however, that reconfigurations in this paper are regarded from a static point of view. Intended for the software design phase, this approach allows the software architect (*i*) to specify connector reconfigurations; and (*ii*) to express and analyse their properties, either from a behavioural or structural perspective. However, the problematic of their dynamic application, including possible mechanisms for keeping the system's consistency along a run-time reconfiguration, is not addressed here.

Contributions The main contributions of this paper are (*i*) the development of a model for connector reconfigurations, their composition and application; (*ii*) a formal framework for comparing reconfigurations along the behavioural and the structural dimensions; and (*iii*) a case study from the e-healthcare domain illustrating the approach. Our previous work reported in references [10] and [11] introduced a notion of a reconfiguration pattern and discussed a first experiment on the use of a hybrid logic to express structural properties of coordination protocols, respectively. Both topics, however, are largely developed in this paper, leading to a new semantic model and a number of results for the logic, including a characterisation of bisimilarity and the proof of a Hennessy–Milner-like theorem on the equivalence between the assertion of two models being bisimilar and satisfying the same hybrid formulas.

Outline Coordination patterns are discussed in the next section paving the way to the detailed characterisation of reconfiguration operations in Section 3. The latter are combined to yield 'big-step' reconfiguration patterns. The following two sections introduce mechanisms for reasoning about connector reconfigurations from two orthogonal perspectives: a *behavioural* one (in Section 4), relying on whatever semantics is chosen for the underlying coordination model, and a *structural* one (in Section 5), in which properties of channel interconnection are expressed in a variant of propositional hybrid logic. Moreover, Section 5 defines bisimilarity for the structural models and proves a Hennessy–Milner-like theorem. A case study, taken from the e-healthcare domain, is discussed in Section 6 to illustrate the application of the approach proposed in the paper. Related work is discussed in Section 7. Finally, Section 8 concludes the paper and points out a number of open issues.

Notation Standard mathematical notation, namely for logic, is used throughout the paper. Maybe not so common is the representation of the powerset of a set X by 2^X and its extension to functions: $2^f(X) = \{f\, x \mid x \in X\}$. Function the and projections π_i, for i a natural number, are used to retrieve the unique element of a singleton set and the ith component of a tuple, respectively. Finally, if S is a set of sets notation $\bigcup S$ refers to the (iterated) union of all its elements, *i.e.,* $\bigcup S = \{x \in X \mid X \in S\}$.

2. Coordination patterns

Software connectors encoding reusable solutions for architectural problems in distributed, loosely-coupled systems are called in this paper *coordination patterns*. They are specified as graphs whose nodes represent interaction points and edges, standing for *communication primitives*, are labelled by pairs composed of their identifier and type. To keep exposition concrete, the Reo coordination model [1,2] (in particular, its set of primitive channels) is adopted, in the sequel, to provide types (and consequently a semantics) to the communication primitives in a coordination pattern.

In order to keep the paper reasonably self-contained, a brief introduction to Reo is given below. Coordination patterns are introduced afterwards.

2.1. A primer on Reo

Reo [1,2] is a popular model for exogenous service coordination based on channels. It is compositional, in the sense that complex coordination structures are obtained from the combination of channels.

2.1.1. Channels, nodes and connectors

A Reo channel is a point-to-point communication device with exactly two directed ends and a behaviour (or coordination protocol) defined within a specific semantic model. A channel *end* may accept or dispense data, in which cases it is said to be a *source*, or a *sink* end, respectively. Normally, a channel has one source and one sink end, but Reo also allows channels to

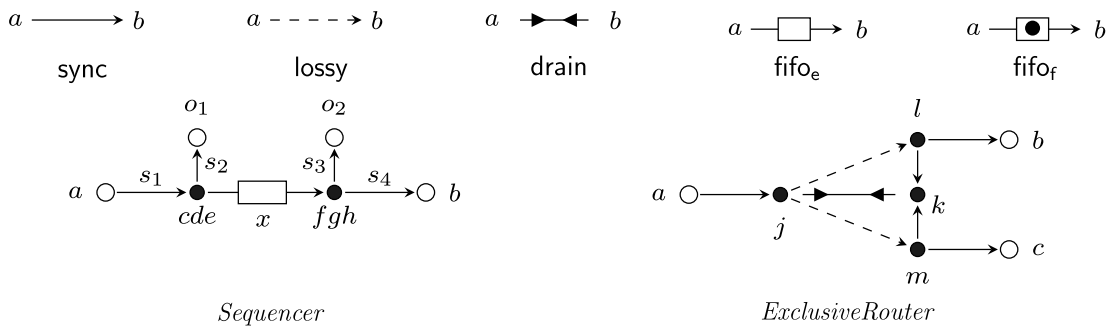


Fig. 1. Primitive Reo channels (above) and two connectors (below).

have two source or two sink ends. Channel ends can be joined into *nodes*, which may be of three distinct types: (*i*) a source node connects only source channel ends, (*ii*) a sink node connects only sink channel ends, and (*iii*) a mixed node which connects both. Source and sink nodes are also referred to as the *boundary nodes* or *ports* of a connector. Fig. 1 depicts a basic set of primitive channels in Reo and two connectors built from their composition.

Informally, the sync channel consumes data at its source end and transmits it through the sink end, provided that both ends can communicate atomically, *i.e.,* at the same logical time. Otherwise the channel blocks until communication is allowed to proceed. The lossy channel behaves like sync when both ends are prepared for communication. However, in the absence of a data request at the sink end, any data present at its source is taken and lost. The drain channel has two source ends from which data is taken synchronously. A fifo channel, on the other hand, has a buffering capacity of one memory position, allowing for asynchronous communication between its ends. Qualifiers e and f refer to the channel internal state (either *empty* or *full*, respectively).

The *Sequencer* and the *Exclusive Router* are examples of Reo connectors built from the composition of several primitive Reo channels, by joining their ends. A mixed node behaves as a replicator (respectively, a merger) when composed of a source (respectively, a sink) channel end and two or more sink (respectively, source) ends. In the first case it accepts data at the source end and replicates it to all connected sink ends. In the second, it merges non-deterministically to the sink end data selected from the connected source ends. Clearly, when a mixed node is composed of multiple source and sink ends, data is simultaneously merged and replicated, a behaviour referred in the Reo literature as *the pumping station*.

The *Sequencer* takes data from node a and transmits it to node o_1 and buffer x in a first synchronous step. Then it takes data from the buffer to nodes o_2 and b in a second synchronous step. The net effect is that nodes o_1 and o_2 receive data in sequence. The *ExclusiveRouter* connector takes data from node a and transmits it either to b or to c. In detail, data is transmitted to b (respectively, c) when this has pending requests, but there are no requests at node c (respectively, b). When there are pending requests at both b and c, the merger node k chooses non-deterministically one of these nodes to receive data. Graphically, the white circles represent the boundaries of the connectors, *i.e.,* source and sink nodes (used to link the connector to external services or other connectors), while the black ones represent mixed (internal) nodes.

2.1.2. Semantics

Several formal semantics have been proposed for Reo for the last ten years [12]. Some models describe data flow through timed data streams [13]; others through a labelling scheme with *colours* [14,15]; others still resort to some form of generalised automata. Let us recall two of these automata-based models, namely *constraint automata* [16,17] and *Reo automata* [18,19], briefly reviewed below.

Constraint automata Constraint automata [16,17] are defined over a set Σ of nodes representing the connector ports, and data constraints over Σ.

Data constraints, collected in a set DC, are given by the grammar:

$$g, g' \ni \texttt{true} \mid d_X = d_Y \mid g \vee g' \mid g \wedge g' \mid \neg g,$$

where d_X is for the data item associated to X, the latter standing either for a node in Σ or a data variable. Formally,

Definition 1. A constraint automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \longrightarrow, Q_0)$, where Q is a set of states, $Q_0 \subseteq Q$, is the set of initial states, Σ is a (finite) set of ports, and $\longrightarrow \subseteq Q \times 2^{\Sigma} \times DC \times Q$, is the transition relation: each state transition is labelled by the set of ports which become active on its firing and a set of data constraints.

Two binary relations, $\equiv$ and $\leq$, are defined over DC as follows: $g_1 \equiv g_2$ if g_1 and g_2 define equal data assignments; on the other hand, $g_1 \leq g_2$ if data assignments in g_1 imply those in g_2. Additionally, $dc(q, N, P) = \bigvee\{g : q \xrightarrow{N,g} p \wedge p \in P\}$ is the weakest data constraint that ensures the existence of a transition from q to any state in P, via a set of names N.

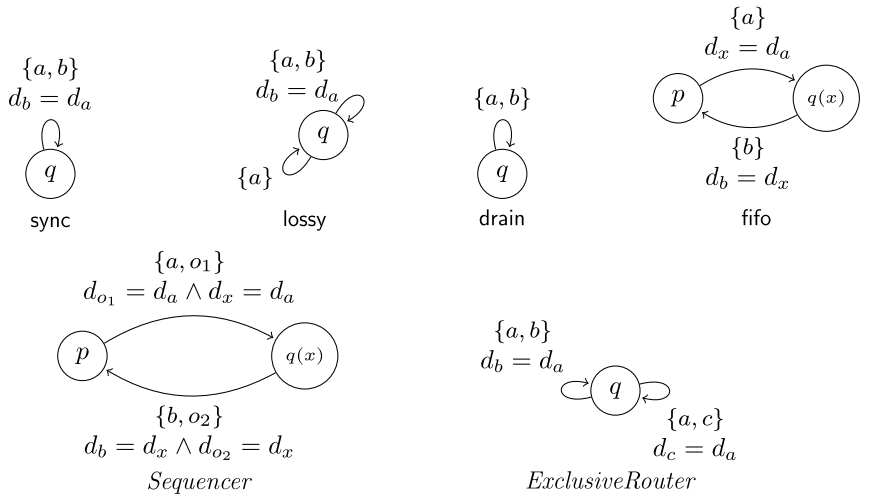


Fig. 2. Constraint automata for primitive Reo channels and two connectors.

Fig. 2 depicts the constraint automata for each of the primitive Reo channels and connectors shown in Fig. 1. Consider, for instance, the sync channel: label $\{a, b\}$ captures the fact that both ends a and b are synchronously activated, while the constraint $d_b = d_a$ specifies that data present in a is transmitted to b.

Constraint automata compose (composition is, as usual in automata theory, a combination of product and hide [16]). However, the model is unable to capture context dependent behaviour. For example, a constraint automata corresponding to a lossy channel models non-deterministically the choice between data flow and data loss, a decision which is intended to be (deterministically) made by the environment.

Reo automata Reo automata [18,19], on the other hand, are context-sensitive and act as acceptors of *guarded strings*. Formally, let $\Sigma = \{\sigma_1, \ldots, \sigma_2\}$ be a set of ports. The set of guards is the free Boolean algebra $\mathcal{B}_\Sigma$ over Σ generated by the following grammar

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

and represent constraints on the firing of a transition. Atomic guards, collected in a set $\mathbf{At}_\Sigma$, are conjunctions of $p, \overline{p}$, for $p \in \Sigma$. Intuitively they specify which ports are and are not enabled (*i.e.,* exhibiting pending requests or their absence). A guarded string over Σ is a sequence $\langle g_1, f_1\rangle \ldots \langle g_n, f_n\rangle$, for $n \geq 0$, $f_i \subseteq \Sigma$, where each g_i is a guard and each f_i stands for the ports that synchronously fire read/write operations. Relation $\leq$ on guards is defined as $g_1 \leq g_2 \iff g_1 \wedge g_2 = g_1$, thus expressing logic implication.

Definition 2. A Reo automaton $\mathcal{A}_{\text{Reo}}$ is a tuple (Σ, Q, δ), where Σ is a set of ports, Q is a set of states and $\delta \subseteq Q \times \mathcal{B}_\Sigma \times 2^\Sigma \times Q$ is the transition function which satisfies the *reactivity* and *uniformity* conditions.

A transition (q, g, f, q'), typically represented as $q \xrightarrow{g|f} q'$, says that if the connector is in state q and the port requests present at the moment, encoded as an atomic guard g', are such that $g' \leq g$, then the ports in f will fire and the connector will evolve to state q'. Intuitively, reactivity ensures that data flows through ports with pending requests, and uniformity enforces that the firing set of a port is a subset of its request set (see [19] for the formal definition).

Fig. 3 depicts the Reo automata for each of the primitive Reo channels and the two connectors considered in Fig. 1. In the sequel, notation $ab|ab$ denotes a guarded string element $\langle a \wedge b, \{a, b\}\rangle$.

As shown in the lossy channel example, the context-awareness is captured through the use of *negative* information in Reo automata. In this example, the operation in a fires (without synchronisation with b) when there is a request in a but not in b (represented by $\overline{b}$), as expressed in the guard.

2.2. Coordination patterns

Reo channels can be put together to form *coordination patterns*. The notion, essentially a methodological one, is intended to encode a reusable architectural solution for a coordination problem. Formally, it is defined as an abstract graph of communication primitives where edges are labelled with a pair formed by an identifier and a type, and nodes represent interaction points. Interaction points are locations where communication primitives synchronise their interfaces (*ends*) for interaction with other patterns or external components. When these locations are composed of more than one end, those

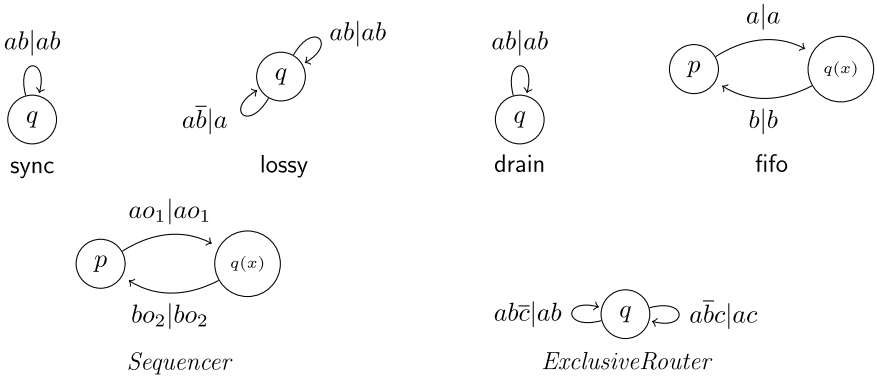


Fig. 3. Reo automata for primitive Reo channels and the *sequencer* connector.

are said *co-located*. In the remaining of this paper, references to the notion of *communication primitive* will be substituted by the more concrete one of a *channel*, without loss of generality.

Definition 3 *(Channel).* Let $\mathcal{E}$ be a set of channel ends, $\mathcal{I}$ a set of unique identifiers and $\mathcal{T} = \{\mathsf{sync}, \mathsf{lossy}, \mathsf{drain}, \mathsf{fifo_e}, \mathsf{fifo_f}, \ldots\}$ a set of channel types. A channel is a tuple

$$c = \langle \mathcal{S}, i, t, \mathcal{K} \rangle$$

where $i \in \mathcal{I}$, $t \in \mathcal{T}$, and $\mathcal{S}, \mathcal{K} \subseteq \mathcal{E}$, such that $\mathcal{S} \cap \mathcal{K} = \emptyset$, are the sets of source and sink ends, respectively.

Definition 4 *(Coordination pattern).* A coordination pattern is a pair

$$\rho = \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle$$

where $\mathcal{C}_\rho$ is a set of channels, and $\mathcal{N}_\rho$ is a set of nodes specified as a partition on the union of all ends of all channels in $\mathcal{C}_\rho$, such that,

1. channel identifiers are unique:

$$\forall_{c_1, c_2 \in \mathcal{C}_\rho} \,.\, \pi_2(c_1) = \pi_2(c_2) \;\Rightarrow\; c_1 = c_2$$

2. the number of channels sharing a node is never greater than the number of co-located ends in it:

$$\forall_{n \in \mathcal{N}_\rho} \,.\, |n| \geq |\{c \in \mathcal{C} \mid n \cap (\pi_1(c) \cup \pi_4(c)) \neq \emptyset\}|$$

Notation $\mathbf{0} = \langle \emptyset, \emptyset \rangle$ denotes the empty coordination pattern; $\mathcal{P}$ is the set of coordination patterns. Whenever clear from the context, subscript ρ in $\mathcal{C}_\rho$ and $\mathcal{N}_\rho$ will be omitted.

Operations $I(\rho) = \{i \in \mathcal{N}_\rho \mid \exists_{c \in \mathcal{C}_\rho} \,.\, i \cap \pi_1(c) \neq \emptyset \wedge in(i, \rho)\}$ and $O(\rho) = \{o \in \mathcal{N}_\rho \mid \exists_{c \in \mathcal{C}_\rho} \,.\, o \cap \pi_4(c) \neq \emptyset \wedge out(o, \rho)\}$, where $in(x, \rho) = \forall_{c \in \mathcal{C}_\rho} \,.\, x \cap \pi_4(c) = \emptyset$ and $out(x, \rho) = \forall_{c \in \mathcal{C}_\rho} \,.\, x \cap \pi_1(c) = \emptyset$, are used to retrieve, respectively, the set of source and sink nodes (*i.e.,* the I/O interface) of coordination pattern ρ.

The set of channel identifiers in a coordination pattern ρ is given by $\mathfrak{I}_\rho = 2^{\pi_2}(\mathcal{C}_\rho)$. Another auxiliary operation computes the set of ends of a channel (uniquely) identified by ch in the context of a coordination pattern ρ: $\mathfrak{E}^{ch}_\rho = \mathsf{let}\,(c = \mathsf{the}\{c' \in \mathcal{C}_\rho \mid \pi_2(c') = ch\})\ \mathsf{in}\ \pi_1(c) \cup \pi_4(c)$. Finally, $\mathfrak{N}^{ch}_\rho = \{n \in \mathcal{N}_\rho \mid \mathfrak{E}^{ch}_\rho \cap n \neq \emptyset\}$ retrieves the nodes of the coordination pattern ρ where the ends of channel ch participate.

Example 1. An instance sc of a sync channel (*cf.,* Fig. 1) is written as $\langle \{a\}, sc, \mathsf{sync}, \{b\} \rangle$. Plugging a drain channel (for example, the one specified as $\langle \{c, d\}, sd, \mathsf{drain}, \emptyset \rangle$) to its output end b yields the *DummySynchroniser* pattern $\rho = \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\} \rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset \rangle\}, \{\underline{a}, \underline{d}, \underline{bc}\} \rangle$[1] which allows data to be written on a, if there exist pending requests at d. Fig. 4 depicts its topology.

Finally, $I(\rho) = \{\underline{a}, \underline{d}\}$ and $O(\rho) = \emptyset$.

In the sequel a visual, Reo-like representation of coordination patterns (as in Fig. 1) will be adopted. To increase readability channel identifiers are only added to the visual representation when needed.

[1] In order to improve readability, nodes $\{a, b\}$ are written as $\underline{ab}$; accordingly, a set of nodes $\{\{c\}, \{d\}\}$ is denoted by $\{\underline{c}, \underline{d}\}$.

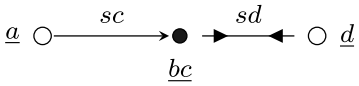


Fig. 4. The *DummySynchroniser* coordination pattern.

3. Reconfigurations

In an exogenous coordination framework, the focus of reconfigurations is the set of coordination pattern themselves, instead of the plugged-in components which are considered external services. Therefore, any change to the original structure of a coordination pattern qualifies as a reconfiguration. These changes are driven by the sequential application of a number of primitive reconfiguration operations which manipulate the basic elements of a coordination pattern.

3.1. Primitive reconfiguration operations

Primitive reconfiguration operations change atomically the basic structure of a coordination pattern, namely, its nodes and channels. We consider the following five primitive operations:

$$Prim = \{\mathsf{const}_\rho, \mathsf{par}_\rho, \mathsf{join}_P, \mathsf{split}_p, \mathsf{remove}_{ch} \mid \rho \in \mathcal{P}, P \subseteq \mathcal{N}, p \in \mathcal{N}, ch \in \mathcal{I}\}$$

where $\mathcal{N}$ is the set of all nodes.

The application of a primitive reconfiguration r to a coordination pattern ρ, denoted by $\rho \bullet r$, yields a new coordination pattern suitably modified. The semantics of $\bullet$ is defined below for each primitive reconfiguration. In the sequel, conditional expressions are written as $(\phi \rightarrow e_1, e_2)$ (read *return* e_1 *if* ϕ *holds,* e_2 *otherwise).*

The most trivial reconfiguration is the constant one, along which the original coordination pattern is replaced by a new one. Formally:

Definition 5 *(*const*).* Let $\rho_1, \rho_2 \in \mathcal{P}$. Then,

$$\rho_1 \bullet \mathsf{const}_{\rho_2} = \rho_2$$

The par operation sets the original coordination pattern in parallel with the one given as a parameter without creating any connection between them. It assumes, without loss of generality, that nodes and channel identifiers in both patterns are disjoint.

Definition 6 *(*par*).* Let $\rho_1, \rho_2 \in \mathcal{P}$. Then,

$$\rho_1 \bullet \mathsf{par}_{\rho_2} = \langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle$$

The join operation performs connections in the coordination pattern by merging a set of nodes into a single one.

Definition 7 *(*join*).* Let $\rho \in \mathcal{P}$, and $N \subseteq \mathcal{N}$. Then

$$\rho \bullet \mathsf{join}_N = \langle \mathcal{C}_\rho, (N \subseteq \mathcal{N}_\rho \rightarrow \{\bigsqcup N\} \cup (\mathcal{N}_\rho \setminus N), \mathcal{N}_\rho) \rangle$$

Example 2. Consider

$$\rho = \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\} \rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset \rangle\}, \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\} \rangle$$

and use the join primitive to obtain the coordination pattern in Fig. 4. The relevant operation is $\rho \bullet \mathsf{join}_{\{\underline{b},\underline{c}\}}$. Since $\{\underline{b}, \underline{c}\} \subseteq \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\} = \emptyset$, this entails computing $\{\bigsqcup\{\underline{b}, \underline{c}\}\} = \{\underline{bc}\}$ and $\mathcal{N}_\rho \setminus \{\underline{b}, \underline{c}\} = \{\underline{a}, \underline{d}\}$, whose union is $\{\underline{a}, \underline{d}, \underline{bc}\}$. Therefore,

$$\rho \bullet \mathsf{join}_{\{\underline{b},\underline{c}\}} = \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\} \rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset \rangle\}, \{\underline{a}, \underline{d}, \underline{bc}\} \rangle.$$

The split primitive reconfiguration is dual to join. It breaks connections within a coordination pattern by separating all channel ends co-located on a given node.

Definition 8 *(*split*).* Let $\rho \in \mathcal{P}$ and $n \in \mathcal{N}$. Then,

$$\rho \bullet \mathsf{split}_n = \langle \mathcal{C}_\rho, (n \in \mathcal{N}_\rho \rightarrow 2^{\mathsf{sing}}\, n \cup (\mathcal{N}_\rho \setminus \{n\}), \mathcal{N}_\rho) \rangle$$

where $\mathsf{sing}(x) = \{x\}$.

Example 3. Let ρ stand for the coordination pattern in Fig. 4. Applying the $\rho \bullet \mathsf{split}_{\underline{bc}}$ primitive makes it possible to retrieve the initial pattern of the previous example. Since $\underline{bc} \in \{\underline{a}, \underline{d}, \underline{bc}\}$, compute $2^{\mathsf{sing}}\,\underline{bc} = \{\underline{b}, \underline{c}\}$ and $\mathcal{N}_{\rho_1} \setminus \{\underline{bc}\} = \{\underline{a}, \underline{d}\}$, whose union is $\{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}$. Therefore,

$$\rho \bullet \mathsf{split}_{\underline{bc}} \;=\; \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{b}, \underline{c}, \underline{d}\}\rangle.$$

Finally, the remove operation removes a channel from a coordination pattern and updates the nodes in which the channel synchronised its ends.

Definition 9 (remove). Let $\rho \in \mathcal{P}$ and $ch \in \mathfrak{I}_\rho$. Then,

$$\begin{aligned} \rho \bullet \mathsf{remove}_{ch} = \;& \mathsf{let}\;\; R = \{e \in \mathcal{C}_\rho \mid \pi_2(e) = ch\} \\ & \quad\; N = (R \neq \emptyset \rightarrow (2^{\mathsf{minus}_{\mathfrak{E}^{ch}_\rho}}\,\mathcal{N}_\rho) \setminus \{\emptyset\},\, \mathcal{N}_\rho) \\ & \mathsf{in}\;\; \langle \mathcal{C}_\rho \setminus R,\, N\rangle \end{aligned}$$

where $\mathsf{minus}_E(X) = X \setminus E$.

Example 4. Consider again the coordination pattern ρ depicted in Fig. 4, but extended with a $\mathsf{fifo_e}$ connected to channels sync and drain as follows:

$$\rho = \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle \{e\}, q, \mathsf{fifo_e}, \{f\}\rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}\{\underline{a}, \underline{be}, \underline{fc}, \underline{d}\}\rangle$$

The original pattern can be obtained as $(\rho \bullet \mathsf{remove}_q) \bullet \mathsf{join}_{\{\underline{b},\underline{c}\}}$ On computing $\rho \bullet \mathsf{remove}_q$ note that $R = \{\langle \{e\}, q, \mathsf{fifo_e}, \{f\}\rangle\}$ and $N = (2^{\mathsf{minus}_{\{e,f\}}}\,\{\underline{a}, \underline{be}, \underline{fc}, \underline{d}\}) \setminus \{\emptyset\} = \{\underline{a}, \underline{bc}, \underline{d}\}$. Thus,

$$(\rho \bullet \mathsf{remove}_q) \bullet \mathsf{join}_{\{\underline{b},\underline{c}\}} \;=\; \langle \{\langle \{a\}, sc, \mathsf{sync}, \{b\}\rangle, \langle \{c, d\}, sd, \mathsf{drain}, \emptyset\rangle\}, \{\underline{a}, \underline{bc}, \underline{d}\}\rangle.$$

The following results characterise the effect of primitive reconfigurations. Lemma 1 shows that the set $\mathcal{P}$ of coordination patterns is closed under the application of these primitives. Lemma 2 identifies contexts which are unaffected by reconfigurations.

Lemma 1. *The set $\mathcal{P}$ of coordination patterns is closed under the application of reconfigurations in Prim.*

Proof. Let $\rho, \rho' \in \mathcal{P}$, $N \subseteq \mathcal{N}$; $n \in \mathcal{N}$, $ch \in \mathcal{I}$ and $r \in Prim$. For each primitive reconfiguration let us check the properties in Definition 4. Thus,

- For $\rho \bullet \mathsf{const}_{\rho'}$, all properties hold since ρ and ρ' are assumed well-formed.
- For $\rho \bullet \mathsf{par}_{\rho'}$, the resulting pattern is the component-wise union of ρ and ρ', which are assumed to be disjoint. Therefore, since ρ and ρ' are well-formed and union does not change their specification, well-formedness is preserved.
- For $\rho \bullet \mathsf{join}_N$, all the nodes in N are merged together. Condition 1 in Definition 4 is preserved because ρ is well formed and no channels are added to it. The second condition in the same definition also holds because each node in N preserves the inequality. By merging these nodes into one, the inequality still remains, because the nodes are disjoint partitions of channel ends.
- For $\rho \bullet \mathsf{split}_n$, all the channel ends co-located in n are separated into simple nodes. Condition 1 is preserved because ρ is well formed and no channels are added to it. Condition 2 is also preserved because, ends being unique, the separation of node n results in $|n|$ nodes, each of which is formed by a single channel end naturally associated to a single channel.
- As a result of $\rho \bullet \mathsf{remove}_{ch}$ the channel identified by ch is removed and its ends are removed from the nodes to which it was previously connected. Condition 1 is preserved because no channel is added to the well-formed pattern ρ, and so is condition 2. Because ρ is well-formed, then each node n to which ch is connected preserves the inequality. By removing ends of ch from n it is obtained either $|n| = 0$, in this case the node is removed from ρ; or $|n| > 0$, meaning that a number k of channels share ends in n. Then, either each of these channels contribute with one end to n, yielding $n = k$; or they contribute with more than one end to n, yielding $n \geq k$. □

Lemma 2. *Let $\rho, \rho' \in \mathcal{P}$; $P \subseteq \mathcal{N}$; $p \in \mathcal{N}$ and $ch \in \mathcal{I}$. The following properties, stated as strict equalities between coordination patterns, hold:*

$$\rho \bullet \mathsf{const}_{\rho'} = \rho \text{ if } \rho = \rho' \tag{1}$$

$$\rho \bullet \mathsf{join}_P = \rho \text{ if } P \setminus \mathcal{N}_\rho \neq \emptyset \tag{2}$$

$$\rho \bullet \mathsf{split}_p = \rho \text{ if } p \notin \mathcal{N}_\rho \tag{3}$$

$$\rho \bullet \mathsf{remove}_{ch} = \rho \text{ if } \forall_{c \in \mathfrak{I}_\rho} c \neq ch \tag{4}$$

Proof. The lemma guarantees that reconfiguration operations that do not affect elements of the coordination pattern have no effect. All of them come easily from the definitions. For (4), note that if $\forall_{c \in \mathfrak{I}_\rho} c \neq ch$ then $\{e \in \mathcal{C}_\rho \mid \pi_2(e) = ch\} = \emptyset$. Therefore, $\rho \bullet \mathsf{remove}_{ch} = \langle \mathcal{C}_\rho \setminus \{\emptyset\}, \mathcal{N}_\rho \rangle = \rho$. □

3.2. Composing reconfigurations

In most cases, the application of a single primitive reconfiguration is not enough. Single steps, however, can be combined *sequentially*.

Definition 10. Let $\rho \in \mathcal{P}$ and r_1, r_2 be two reconfigurations. The application of r_1 followed by r_2 is given by

$$\rho \bullet \{r_1 \ ; \ r_2\} = (\rho \bullet r_1) \bullet r_2$$

Lemma 3. *The set $\mathcal{P}$ of coordination patterns is closed for sequential composition.*

Proof. The proof is by induction on the structure of reconfigurations. The base case, of primitive reconfigurations, is already proved in Lemma 1. Consider now $\rho \bullet \{r_1; r_2\}$, and assume, without loss of generality that r_2 is a primitive reconfiguration. If not, r_2 can always be rewritten as a sequence of reconfigurations $r'_1, r'_2, \cdots$, such that its last element is a primitive reconfiguration. By induction hypothesis $\rho \bullet r_1$ is in $\mathcal{P}$. Then conclude by Lemma 1, for r_2 primitive. □

The following lemma introduces a number of properties of primitive reconfigurations, stated as strict equalities between coordination patterns.

Lemma 4. *Let $\rho, \rho_1, \rho_2 \in \mathcal{P}$, $\{n\}, N, N_1, N_2 \in \mathcal{N}$. Then,*

$$\rho \bullet \mathsf{const}_{\rho_1} = \rho_1 \tag{5}$$

$$\rho \bullet \mathsf{par}_{\rho_1} = \rho_1 \bullet \mathsf{par}_{\rho} \tag{6}$$

$$(\rho \bullet \mathsf{par}_{\rho_1}) \bullet \mathsf{par}_{\rho_2} = \rho \bullet \mathsf{par}_{\rho_1 \bullet \mathsf{par}_{\rho_2}} \tag{7}$$

$$(\rho \bullet \mathsf{join}_{N_1}) \bullet \mathsf{join}_{N_2} = (\rho \bullet \mathsf{join}_{N_2}) \bullet \mathsf{join}_{N_1} \quad \textit{if } \bigcup N_1 \cap \bigcup N_2 = \emptyset \tag{8}$$

$$(\rho \bullet \mathsf{join}_{N}) \bullet \mathsf{split}_{n} = \rho \quad \textit{if } \bigcup N = n \wedge \forall_{p \in N} \cdot |p| = 1 \tag{9}$$

$$(\rho \bullet \mathsf{split}_{n}) \bullet \mathsf{join}_{N} = \rho \quad \textit{if } \bigcup N = n \wedge \forall_{p \in N} \cdot |p| = 1 \tag{10}$$

$$(\rho \bullet \mathsf{par}_{\iota\langle \mathcal{S}, i, t, \mathcal{K} \rangle}) \bullet \mathsf{remove}_{i} = \rho \tag{11}$$

where $\iota\langle \mathcal{S}, i, t, \mathcal{K} \rangle = \langle \{\langle \mathcal{S}, i, t, \mathcal{K} \rangle\}, \{\{\mathcal{S}\}, \{\mathcal{K}\}\} \rangle$ regards channel $\langle \mathcal{S}, i, t, \mathcal{K} \rangle$ as a single coordination pattern.

Proof. Property (5) is an immediate consequence of the definition of const. Laws (6) and (7) are proved in a similar way resorting to commutativity and associativity of set union, respectively. Thus,

$$\begin{aligned}
& \rho \bullet \mathsf{par}_{\rho_1} \\
= \quad & \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho \cup \mathcal{C}_{\rho_1}, \mathcal{N}_\rho \cup \mathcal{N}_{\rho_1} \rangle \\
= \quad & \{ \cup \text{ commutative} \} \\
& \langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_\rho, \mathcal{N}_{\rho_1} \cup \mathcal{N}_\rho \rangle \\
= \quad & \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_{\rho_1}, \mathcal{N}_{\rho_1} \rangle \bullet \mathsf{par}_{\langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle} \\
= \quad & \{ \text{ projections} \} \\
& \rho_1 \bullet \mathsf{par}_{\rho}
\end{aligned}$$

For (7),

$$
\begin{aligned}
& (\rho \bullet \mathsf{par}_{\rho_1}) \bullet \mathsf{par}_{\rho_2} \\
= \quad & \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho \cup \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_\rho \cup \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle \\
= \quad & \{ \cup \text{ associative, definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \bullet \mathsf{par}_{\langle \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_1} \cup \mathcal{N}_{\rho_2} \rangle} \\
= \quad & \{ \text{ definition of } \mathsf{par} \ \} \\
& \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \bullet \mathsf{par}_{\langle \mathcal{C}_{\rho_1}, \mathcal{N}_{\rho_1} \rangle \bullet \mathsf{par} \langle \mathcal{C}_{\rho_2}, \mathcal{N}_{\rho_2} \rangle} \\
= \quad & \{ \text{ projections } \} \\
& \rho \bullet \mathsf{par}_{\rho_1 \bullet \mathsf{par} \rho_2}
\end{aligned}
$$

For the following laws, involving join and split, we restrict attention to the cases in which the reconfiguration actually changes the pattern (*i.e.,* in the *then*-cases in the conditional definition of both operators). The other cases are trivial. Thus, for (8),

$$
\begin{aligned}
& (\rho \bullet \mathsf{join}_{N_1}) \bullet \mathsf{join}_{N_2} \\
= \quad & \{ \text{ definition of } \mathsf{join} \ (\textit{then}\text{-case}) \text{ applied twice } \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N_2\} \cup (\{\bigcup N_1\} \cup (\mathcal{N}_\rho \setminus N_1)) \setminus N_2 \rangle \\
= \quad & \{ \text{ set difference distributes over union } \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N_2\} \cup \{\bigcup N_1\} \setminus N_2 \cup (\mathcal{N}_\rho \setminus (N_1 \cup N_2)) \rangle \\
= \quad & \{ \text{ condition } \textstyle\bigcup N_1 \cap \bigcup N_2 = \emptyset \text{ implies } \{\bigcup N_1\} \cap N_2 = \emptyset \ \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N_2\} \cup \{\bigcup N_1\} \cup (\mathcal{N}_\rho \setminus (N_1 \cup N_2)) \rangle \\
= \quad & \{ \text{ similarly } \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N_1\} \cup (\{\bigcup N_2\} \cup (\mathcal{N}_\rho \setminus N_2)) \setminus N_1 \rangle \\
= \quad & \{ \text{ definition of } \mathsf{join} \ (\textit{then}\text{-case}) \text{ applied twice } \} \\
& (\rho \bullet \mathsf{join}_{N_2}) \bullet \mathsf{join}_{N_1}
\end{aligned}
$$

For (9)

$$
\begin{aligned}
& (\rho \bullet \mathsf{join}_N) \bullet \mathsf{split}_n \\
= \quad & \{ \text{ definition of } \mathsf{join} \ (\textit{then}\text{-case}) \ \} \\
& \langle \mathcal{C}_\rho, \{\bigcup N\} \cup (\mathcal{N}_\rho \setminus N) \rangle \bullet \mathsf{split}_n \\
= \quad & \{ \text{ definition of } \mathsf{split} \ (\textit{then}\text{-case}) \ \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\{\bigcup N\} \cup (\mathcal{N}_\rho \setminus N)) \setminus \{n\} \rangle \\
= \quad & \{ \text{ assumption: } \textstyle\bigcup N = n \ \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\mathcal{N}_\rho \setminus N) \rangle \\
= \quad & \{ \text{ assumption: } \forall_{p \in N} \cdot |p| = 1 \ \} \\
& \langle \mathcal{C}_\rho, N \cup (\mathcal{N}_\rho \setminus N) \rangle \\
= \quad & \{ \text{ sets, projections } \} \\
& \rho
\end{aligned}
$$

For (10),

$$
\begin{aligned}
& (\rho \bullet \mathsf{split}_n) \bullet \mathsf{join}_N \\
= \quad & \{ \text{ definition of } \mathsf{split} \ (\textit{then}\text{-case}) \ \} \\
& \langle \mathcal{C}_\rho, 2^{\mathsf{sing}} n \cup (\mathcal{N}_\rho \setminus \{n\}) \rangle \bullet \mathsf{join}_N
\end{aligned}
$$

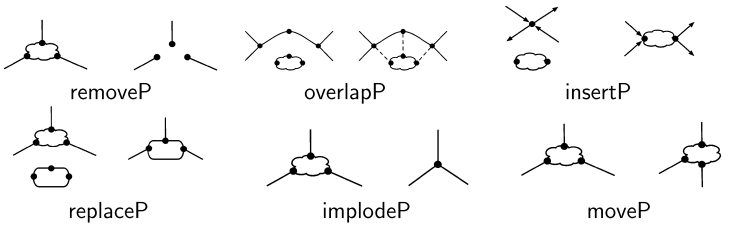


Fig. 5. Reconfiguration patterns.

$$
\begin{aligned}
&= \quad \{ \text{ assumptions: } \bigcup N = n \text{ (which implies } 2^{\mathsf{sing}} n = N) \text{ and } \forall_{p \in N} \cdot |p| = 1 \ \} \\
&\quad \langle \mathcal{C}_\rho, N \cup (\mathcal{N}_\rho \setminus \{n\}) \rangle \bullet \mathsf{join}_N \\
&= \quad \{ \text{ definition of } \mathsf{join} \text{ (\textit{then}-case) } \} \\
&\quad \langle \mathcal{C}_\rho, \{\bigcup N\} \cup (N \cup (\mathcal{N}_\rho \setminus \{n\})) \setminus N \rangle \\
&= \quad \{ \bigcup N = n; \text{ sets } \} \\
&\quad \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \\
&= \quad \{ \text{ projections } \} \\
&\quad \rho
\end{aligned}
$$

For (11),

$$
\begin{aligned}
&\quad (\rho \bullet \mathsf{par}_{\iota\langle \mathcal{S},i,t,\mathcal{K}\rangle}) \bullet \mathsf{remove}_i \\
&= \quad \{ \text{ definition of } \iota \ \} \\
&\quad (\rho \bullet \mathsf{par}_{\langle\{\langle \mathcal{S},i,t,\mathcal{K}\rangle\},\{\{\mathcal{S}\},\{\mathcal{K}\}\}\rangle}) \bullet \mathsf{remove}_i \\
&= \quad \{ \text{ definition of } \mathsf{par} \ \} \\
&\quad \langle \mathcal{C}_\rho \cup \{\langle \mathcal{S}, i, t, \mathcal{K}\rangle\}, \mathcal{N}_\rho \cup \{\{\mathcal{S}\},\{\mathcal{K}\}\} \rangle \bullet \mathsf{remove}_i \\
&= \quad \{ \text{ definition of } \mathsf{remove} \ \} \\
&\quad \langle \mathcal{C}_\rho \setminus \{\langle \mathcal{S}, i, t, \mathcal{K}\rangle\}, 2^{\mathsf{minus}_{\mathcal{S}\cup\mathcal{K}}} \mathcal{N}_\rho \cup \{\{\mathcal{S}\},\{\mathcal{K}\}\} \rangle \\
&= \quad \{ \text{ sets; } \mathcal{S} \text{ and } \mathcal{K} \text{ are new nodes, \textit{i.e.}, not present in } \mathcal{N}_\rho \ \} \\
&\quad \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle \\
&= \quad \{ \text{ projections } \} \\
&\quad \rho \qquad \square
\end{aligned}
$$

3.3. Reconfiguration patterns

Through composition, one may express 'big step' reconfigurations able to affect significant parts of a connector (rather than just a node or a channel). These will be referred to as reconfiguration patterns. The word *pattern* is used to emphasise their generic nature and reusability.

Fig. 5 depicts a set of reconfiguration patterns, first introduced in [10], and found useful in practice. Each of them shall be read from left (*the original configuration*) to right (*what was obtained after applying the reconfiguration*). They are formally described and exemplified in the following paragraphs.

Remove. Pattern $\rho \bullet \mathsf{removeP}(Cs)$ removes a set of channel identifiers $Cs \subseteq \mathfrak{I}_\rho$ from $\rho \in \mathcal{P}$, by successively applying the primitive remove operation. Formally,

$$\rho \bullet \mathsf{removeP}(Cs) = rS(\rho, Cs)$$

where

$$
\begin{aligned}
rS(\rho, \emptyset) &= \rho \\
rS(\rho, \{c\} \cup C) &= rS(\rho \bullet \mathsf{remove}_c, C)
\end{aligned}
$$

Overlap. Pattern $\rho \bullet \mathsf{overlapP}(\rho_r, X)$ connects a new pattern $\rho_r \in \mathcal{P}$ to ρ by joining specific nodes in $X \subseteq \mathcal{N}_\rho \times \mathcal{N}_{\rho_r}$. Each pair in X indicates which nodes from ρ and ρ_r are to be joined. Formally,

$$\rho \bullet \mathsf{overlapP}(\rho_r, X) = rO(\rho \bullet \mathsf{par}_{\rho_r}, X)$$

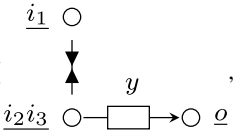
where

$$rO(\rho, \emptyset) = \rho$$
$$rO(\rho, \{y\} \cup Y) = rO(\rho \bullet \mathsf{join}_{\{\pi_1(y), \pi_2(y)\}}, Y)$$

Insert. Pattern $\rho \bullet \mathsf{insertP}(\rho_r, n, m_i, m_o)$ places $\rho, \rho_r \in \mathcal{P}$ side by side and splits $n \in \mathcal{N}_\rho$ to make room for ρ_r to be interset Connections are then re-built as follows: all the output ports produced by the split operation are joined with $m_i \in I(\rho_r)$. Dually, the input ports produced by the split operation are joined with $m_o \in O(\rho_r)$. Formally,

$$\begin{aligned} \rho \bullet \mathsf{insertP}(\rho_r, n, m_i, m_o) = \ & \text{let} \ \ \rho_1 = \rho \bullet \mathsf{par}_{\rho_r} \\ & \qquad \rho_2 = \rho_1 \bullet \mathsf{split}_n \\ & \qquad I_{sp} = I(\rho_2) \setminus I(\rho_1) \\ & \qquad O_{sp} = O(\rho_2) \setminus O(\rho_1) \\ & \text{in} \ \ (\rho_2 \bullet \mathsf{join}_{O_{sp} \cup \{m_i\}}) \bullet \mathsf{join}_{I_{sp} \cup \{m_o\}} \end{aligned}$$

Example 5. Consider ρ_s to be the *sequencer* coordination pattern depicted in Fig. 1. Suppose that a company uses this protocol to coordinate the sequential execution of two services connected to ports o_1 and o_2. For some reason there was a need to restrict the second service to execute only on completion of the first. A possible solution is to let the first service to acknowledge its termination and the protocol to memorise it. The following reconfiguration

$r_{proactive} = \rho_s \bullet \mathsf{insertP}($  $, \underline{fgh}, \underline{i_2 i_3}, \underline{o})$

does the job, yielding the *proactive waiting sequencer* coordination pattern depicted in Fig. 6.

Replace. Pattern $\rho \bullet \mathsf{replaceP}(\rho_r, X, Cs)$ replaces a sub-structure of $\rho \in \mathcal{P}$ by removing the old structure composed of the channels in set $Cs \subseteq \mathfrak{I}_\rho$ and overlapping ρ_r via information in a set $X \subseteq \mathcal{N}_\rho \times \mathcal{N}_{\rho_r}$. Formally,

$$\rho \bullet \mathsf{replaceP}(\rho_r, X, Cs) = (\rho \bullet \mathsf{removeP}(Cs)) \bullet \mathsf{overlapP}(\rho_r, X))$$

Example 6. Consider again the *sequencer* coordination pattern. Suppose that the services connected to ports o_1 and o_2 can fail for long periods of time, possibly leading to deadlocks. A possible solution is not to always enforce services to answer. Replacing the sync channels that provide ports o_1 and o_2 by lossy channels solves the problem, as encoded in the following reconfiguration:

$$r_{weak} = \rho_s \bullet \mathsf{replaceP}(\begin{matrix} \underline{i_1} \ \bigcirc \text{- - }^{s_5}\text{-> } \bigcirc \ \underline{o_1} \\ \underline{i_2} \ \bigcirc \text{- - }^{s_6}\text{-> } \bigcirc \ \underline{o_2} \end{matrix}, \{(\underline{cde}, \underline{i_1}), (\underline{fgh}, \underline{i_2})\}, \{s_2, s_3\})$$

which produces the *weak sequencer* depicted in Fig. 6, which actually avoids deadlock.

Implode. Pattern $\rho \bullet \mathsf{implodeP}(Cs)$ collapses a sub-structure of $\rho \in \mathcal{P}$ composed of the channels in $Cs \subseteq \mathfrak{I}_\rho$. The resulting ports are joined together into a new node. Formally,

$$\begin{aligned} \rho \bullet \mathsf{implodeP}(Cs) = \ & \text{let} \ \ \rho_1 = \rho \bullet \mathsf{removeP}(Cs) \\ & \text{in} \ \ \rho_1 \bullet \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho} \end{aligned}$$

Move. Pattern $\rho \bullet \mathsf{moveP}(ch, e, n)$ moves the end of a channel $ch \in \mathfrak{I}_\rho$ from node e to node $n \in \mathcal{N}_\rho$. Formally,

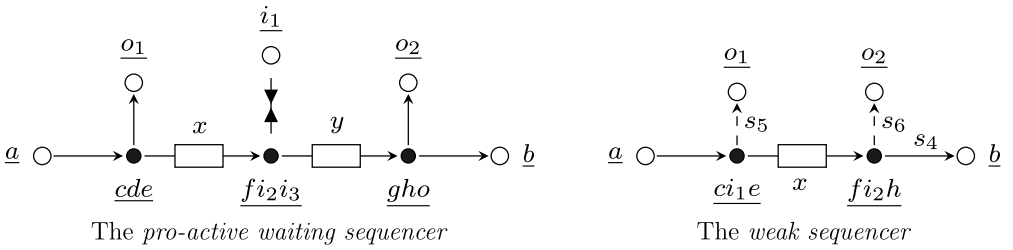


Fig. 6. Reconfigurations of the *sequencer* pattern.

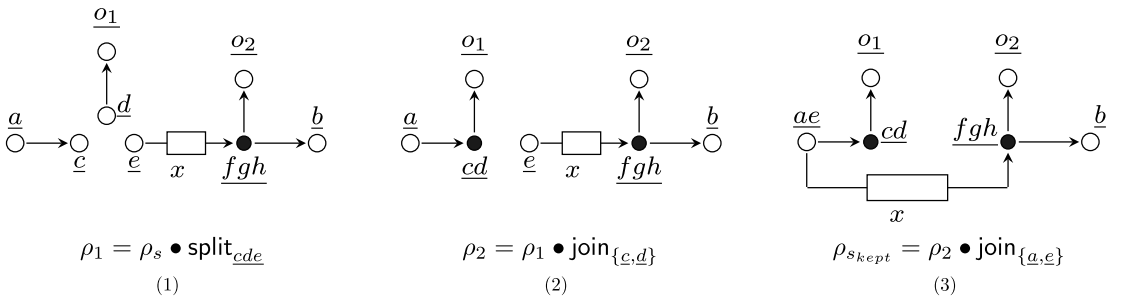


Fig. 7. Step-by-step example of moveP reconfiguration.

$$
\begin{aligned}
\rho \bullet \mathsf{moveP}(ch, e, n) = \ \mathsf{let}\ \ & e' = \mathfrak{N}^{ch}_{\rho} \setminus \{e\} \\
& \rho_1 = \rho \bullet \mathsf{split}_e \\
& E = \mathfrak{N}^{ch}_{\rho_1} \\
& I_{sp} = I(\rho_1) \setminus I(\rho) \\
& O_{sp} = O(\rho_1) \setminus O(\rho) \\
\mathsf{in}\ \ & (\rho_1 \bullet \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E}) \bullet \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}}
\end{aligned}
$$

Example 7. This example uses the sequencer coordination pattern in order to explain, step by step, the moveP reconfiguration pattern. Consider the reconfiguration $r_{s_{kept}} = \rho_s \bullet \mathsf{moveP}(x, \underline{cde}, \underline{a})$. that takes a channel identified by x with an end in node $\underline{cde}$ to be moved to node $\underline{a}$. The first step is to obtain the node where the other end of channel x is: $e' = \mathfrak{N}^{x}_{\rho_s} \setminus \{\underline{cde}\} = \underline{fgh}$. Then reconfiguration $\mathsf{split}_{\underline{cde}}$ is applied to the sequencer pattern, yielding ρ_1 as depicted in Fig. 7 (1). Now, the three sets of nodes E, I_{sp} and O_{sp} are computed:

$$
\begin{aligned}
E &= \mathfrak{N}^{x}_{\rho_1} = \{\underline{e}, \underline{fgh}\}; \\
I_{sp} &= I(\rho_1) \setminus I(\rho) = \{\underline{a}, \underline{e}, \underline{d}\} \setminus \{\underline{a}\} = \{\underline{e}, \underline{d}\}; \\
O_{sp} &= O(\rho_1) \setminus O(\rho) = \{\underline{c}, \underline{o_1}, \underline{o_2}, \underline{b}\} \setminus \{\underline{o_1}, \underline{o_2}, \underline{b}\} = \{\underline{c}\}.
\end{aligned}
$$

Finally, two join reconfigurations are applied with arguments $(I_{sp} \cup O_{sp}) \setminus E = \{\underline{c}, \underline{d}\}$ and $(E \setminus \{\underline{fgh}\}) \cup \{\underline{a}\} = \{\underline{a}, \underline{e}\}$, respectively. Their effects are depicted in Fig. 7 (2) and (3). The final coordination pattern is $\rho_{s_{kept}}$.

In spite of the apparently complex definition of some of these reconfiguration patterns, they all arise as sequential applications of primitive reconfigurations. Lemma 5 makes this observation precise.

Lemma 5. *Each reconfiguration pattern above arises as the product of a sequence of primitive reconfigurations.*

Proof.

- An inductive argument establishes the result for $\mathsf{removeP}(C)$:
  - Base case: $C = \emptyset$. By definition, $\rho \bullet \mathsf{removeP}(\emptyset) = rS(\rho, \emptyset) = \rho$.
  - Inductive case: $C \neq \emptyset$. Let $C = \{c\} \cup C'$. Assume that $rS(\rho, C') = \rho \bullet r$, where r is a sequence of remove reconfiguration primitives. By definition, $\rho \bullet \mathsf{removeP}(C) = rS(\rho, C) = rS(\rho \bullet \mathsf{remove}_c, \{c\} \cup C')$. Using the hypothesis we obtain $(\rho \bullet \mathsf{remove}_c) \bullet r$, which is equal to $\rho \bullet \{\mathsf{remove}_c \ ; \ r\}$. Therefore, the pattern may be expressed as a sequence of remove primitives.

- For pattern $\mathsf{overlapP}(\rho_r, X)$ the proof is also by induction (over a set X):
  - Base case: $X = \emptyset$. By definition, $\mathsf{overlapP}(\rho_r, \emptyset) = rO(\rho \bullet \mathsf{par}_{\rho_r}, \emptyset) = \rho \bullet \mathsf{par}_{\rho_r}$
  - Inductive case: $X \neq \emptyset$. Let $X = \{x\} \cup X'$. Assume, as induction hypothesis, that $rO(\rho, X') = \rho \bullet r$, where r is a sequence of reconfiguration primitives. By definition, $\rho \bullet \mathsf{overlapP}(\rho_r, X) = rO(\rho \bullet \mathsf{par}_{\rho_r}, \{x\} \cup X')$, and yet, it is equivalent to $rO(((\rho \bullet \mathsf{par}_{\rho_r}) \bullet \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}}), X')$. Using the hypothesis we get $((\rho \bullet \mathsf{par}_{\rho_r}) \bullet \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}}) \bullet r$. This is equivalent to

$$\rho \bullet \{\mathsf{par}_{\rho_r} \ ; \ \mathsf{join}_{\{\pi_1(x),\pi_2(x)\}} \ ; \ r\}$$

    which encodes the overlapP pattern as a sequence of reconfiguration primitives.
- For pattern $\mathsf{insertP}(\rho_r, n, m_i, m_o)$ the result comes directly from the definition. Let $I_{sp} = I((\rho \bullet \mathsf{par}_{\rho_r}) \bullet \mathsf{split}_n) \setminus I(\rho \bullet \mathsf{par}_{\rho_r})$ and $O_{sp} = O((\rho \bullet \mathsf{par}_{\rho_r}) \bullet \mathsf{split}_n) \setminus O(\rho \bullet \mathsf{par}_{\rho_r})$. Then,

$$\begin{aligned}
&\rho \bullet \mathsf{insertP}(\rho_r, n, m_i, m_o) \\
&= (((\rho \bullet \mathsf{par}_{\rho_r}) \bullet \mathsf{split}_n) \bullet \mathsf{join}_{O_{sp} \cup \{m_i\}}) \bullet \mathsf{join}_{I_{sp} \cup \{m_o\}}) \\
&= \rho \bullet \{\mathsf{par}_{\rho_r} \ ; \ \mathsf{split}_n \ ; \ \mathsf{join}_{O_{sp} \cup \{m_i\}} \ ; \ \mathsf{join}_{I_{sp} \cup \{m_o\}}\}
\end{aligned}$$

- The proof for $\mathsf{replaceP}(\rho_r, X, C)$ is also a consequence of its definition and the fact that this Lemma holds for both removeP and overlapP. Therefore, assuming $r_1 = \mathsf{removeP}(C)$ and $r_2 = \mathsf{overlapP}(\rho_r, X)$, with r_1 and r_2 sequences of primitive reconfigurations, one gets

$$\begin{aligned}
&\rho \bullet \mathsf{replaceP}(\rho_r, X, C) \\
&= (\rho \bullet \mathsf{removeP}(C)) \bullet \mathsf{overlapP}(\rho, X) \\
&= (\rho \bullet r_1) \bullet r_2 \\
&= \rho \bullet \{r_1 \ ; \ r_2\}
\end{aligned}$$

- For $\mathsf{implodeP}(C)$ the proof is similar to the previous one. Assume that $r = \mathsf{removeP}(C)$, where r is a sequence of reconfiguration primitives, because this lemma holds for removeP. Let $\rho_1 = \rho \bullet \mathsf{removeP}(C)$ Therefore,

$$\begin{aligned}
&\rho \bullet \mathsf{implodeP}(C) \\
&= (\rho \bullet \mathsf{removeP}(C)) \bullet \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho} \\
&= (\rho \bullet r) \bullet \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho} \\
&= \rho \bullet \{r \ ; \ \mathsf{join}_{\mathcal{N}_{\rho_1} \setminus \mathcal{N}_\rho}\}
\end{aligned}$$

- Finally, for $\mathsf{moveP}(ch, e, n)$ let $e' = \mathfrak{N}^{ch}_{\rho} \setminus \{e\}$, $\rho_1 = \rho \bullet \mathsf{split}_e$; $I_{sp} = I(\rho_1) \setminus I(\rho))$; $O_{sp} = O(\rho_1) \setminus O(\rho))$ and $E = \mathfrak{N}^{ch}_{\rho_1}$. Then,

$$\begin{aligned}
&\rho \bullet \mathsf{moveP}(ch, e, n) \\
&= (\rho_1 \bullet \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E}) \bullet \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}} \\
&= \rho \bullet \{\mathsf{split}_e \ ; \ \mathsf{join}_{(I_{sp} \cup O_{sp}) \setminus E} \ ; \ \mathsf{join}_{(E \setminus \{e'\}) \cup \{n\}}\}. \qquad \square
\end{aligned}$$

4. Reasoning about reconfigurations: behaviour

The following two sections investigate criteria to order reconfigurations and enable the working software architect to choose among them. The objective is to provide means to rule out configurations that, *e.g.,* fail to preserve the overall interconnection behaviour or introduce channels which are, for some reason, not immediately available. This section introduces a *behaviour-based* criterium for reasoning about reconfigurations. A complementary *structural* criterium is discussed later in Section 5.

4.1. Semantic models

The behavioural point of view requires fixing a concrete semantic model for the coordination pattern. For illustration purposes, constraint automata and Reo automata, two well-known models for Reo already discussed in Section 2, will be considered here. Comparing reconfigurations in terms of the behaviour enforced on the coordination patterns to which they are applied, entails the need for suitable notions of similarity and bisimilarity. Fortunately, they are already available in the literature.

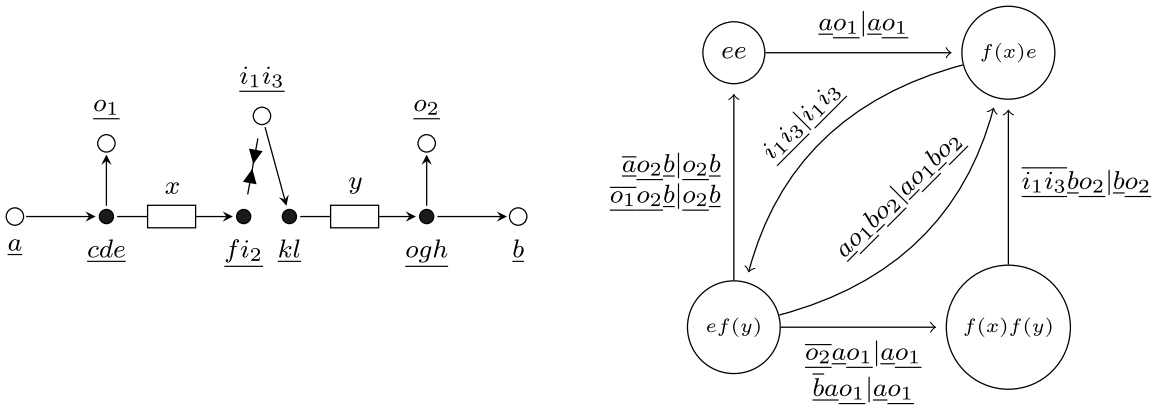


Fig. 8. The *pro-active dependent sequencer* patterns and corresponding Reo automata.

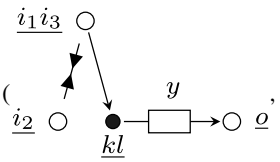
4.2. Comparing reconfigurations

Let $[\![\rho]\!]_{\mathfrak{M}}$ stand for the semantics of the coordination pattern $\rho \in \mathcal{P}$ in the semantic model $\mathfrak{M}$, and $\sim$ and $\preceq$ the bisimilarity and similarity relations in $\mathfrak{M}$, respectively. As before consider $\mathfrak{M} \in \{\mathsf{CA}, \mathsf{RA}\}$, with CA standing for constraint automata, and RA for Reo automata. Reference to $\mathfrak{M}$ can be omitted when the model is clear from the context. Reconfigurations can be compared with respect to their application to a specific coordination pattern. Thus,

Definition 11. Let $\rho \in \mathcal{P}$, r_1, r_2 be reconfigurations and $\mathfrak{M}$ a semantic model. Then,

$$(r_1 \stackrel{\circ}{=}_{\mathfrak{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \sim [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$
$$(r_1 \preccurlyeq_{\mathfrak{M}} r_2)_\rho \quad \text{iff} \quad [\![\rho \bullet r_1]\!]_{\mathfrak{M}} \preceq [\![\rho \bullet r_2]\!]_{\mathfrak{M}}$$

Example 8. Consider again the *sequencer* coordination pattern. Suppose that a new requirement enforces a strict dependence between services in a row. In particular, suppose the second service (connected to port o_2) is launched with the output of the first service, and that such an output is memorised whenever the second service is not ready to consume it. Reconfiguration

$$r_{dependent} = \mathsf{insertP}($$  $$, \underline{fgh}, \underline{i_2}, \underline{o})$$

which is akin to $r_{proactive}$ (see Example 5), meets the envisaged requirement. Fig. 8 presents the resulting pattern, which will be referred to as the *pro-active dependent sequencer*. The corresponding Reo automata is exactly the same of the coordination pattern obtained through application of the $r_{proactive}$ reconfiguration. So, although both patterns are slightly different, they exhibit a bisimilar behaviour (up to port names), expressed in the Reo automata model.

Therefore,

$$(r_{proactive} \stackrel{\circ}{=}_{RA} r_{dependent})_{\rho_s}$$

Using constraint automata as a semantic model, however, bisimilarity fails. Fig. 9 shows the constraint automata for each case. The difference (highlighted in the dashed transition) is that on the *pro-active sequencer*, data on the buffer y comes from buffer x; while in the *pro-active dependant sequencer* data on buffer y comes from node $\underline{i_1 i_3}$.

Example 9. Consider again the *sequencer* coordination pattern. Suppose that the results delivered by the second service are an alternative to those offered by the first. Therefore, whenever this fails, the system may proceed normally through port b, disregarding port o_2. This requirement is met by applying the following reconfiguration:

$$r_{quasiweak} = \rho_s \bullet \mathsf{replaceP}(\underline{i} \bigcirc\!\!- - - \!\!\rightarrow\!\bigcirc\ \underline{o_2}, \{(\underline{fgh}, \underline{i})\}, \{s_3\})$$

which is actually part of the reconfiguration r_{weak} discussed in Example 6. The resulting coordination pattern (referred to as a *quasi-weak sequencer*) is a variant of the *weak sequencer*. Their structure and behaviour model are depicted in Fig. 10.

In spite of their similarity, it is possible to see that the upper transition of the automata for the *weak sequencer* has an extra guard expressing that data may be lost on the first lossy channel. So that, for $\mathfrak{M} \in \{\mathsf{CA}, \mathsf{RA}\}$, $[\![\textit{quasi-weak sequencer}]\!]_{\mathfrak{M}} \preceq [\![\textit{weak sequencer}]\!]_{\mathfrak{M}}$. Thus, $(r_{quasiweak} \preccurlyeq r_{weak})_{\rho_s}$.

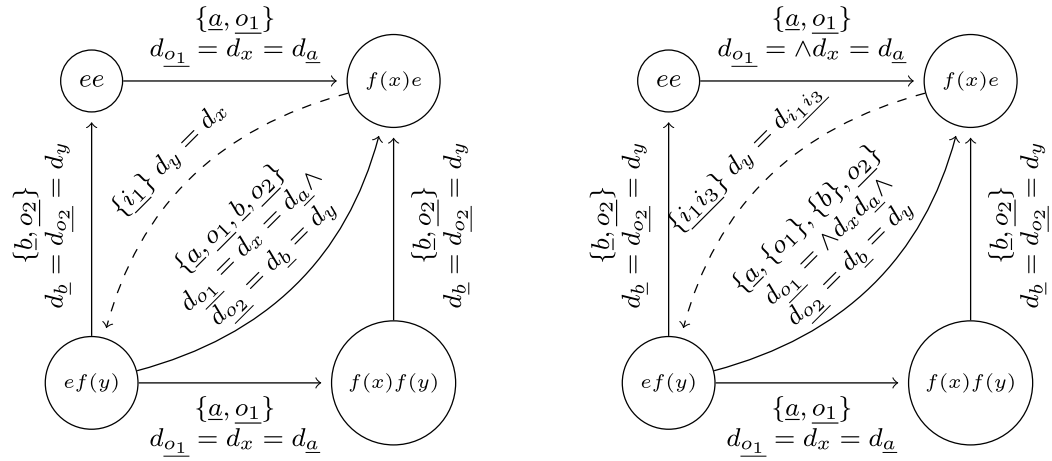


Fig. 9. CA semantics for the *pro-active sequencer* (left) and the *pro-active dependent sequencer* (right).

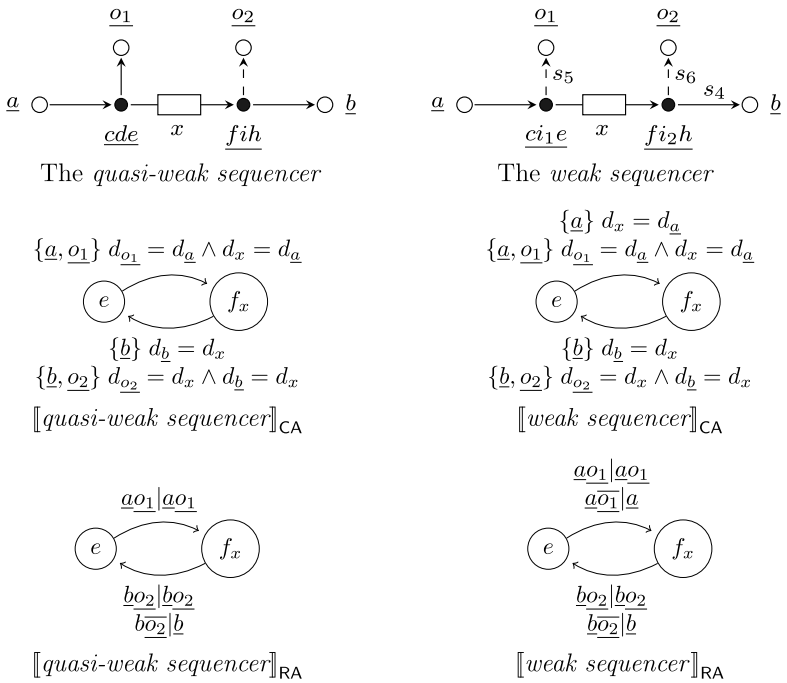


Fig. 10. The *quasi weak* and the *weak sequencer* semantics.

This suggests a possible classification of reconfigurations w.r.t. a coordination pattern and a semantic model as follows:

Definition 12. Let $\rho \in \mathcal{P}$, r a reconfiguration, and $\mathfrak{M}$ a semantic model. Then,

- r is *unobtrusive* iff $(\mathbf{1}_r \stackrel{\circ}{=}_{\mathfrak{M}} r)_\rho$ (original behaviour preserved),
- r is *expansive* iff $(\mathbf{1}_r \preccurlyeq_{\mathfrak{M}} r)_\rho$ (new behaviour added),
- r is *contractive* iff $(r \preccurlyeq_{\mathfrak{M}} \mathbf{1}_r)_\rho$ (original behaviour partially removed),
- r is *disruptive* otherwise,

where $\mathbf{1}_r$ denotes the identity reconfiguration, which, as expected, when applied to a coordination pattern keeps it unchanged.

Example 10. It is now possible to classify all the reconfigurations of the *sequencer* pattern mentioned above w.r.t. either the constraint automata or Reo automata semantic models. The $r_{proactive}$ and the $r_{dependent}$ reconfigurations are *disruptive*

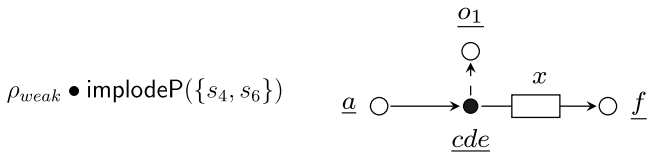


Fig. 11. Contractive reconfiguration.

while the r_{weak} and the $r_{quasiweak}$ are *expansive*. Moreover, $r_{s_{kept}}$ is an *unobtrusive* reconfiguration. Fig. 11 shows a *contractive* reconfiguration w.r.t., and up to node renaming, the *weak sequencer* coordination pattern.

5. Reasoning about reconfigurations: structure

As discussed in the previous section, connectors exhibit behaviour, determined by the underlying semantic model, and therefore the effect of a reconfiguration can be 'measured' by whatever behavioural changes are entailed by its application. For example, in certain cases, it may be necessary to rule out reconfigurations leading to non-bisimilar behaviours.

There is, however, another perspective whose focus is placed on the interconnection structure, with no reference to the emerging behaviour. Examples of *structural*, or 'syntactic' properties are:

i) *every* $\mathsf{fifo_e}$ *channel from a node n is connected to at least a* lossy *channel* or
ii) *node i is a connector's output node.*

One may then require that a reconfiguration preserves such properties. This will lead to a different family of relations to compare reconfigurations. What should be remarked is the fact that such relations are independent of the underlying semantic model and, in a broader sense, not committed to the use of a specific coordination modelling language. This section introduces a hybrid logic to express and reason about them.

5.1. A hybrid logic

Modal logic provides the standard way of expressing properties over the graph-like structure underlying coordination patterns. Actually, any coordination pattern ρ gives rise to a transition system $\mathcal{G}(\rho)$ over nodes in $\mathcal{N}$ and labelled by connector types in $\mathcal{T}$, such that $m \xrightarrow{t} n$ if *data may flow* from node m to node n through a connector of type t in ρ. Formally,

Definition 13. Given a coordination pattern $\rho = \langle \mathcal{C}_\rho, \mathcal{N}_\rho \rangle$, its *may-flow graph* $\mathcal{G}(\rho)$ is a labelled transition structure over $\mathcal{N}_\rho$, labelled by channel types, and given by

$$\bigcup_{\langle \mathcal{S},i,t,\mathcal{K}\rangle \in \mathcal{C}_\rho,\ \mathcal{S},\mathcal{K}\neq\emptyset} \{\langle m,t,n\rangle \mid m,n \in \mathcal{N}_\rho \ \wedge\ m \cap \mathcal{S} \neq \emptyset \ \wedge\ n \cap \mathcal{K} \neq \emptyset\} \tag{12}$$

The set of nodes in $\mathcal{G}(\rho)$ is denoted by $\mathcal{G}_{nodes}(\rho)$.

Clearly, $\mathcal{G}_{nodes}(\rho) \subseteq \mathcal{N}_\rho$. A similar transition could be defined labelled by connector identifiers in $\mathcal{I}$ or even by pairs in $\mathcal{T} \times \mathcal{I}$, both cases giving rise to deterministic transition systems. The logic below is, of course, independent of whatever labels are chosen for representing specific views of the connector structure.

Often, however, structural properties are to be formulated relatively to a particular node in the pattern. An example is given by property *ii*) above. In general, one may require, for instance, that all the channels incident in a specific node and their interconnections remain unchanged under a reconfiguration. This justifies the choice of *hybrid* logic [20] to express such properties.

In general, hybrid logic adds to a modal language the ability to name, or to explicitly refer to specific states of the underlying Kripke structure. This is done through the introduction of propositional symbols of a new sort, called nominals, each of which is true at exactly one possible state. The sentences are then enriched in two directions. On the one hand, nominals are used as simple sentences holding exclusively in the state they name. On the other hand, explicit reference to states is provided by a *satisfaction* operator @ such that $@_i\phi$ asserts the validity of ϕ at the state named i.

Structural properties of coordination patterns will be expressed in a variant of hybrid propositional logic, called $Hp\mathcal{E}$, where modalities are indexed by channel *types* in $\mathcal{T}$, or, more generally, by *subsets* of $\mathcal{T}$. The logic is interpreted over $\mathcal{G}(\rho)$, for each coordination pattern ρ. Each node n in $\mathcal{G}_{nodes}(\rho)$ is endowed with a model of a propositional logic $p\mathcal{E}$ defined over channel ends in $\mathcal{E}$ and the usual Boolean connectives:

$$p, p' \ni e \mid \neg p \mid p \wedge p'$$

where $e \in \mathcal{E}$. The satisfaction relation is as follows

$$
\begin{array}{lll}
n \models^{p\mathcal{E}} e & \text{iff} & e \in n \\
n \models^{p\mathcal{E}} \neg p & \text{iff} & n \not\models^{p\mathcal{E}} p \\
n \models^{p\mathcal{E}} p \wedge p' & \text{iff} & n \models^{p\mathcal{E}} p \wedge n \models^{p\mathcal{E}} p'
\end{array}
$$

On top of $p\mathcal{E}$ a hybrid language is defined as follows:

$$\phi, \phi' \ni p \mid i \mid \neg\phi \mid \phi \wedge \phi' \mid [K]\phi \mid [\![K]\!]\phi \mid @_i\phi$$

where p is a sentence of $p\mathcal{E}$, $K \subseteq \mathcal{T}$, and $i \in \mathsf{Nom}$ for a set Nom of nominals. Constants true and false, as well as the usual Boolean connectives, are defined by abbreviation. For simplicity, in modalities '$-$' stands for the whole set $\mathcal{T}$, and $[-t]$, with $t \in \mathcal{T}$, stands for $\mathcal{T} \setminus \{t\}$.

Modality $[K]$ quantifies universally over the edges of $\mathcal{G}(\rho)$ labelled by channel types in K; its dual $\langle K\rangle = \neg[K]\neg$ provides an existential quantification. Modalities $\langle K\rangle$ and $[K]$ express properties of *outgoing* connections from the node in which they are evaluated, in a coordination pattern. Dually, modalities $\langle\langle K\rangle\rangle$ and $[\![K]\!]$ express properties of *incoming* connections. Finally, the satisfaction operator @ *redirects* the formula evaluation to the context of a specific node. Nominals make possible to express proprieties *local* to a specific node.

A semantic model, $\mathcal{M}$, for this language is a pair $\langle \mathcal{G}(\rho), \sigma : \mathsf{Nom} \longrightarrow \mathcal{N}\rangle$, where ρ is a coordination pattern, and σ assigns to each nominal a node in $\mathcal{G}(\rho)$. The satisfaction relation, given a model $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma\rangle$ and a node n is defined as follows:

$$
\begin{array}{lll}
\mathcal{M}, n \models p & \text{iff} & n \models^{p\mathcal{E}} p \\
\mathcal{M}, n \models \neg\phi & \text{iff} & \mathcal{M}, n \not\models \phi \\
\mathcal{M}, n \models \phi \wedge \phi' & \text{iff} & \mathcal{M}, n \models \phi \text{ and } \mathcal{M}, n \models \phi' \\
\mathcal{M}, n \models i & \text{iff} & \sigma(i) = n \\
\mathcal{M}, n \models @_i\phi & \text{iff} & \mathcal{M}, \sigma(i) \models \phi \\
\mathcal{M}, n \models [K]\phi & \text{iff} & \forall_{m \in \{p \mid \langle n,t,p\rangle \in \mathcal{G}(\rho) \,\wedge\, t \in K\}} \,.\, \mathcal{M}, m \models \phi \\
\mathcal{M}, n \models [\![K]\!]\phi & \text{iff} & \forall_{m \in \{p \mid \langle p,t,n\rangle \in \mathcal{G}(\rho) \,\wedge\, t \in K\}} \,.\, \mathcal{M}, m \models \phi
\end{array}
$$

The satisfaction relation $\models$ lifts, as usual, to (global) satisfiability by quantifying over all the nodes in the model. *I.e.*, ϕ is *globally satisfied* in $\mathcal{M}$ ($\mathcal{M} \models \phi$) if it is satisfied at all nodes in $\mathcal{M}$.

Example 11. Consider the following properties:

- Property *i*) above, is expressed as

  $@_n[\mathsf{fifo_e}]\langle\mathsf{lossy}\rangle\mathsf{true}$.

- Property *ii*), expressed as

  $@_i[-]\mathsf{false}$,

  uses $[-]$false to state the absence of outgoing channels from the node referred by nominal i.
- All outgoing channels from i are lossy:

  $@_i(\langle-\rangle\mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false})$.

- Absence of a loop formed by a sync followed by a lossy channel at i:

  $i \rightarrow \neg\langle\mathsf{sync}\rangle\langle\mathsf{lossy}\rangle i$.

  Notice that the absence of loops, and in general, irreflexivity of a binary relation is not expressible in classical modal logic [21].
- All output nodes are accessible through a sync channel but never through a $\mathsf{fifo_e}$ channel:

  $[-]\mathsf{false} \rightarrow (\langle\langle\mathsf{sync}\rangle\rangle\,\mathsf{true} \wedge [\![\mathsf{fifo_e}]\!]\mathsf{false})$

Properties expressed in $Hp\mathcal{E}$ can be verified, after a translation to classical propositional hybrid logic in the HyloRes [22] system. The translation involves a restriction of propositional symbols to nominals and an encoding of the backward modalities $\langle\langle K\rangle\rangle$ and $[\![K]\!]$.

5.2. Bisimulation for $Hp\mathcal{E}$

The notion of bisimulation for $Hp\mathcal{E}$-models provides the right tool to compare coordination patterns from a *structural* point of view. Formally, let $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma\rangle$ and $\mathcal{M}' = \langle \mathcal{G}(\rho'), \sigma'\rangle$, defined over the same set Nom of nominals and, of course, the same set of channel ends $\mathcal{E}$. Then,

Definition 14. A *bisimulation* for $Hp\mathcal{E}$-models is a binary relation $R \subseteq \mathcal{G}_{nodes}(\rho) \times \mathcal{G}_{nodes}(\rho')$ such that

i) for any $i \in \mathsf{Nom}$ and nRn', $\sigma(i) = n$ iff $\sigma'(i) = n'$
ii) for any $i \in \mathsf{Nom}$, $\sigma(i) R \sigma'(i)$
iii) if nRn', nodes n and n' are elementary equivalent, *i.e.,*

$$\forall_{p \in p\mathcal{E}}\ n \models^{p\mathcal{E}} p \text{ iff } n' \models^{p\mathcal{E}} p$$

iv) (zig) for any $t \in \mathcal{T}$, if nRn' and $\langle n, t, m\rangle \in \mathcal{G}(\rho)$, then there exists a node m' such that $\langle n', t, m'\rangle \in \mathcal{G}(\rho')$ and mRm'
v) (zag) for any $t \in \mathcal{T}$, if nRn' and $\langle n', t, m'\rangle \in \mathcal{G}(\rho)$, then there exists a node m such that $\langle n, t, m\rangle \in \mathcal{G}(\rho)$ and mRm'

Lemma 6. *The union and the relational composition of two bisimulations are still bisimulations.*

Proof. *Union.* Let $R = R_1 \cup R_2$, where both R_1, R_2 are bisimulations. Clearly, any two nodes n, n' related by R also satisfy nR_1n' or nR_2n'. Therefore, clauses i), ii) and iii) of Definition 14 hold. For clause iv) consider a connection $\langle n, c, m\rangle$. If nR_1n' (respectively, nR_2n') there is a node m' such that $\langle n', c, m'\rangle$ and mR_1m' (respectively, mR_2m'), which, in either case, entails mRm'. Clause v) is proved similarly.
Composition. Consider two (composable) bisimulations R_1, R_2 and suppose $nR_2 \cdot R_1n'$, for nodes n, n'. Clauses i), ii) and iii) of Definition 14 hold for $R_2 \cdot R_1$ because they also hold individually for R_1 and R_2; for clause iii) note that the elementary equivalence used in its statement is an equivalence relation. For clause iv), suppose $nR_2 \cdot R_1n'$, for nodes n, n'. By definition of relational composition, there is a node p such that nR_1p and pR_2n'. Suppose there is a connection $\langle n, c, m\rangle$; then, R_1 being a bisimulation, there exists a node p' and a connection $\langle p, c, p'\rangle$, with mR_1p'. Connection $\langle p, c, p'\rangle$ and the fact that pR_2n' for R_2 a bisimulation, on the other hand, entails the existence of a node m' such that there is a connection $\langle n', c, m'\rangle$ and $p'R_2m'$. Thus $mR_2 \cdot R_1m'$, as wanted. Clause v) is proved similarly. □

Definition 15. Given two models, $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma\rangle$ and $\mathcal{M}' = \langle \mathcal{G}(\rho'), \sigma'\rangle$, and two nodes n, n' in $\mathcal{G}_{nodes}(\rho)$ and $\mathcal{G}_{nodes}(\rho')$, respectively, n, n' are *bisimilar*, denoted by $n \rightleftharpoons n'$, iff there is a relation R which is a bisimulation over $\mathcal{M}, \mathcal{M}'$ and also over $\mathcal{M}^\circ, \mathcal{M}'^\circ$, such that nRn', where $\mathcal{M}^\circ$ is a model identical to $\mathcal{M}$ but defined over the relational converse of $\mathcal{G}(\rho)$.

This extra requirement for R in the definition above comes from the presence of a backwards modality $[\![K]\!]$ to reason about incoming connections. Clearly,

Lemma 7. *Let $\mathcal{U}$ be a set of coordination patterns. Relation $\rightleftharpoons$ is an equivalence relation over the set of nodes of all patterns in $\mathcal{U}$.*

Proof. It is immediate to show that the identity relation is a bisimulation, and so is the relational converse of a bisimulation (if $n \rightleftharpoons n'$ is witnessed by a bisimulation $R \subseteq \mathcal{G}_{nodes}(\rho) \times \mathcal{G}_{nodes}(\rho')$, for $\rho, \rho' \in \mathcal{U}$, then $n' \rightleftharpoons n$ is witnessed by $R^\circ \subseteq \mathcal{G}_{nodes}(\rho') \times \mathcal{G}_{nodes}(\rho)$). Transitivity, on the other hand, comes from the fact that the relational composition of bisimulations is also a bisimulation, as proved in the second part of Lemma 6. □

The usual connection between bisimilarity and modal satisfaction also holds as shown in the sequel.

Lemma 8. *Given two models, $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma\rangle$ and $\mathcal{M}' = \langle \mathcal{G}(\rho'), \sigma'\rangle$, and two nodes n, n' in $\mathcal{G}_{nodes}(\rho)$ and $\mathcal{G}_{nodes}(\rho')$, such that $n \rightleftharpoons n'$, then*

$$\mathcal{M}, n \models \Phi \iff \mathcal{M}', n' \models \Phi$$

for every formula $\Phi \in Hp\mathcal{E}$.

Proof. The proof proceeds by induction on the structure of formulas.

i) $\Phi = p$. Let $\mathcal{M}, n \models p$ which, by definition of $\models$, is equivalent to $n \models^{p\mathcal{E}} p$. Nodes n and n' are elementary equivalent because $n \rightleftharpoons n'$, which entails $n' \models^{p\mathcal{E}} p$. Therefore, $\mathcal{M}', n' \models p$.
ii) $\Phi = i$. Let $\mathcal{M}, n \models i$ which, by definition of $\models$, is equivalent to $\sigma(i) = n$. Then $\sigma'(i) = n'$ because $n \rightleftharpoons n'$, which entails $\mathcal{M}', n' \models i$.
iii) $\Phi = @_i\phi$. Let $\mathcal{M}, n \models @_i\phi$ which, by definition of $\models$, is equivalent to $\mathcal{M}, \sigma(i) \models \phi$. By definition of bisimulation $\sigma(j) \rightleftharpoons \sigma'(j)$, for any nominal j. This combined with the induction hypothesis yields $\mathcal{M}', \sigma'(i) \models \phi$. Therefore, $\mathcal{M}', n' \models @_i\phi$.
iv) $\Phi = [K]\phi$. Let $\mathcal{M}, n \models [K]\phi$ which, by definition of $\models$, is equivalent to $\forall_{m \in \{p \mid \langle n, c, p\rangle \in \mathcal{G}(\rho) \wedge c \in K\}} \cdot \mathcal{M}, m \models \phi$. Assume $n \rightleftharpoons n'$ is witnessed by a relation R, and for each m above. For each connection $\langle n', c, m'\rangle$ in $\mathcal{M}'$, there exists, by clause v) in Definition 14, a node $m \in \mathcal{M}$ such that $\langle n, c, m\rangle$ in $\mathcal{M}$ and $m \rightleftharpoons m'$. As $\mathcal{M}, m \models \phi$, by induction hypothesis, $\mathcal{M}', m' \models \phi$. Therefore, $\forall_{m' \in \{p' \mid \langle n', c, p'\rangle \in \mathcal{G}(\rho) \wedge c \in K\}} \cdot \mathcal{M}', m' \models \phi$ which entails $\mathcal{M}', n' \models [K]\phi$. Clause iv) in Definition 14 gives the converse implication.

v) $\Phi = [\![K]\!]\phi$. The argument is similar to the one used in iv) with R being a bisimulation for $\mathcal{M}^\circ, \mathcal{M}'^\circ$.

vi) $\Phi = \phi \wedge \psi$. Let $\mathcal{M}, n \models \phi \wedge \psi$ which, by definition of $\models$, is equivalent to $\mathcal{M}, n \models \phi$ and $\mathcal{M}, n \models \psi$. By induction hypothesis $\mathcal{M}', n' \models \phi$ and $\mathcal{M}', n' \models \psi$, which entails $\mathcal{M}', n' \models \phi \wedge \psi$. The argument is similar for $\Phi = \neg\phi$ (and, in general, for the derived Boolean connectives). □

The converse of this result also holds whenever $\mathcal{G}(\rho)$ is image finite. Being image finite means that the number of incident connections in a node, for all nodes in $\mathcal{G}(\rho)$, is finite. Note, however, this is always the case, for any ρ, as coordination patterns are typically composed of a finite number of connectors.

Lemma 9. *Consider two models, $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma \rangle$ and $\mathcal{M}' = \langle \mathcal{G}(\rho'), \sigma' \rangle$, and two nodes n, n' in $\mathcal{G}_{nodes}(\rho)$ and $\mathcal{G}_{nodes}(\rho')$, respectively, such that $\mathcal{M}, n \models \Phi \iff \mathcal{M}', n' \models \Phi$, for every formula $\Phi \in Hp\mathcal{E}$. Then $n \rightleftharpoons n'$.*

Proof. Let us show that

$$Z = \{(n, n') \in \mathcal{G}_{nodes}(\rho) \times \mathcal{G}_{nodes}(\rho') \mid \forall_{\phi \in Hp\mathcal{E}} \,.\, \mathcal{M}, n \models \phi \Leftrightarrow \mathcal{M}', n' \models \phi\}$$

is a bisimulation over $\mathcal{M}, \mathcal{M}'$ and $\mathcal{M}^\circ, \mathcal{M}'^\circ$. The atomic conditions, in clauses i), ii) and iii) of Definition 14 trivially hold. Consider, now clause iv) (the *(zig)* condition). Let nZn' and assume there is a connection $\langle n, c, m \rangle$.

Both $\mathcal{G}(\rho)$ and $\mathcal{G}(\rho')$ are, by construction, image finite, which makes $S = \{p' \mid \langle n', c, p' \rangle\}$ finite. It cannot be empty, however, because in such a case $M', n' \models \neg\langle c \rangle \mathsf{true}$ and, by definition of Z, $M, n \models \neg\langle c \rangle \mathsf{true}$ which is inconsistent with the existence of connection $\langle n, c, m \rangle$ assumed above.

To obtain a contradiction, suppose that there is no node $m' \in \mathcal{G}_{nodes}(\rho')$ such that mZm' and is part of a connection $\langle n', c, m' \rangle$. Therefore, for every $v \in S$, there is a formula ψ_v such that $\mathcal{M}, m \models \psi_v$ and $\mathcal{M}', v \not\models \psi_v$. Consider now the conjunction

$$\psi = \bigwedge_{v \in S} \psi_v$$

of all of these formulas. Then, on the one hand, $\mathcal{M}, n \models \langle c \rangle \psi$, but, on the other, $\mathcal{M}', n' \not\models \langle c \rangle \psi$. This, however, contradicts nZn'. The *(zag)* condition, clause v) in Definition 14 is shown in a similar way.

This proves relation Z is a bisimulation over $\mathcal{G}(\rho)$ and $\mathcal{G}(\rho')$. A similar argument shows it is also a bisimulation for their relational converses, which are also image finite. □

Combining both lemmas entails a Hennessy–Milner like result: modal equivalence in $Hp\mathcal{E}$ and bisimilarity coincide for models constructed from coordination patterns according to Definition 13. Note the result is valid in general for image finite models.

Theorem 1. *Consider two models $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma \rangle$ and $\mathcal{M}' = \langle \mathcal{G}(\rho'), \sigma' \rangle$, and two nodes n, n' in $\mathcal{G}_{nodes}(\rho)$ and $\mathcal{G}_{nodes}(\rho')$, respectively. Then,*

$$n \rightleftharpoons n' \text{ iff } \forall_{\Phi \text{ in } Hp\mathcal{E}}\ \mathcal{M}, n \models \Phi \iff \mathcal{M}', n' \models \Phi$$

Proof. Corollary of Lemmas 8 and 9 above. □

5.3. Expressing 'long scope' properties

$Hp\mathcal{E}$ can be extended to allow for modalities which take the form of regular expressions over pairs of channel *names* and *types*. Technically this is a particular case of the extension of a modal logic with fixed points as in the μ-calculus [23]. This is particularly useful to express 'long scope' properties, such as

i) *A channel of type* t *is accessible from a node referred to by* i
ii) *All input ports lead to an output port via, at least, one* $\mathsf{fifo_e}$ *channel*

Consider, thus, the following syntax for modalities:

$$\nu \ni \epsilon \mid t \mid \nu.\nu \mid \nu + \nu \mid \nu^* \mid \nu^+$$

where ϵ is the *empty word*, $t \in \mathcal{T}$, '.' denotes concatenation, '+' choice and ν^* and ν^+ the Kleene and transitive closures. The intuitive semantics is given below (a precise rendering requires the introduction of fixed points). Note that the modalities referring to *incoming* connections are defined dually.

Definition 16. Let ν, ν_1 and ν_2 be modalities and ϕ a formula. The following define the semantics of the modalities extension:

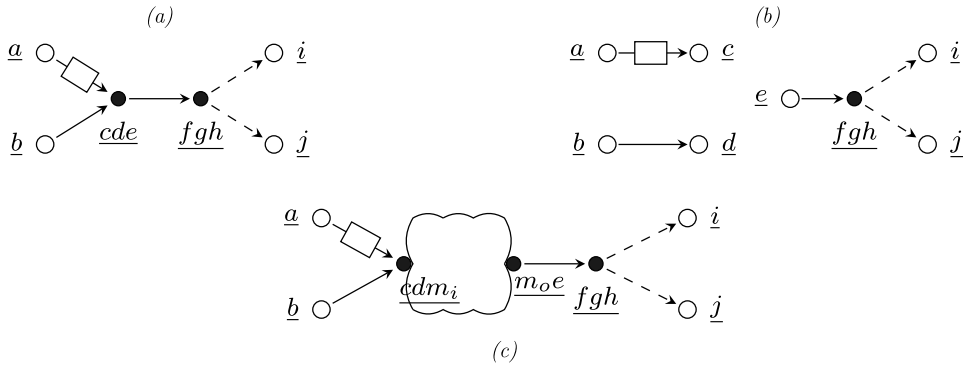


Fig. 12. Example of a *displaced* invariant.

- $\langle\epsilon\rangle\phi = \phi = [\epsilon]\phi$
- $[\nu_1.\nu_2]\phi = [\nu_1][\nu_2]\phi$
- $[\nu_1 + \nu_2]\phi = [\nu_1]\phi \vee [\nu_2]\phi$
- $[\nu^*]\phi = \phi \vee [\nu]\phi \vee [\nu][\nu]\phi \vee \ldots$
- $[\nu^+]\phi = [\nu]\phi \vee [\nu][\nu]\phi \vee \ldots$
- $\langle\nu_1.\nu_2\rangle\phi = \langle\nu_1\rangle\langle\nu_2\rangle\phi$
- $\langle\nu_1 + \nu_2\rangle\phi = \langle\nu_1\rangle\phi \vee \langle\nu_2\rangle\phi$
- $\langle\nu^*\rangle\phi = \phi \vee \langle\nu\rangle\phi \vee \langle\nu\rangle\langle\nu\rangle\phi \vee \ldots$
- $\langle\nu^+\rangle\phi = \langle\nu\rangle\phi \vee \langle\nu\rangle\langle\nu\rangle\phi \vee \ldots$

Example 12. With this extension it is now possible to express the properties above as follows:

i) $@_i\langle -^*.t\rangle \mathsf{true}$
ii) $[\![-]\!]\mathsf{false} \rightarrow \langle -^*.\mathsf{fifo_e}.-^*\rangle[-]\mathsf{false}$

5.4. Comparing reconfigurations

Equipped with a language to express structural properties of coordination patterns, we can define a new criterium for comparing reconfigurations. Without loss of generality we will identify the set of nominals in each model $\mathcal{M} = \langle \mathcal{G}(\rho), \sigma\rangle$ with the set $\mathcal{N}_\rho$ of nodes in pattern ρ, thus making $\sigma = id_{\mathcal{N}_\rho}$. Notation id_X stands for the identity on set X, *i.e.,* for all $x \in X$ $id_X(x) = x$. In such a context, we define:

Definition 17 *(Structural invariant).* A structural property ϕ is *invariant* for a reconfiguration r in a model $\mathcal{M} = \langle \mathcal{G}(\rho), id_{\mathcal{N}_\rho}\rangle$ iff it is preserved by r. Formally,

$$\langle \mathcal{G}(\rho), id_{\mathcal{N}_\rho}\rangle \models \phi \;\Rightarrow\; \langle \mathcal{G}(\rho \bullet r), id_{\mathcal{N}_{\rho\bullet r}}\rangle \models \phi$$

Then,

Definition 18 *(Structural equivalence).* Given a model $\mathcal{M} = \langle \mathcal{G}(\rho),\ id_{\mathcal{N}_\rho}\rangle$, reconfigurations r, r_1 and r_2 and a set of formulas Φ, it is said that

1. reconfiguration r *preserves* Φ iff every $\phi \in \Phi$ is *invariant* for r in $\mathcal{M}$;
2. reconfigurations r_1 and r_2 are *structurally equivalent* with respect to Φ, written $r_1 \equiv^{\mathcal{M}}_{\Phi} r_2$, iff

$$\langle \mathcal{G}(\rho \bullet r_1), id_{\mathcal{N}_{\rho\bullet r_1}}\rangle \models \phi \;\Leftrightarrow\; \langle \mathcal{G}(\rho \bullet r_2), id_{\mathcal{N}_{\rho\bullet r_2}}\rangle \models \phi$$

for every $\phi \in \Phi$.

In practice, however, reconfigurations entail a sort of *displacement* of most structural relationships in the coordination pattern. This means they often remain valid but at a different node in the pattern. A typical situation is illustrated in the following example.

Example 13. Consider the coordination pattern whose evolution is depicted in Fig. 12(a). Clearly, at node $\underline{cde}$ it is true that, after a connection made through a sync channel, all the others are established by lossy channels, *i.e.*

$$@_{\underline{cde}}\,\langle \mathsf{sync}\rangle(\langle -\rangle\mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false}).$$

Consider now that an insertP reconfiguration pattern is applied at node $\underline{cde}$. In a first step, node $\underline{cde}$ is split because of the application of the split elementary reconfiguration (Fig. 12(b)). Then a new structure is linked to the nodes resulting from

this operation (Fig. 12(c)). In both steps, however, the property is still valid for nodes $\underline{e}$ and $\underline{m_o e}$, respectively. With respect to the example in Fig. 12(c), this is expressed as

$$@_{\underline{m_o e}}\ \langle \mathsf{sync} \rangle (\langle - \rangle \mathsf{true} \wedge [-\mathsf{lossy}]\mathsf{false}).$$

This example motivates the following generalisation of Definition 17.

Definition 19. Let τ be a surjection on nominals. A structural property ϕ is *invariant* for a reconfiguration r, up to a name translation τ iff

$$\langle \mathcal{G}(\rho), id_{\mathcal{N}_\rho} \rangle \models \phi \ \Rightarrow\ \langle \mathcal{G}(\rho \bullet r), id_{\mathcal{N}_{\rho \bullet r}} \rangle \models \phi[\tau]$$

where $\phi[\tau]$ stands for ϕ with all occurrences of a nominal i replaced by $\tau(i)$.

Back to the example, if $\langle \mathcal{G}(\rho), \sigma \rangle \models \psi$, then $\langle \mathcal{G}(\rho \bullet r), \sigma \rangle \models \psi[\underline{cde} \mapsto \underline{m_o e}]$ for r the relevant insertP reconfiguration. Notation $[x \mapsto y]$ denotes a function that maps x to y and behaves as the identity in all other cases. Naturally, structural equivalence can be tailored to this more general notion of invariant.

6. A case study

This section introduces a (fragment of a) case study in the domain of Healthcare Information Systems (HIS), to illustrate the approach proposed in the paper. The implementation of Electronic Medical Records (EMR) systems brings benefits to safely store all patient records, including their clinical episodes information and replacing manually kept medical records, as well as to leverage the processing, interpretation and interchanging of medical information. EMR systems track and store patient records and institution activity by supporting all the general healthcare workflows from scheduling patient's appointments to their admission, observation, exam, prescription and billing. Some systems also embody more specific medical workflows concerning surgeries, ambulatory or post-surgery surveillance. The case study is concerned the management of emergency waiting times in Permanent Hospital Services.

6.1. Permanent service workflow

A simplified version of the generic workflow followed in permanent hospital services is depicted in Fig. 13. Informally, the patient arrives at the hospital and triggers the process by asking the administrative operator for a medical appointment. The operator registers the patient's admission as a new episode of his Electronic Patient Record (EPR). Later the patient is admitted to triage. Upon the triage outcome, the patient may be sent to an immediate medical appointment or asked to wait. Once admitted to a medical appointment, and depending on the severity of the symptoms, the healthcare professional may terminate the occurrence (often issuing a prescription) or send the patient for further exams. If this is the case, the patient waits to be called, then goes through the exam and finally, waits for the results. At this point, the patient meets the healthcare professional again and, depending on the examination results, he may be routed to a specialist or to more specific exams. Eventually the patient may be discharged, normally with a pharmacy prescription.

In an EMR system, this workflow is mapped into interacting services and a coordination layer to define the overall system behaviour, which can be modelled as a coordination pattern. In the sequel we will focus on the parts of the workflow involving waiting periods for the patients (the sub-workflow delimited by a dashed line in Fig. 13). The coordination pattern depicted in Fig. 14 is a service interaction model in this sub-workflow. It will be referred to as ρ_{ps}.

Note that, for simplicity, services Medical Appointment (*MAs*) and Examination (*Es*) are modelled as buffered communication channels (regarded, henceforth, as a new type of channels). On the other hand, w (waiting) channels represent the relevant waiting queues within the workflow. Other waiting times are left implicit within the Examination service. Port t_o is the unique input port of the coordination pattern to be connected to an external service for triage which is not represented here. Similarly, port h_o is the output port to be connected to a (also omitted) billing service. To simplify the picture, node $\bigotimes$ represents an *exclusive router* coordination pattern that accepts data on its input port (x_i), and, depending on context, routes data to one of its two output ports (x_1 or x_2). The mixed nodes are named a, e and es_i.

Usually, patient waiting times in the permanent hospital services are due to a lack of medical resources or to their inappropriate scheduling. To lower costs hospital administrators tend to schedule a low number of professionals to deal with the rather high number of emergency episodes.

The implementation of EMR systems opens the possibility to infer knowledge about the requirements depending on contextual parameters like periods of the day, number of patients, days in the week, etc., and to design adaptations based on such knowledge.

More interesting would be the possibility to anticipate a set of exams that would, most likely, be issued only during the medical appointment phase. The inference of such a set by a decision support service is made possible by taking into account the triage diagnosis, the patient's symptoms and the generic patients' clinical history stored in the EMR system. Therefore, some patients would be routed immediately for exam, while others remain in a waiting stage. Only when the

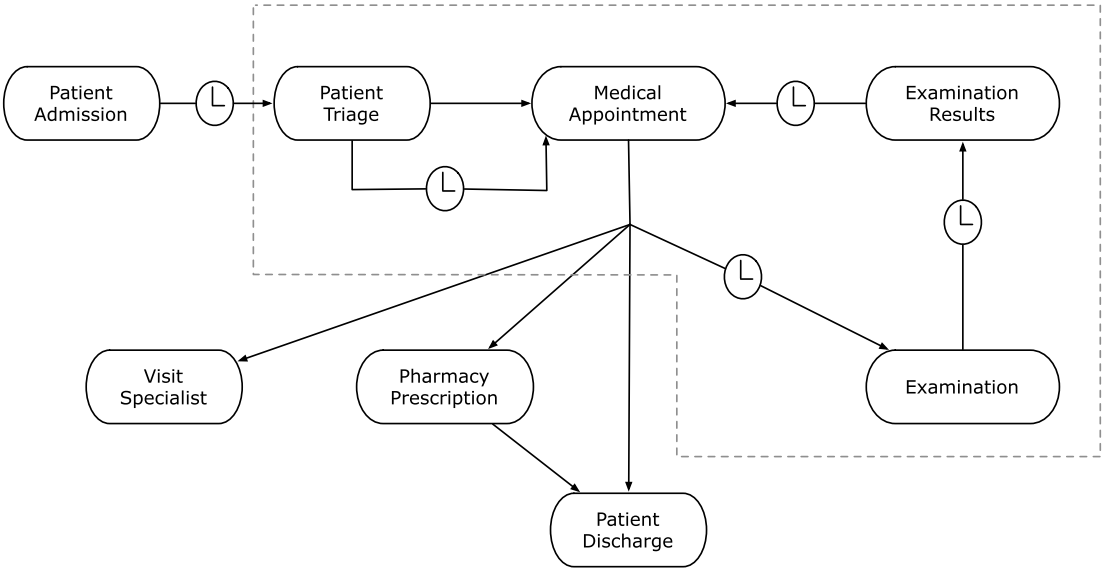


Fig. 13. A generic workflow for permanent hospital services.

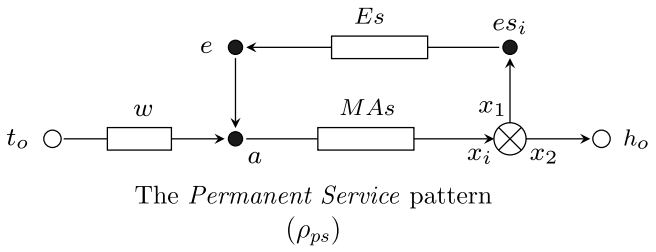


Fig. 14. The coordination pattern for part of the permanent service system.

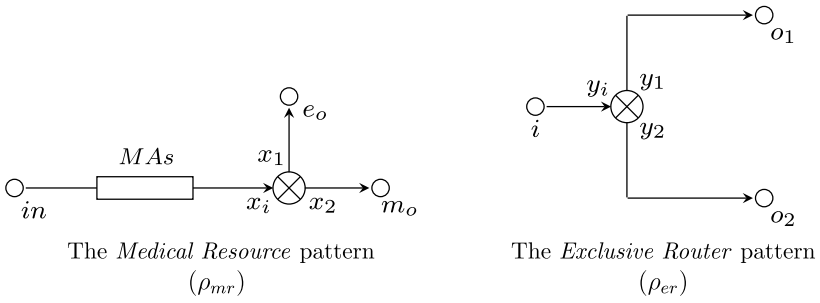


Fig. 15. Coordination patterns for the first reconfiguration: the *medical resource pattern* (left), is a variant of the *exclusive router* (right).

patients' symptoms are inconclusive to infer the set of exams to be carried on, would he be dispatched to the waiting queue. With this reconfiguration of the permanent service workflow, the overall waiting periods for the medical appointment are expected to decrease, upon specific contextual conditions.

6.2. Reconfiguring the workflow

To cut down waiting times two possible solutions can be considered: either to assign extra medical resources, or to anticipate a set of exams according to the symptoms of the patient. Let us consider the inclusion of a new medical resource in the system. The two coordination pattern, depicted in Fig. 15, will be of use in the sequel.

The *medical resource* coordination pattern (henceforth referred to as ρ_{mr}) represents the part of the protocol to deal with patient admittance for medical appointment and its final decision. The variant of the *exclusive router* (henceforth referred to

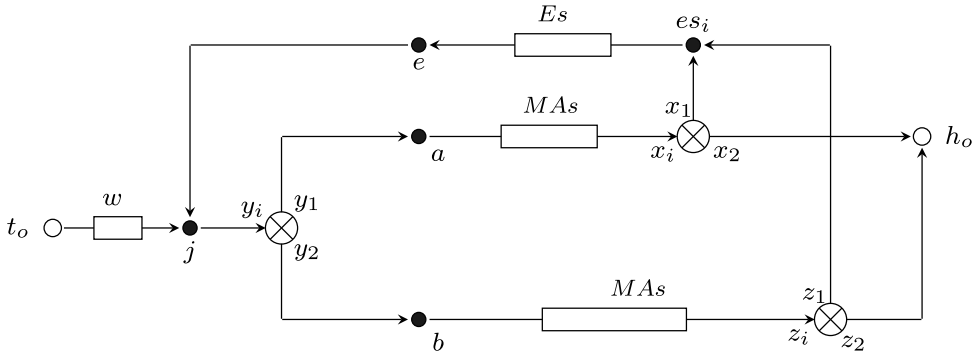


Fig. 16. The coordination pattern $\rho_{add_{res}}$ for the reconfigured permanent service system by the addition of medical resources.

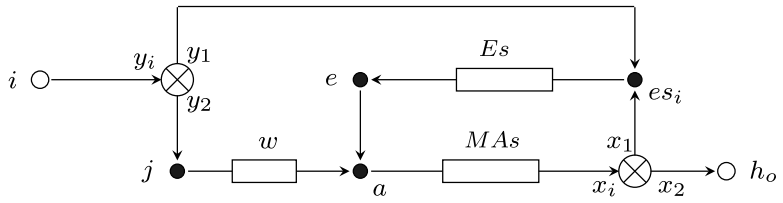


Fig. 17. The coordination pattern for the reconfigured permanent service system to support immediate examination period, referred to as ρ_{exams}.

as ρ_{er}) considers a number of synchronous channels to observe its ports. In this context, the reconfiguration $r_{add_{res}}$ to add a new medical resource is specified as

$$\mathsf{insertP}(\rho_{er}, a, i, o_1)\ ;\ \mathsf{overlapP}(\rho_{mr}, \{(o_2, in), (es_i, e_o), (h_o, m_o)\})$$

Application $\rho_{ps} \bullet r_{add_{res}}$ yields the desired reconfiguration on the permanent service system. The corresponding coordination pattern is depicted in Fig. 16 and referred to as $\rho_{add_{res}}$.

The second solution entails the need for a reconfiguration which adapts the system to decide the next step based on the patients' symptoms. For this we resort again to the ρ_{er} coordination pattern depicted in Fig. 15. A suitable reconfiguration is therefore

$$r_{exams} = \mathsf{overlapP}(\rho_{er}, \{(t_o, o_2), (es_i, o_1)\}).$$

The application of $\rho_{ps} \bullet r_{exams}$ reconfigures the permanent service system coordination as illustrated in Fig. 17, referred to, henceforth, as ρ_{exams}. With this reconfiguration, the system will automatically distribute patients either to be immediately examined or to wait in a queue for medical appointment.

6.3. Reasoning about reconfigurations

A typical requirement of hospital administrations is that *although the workflow may change at runtime, it must keep its original features* (as those embody the hospital business model). Therefore, any reconfiguration should meet this requirement. Let's see how this discussion can be made rigorous in our framework.

For the *behavioural* assessment of the two reconfigurations proposed one rely on *Reo automata*, assuming Reo as the concrete underlying coordination language. Clearly:

$$(\mathbf{1}_r \preccurlyeq r_{add_{res}})_{\rho_{ps}} \quad \text{and} \quad (\mathbf{1}_r \preccurlyeq r_{exams})_{\rho_{ps}}.$$

Therefore, both reconfigurations are classified as *expansive*. This means the original behaviour of the coordination layer of the permanent service system is preserved, but more functionality is added, as expected. In this sense the requirement above is preserved.

What one is able to say about the functionality added by each reconfiguration forces us to conclude that the two reconfigurations are behaviourally incomparable. There is not even a simulation relating them. Therefore, the emergent behaviours are different.

The *structural* perspective provides a different insight. The objective here is to check that both reconfigurations are structurally equivalent for a given set of properties. These properties are supposed to exist on the original coordination pattern and to remain true upon reconfiguration.

For illustration purposes, suppose the hospital administration placed the following two requirements to keep the system consistent with the main workflow: 1. a patient always meets a doctor in a medical appointment after triage; 2. the patient is always routed to a billing service at the end of the procedure. These requirements are essentially behavioural, and as

such discussed in terms of the underlying semantic model for Reo. However, they can also be analysed from a structural perspective: the question becomes to know if such a data flow is possible, *i.e.,* if there exist in the graph the necessary connections to make the intended flow *possible*. The requirements are then rephrased as: 1. there is a path from triage input port (t_o) to a *MAs* edge; 2. all paths from input ports, lead to the billing service (h_o) output port, which can be expressed in the hybrid logic introduced in Section 5 as follows:

1. $\phi_1 = @_{t_o}\langle - \rangle \mathsf{true} \wedge [-^*][-MAs]\mathsf{false}$
2. $\phi_2 = [\![-]\!]\mathsf{false} \rightarrow [-^*]h_o$.

Pattern $\langle t \rangle \mathsf{true} \wedge [-t]\mathsf{false}$ captures the fact of a connector of type t being the only outgoing connector from the current node, *i.e.,* the node at which the formula is evaluated. Expression $[\![-]\!]\mathsf{false}$, in ϕ_2, identifies which nodes are entry points into the coordination pattern as the ones which are not connected backwards.

We may now establish that both formulas are preserved by the two reconfigurations suggested above. Formally this amounts to show that, given model $\mathcal{M} = \langle \mathcal{G}(\rho_{ps}), id_{\mathcal{N}_{\rho_{ps}}} \rangle$, equivalence

$$\langle \mathcal{G}(\rho_{ps} \bullet r_{add_{res}}), id_{\mathcal{N}_{\rho_{ps} \bullet r_{add_{res}}}} \rangle \models \phi \;\Leftrightarrow\; \langle \mathcal{G}(\rho_{ps} \bullet r_{exams}), id_{\mathcal{N}_{\rho_{ps} \bullet r_{exams}}} \rangle \models \phi'$$

holds for $\phi' = \phi[i \mapsto t_o]$ for $\phi \in \{\phi_1, \phi_2\}$.

By inspection of diagrams depicted in Figs. 16 and 17 it is easy to verify that both properties, up to the given correspondence of nominals, are valid in both (reconfigured) patterns. For ϕ_1 note that in the first pattern there are only two possible connections from node t_o and both of them involve a *MAs* connector. After applying r_{exams} to ρ_{ps}, node t_o is replaced by a node i, which entails the need for introducing surjection $[i \mapsto t_o]$. Then, all connections from i link to a channel of type *MAs*. The argument for ϕ_2 similar, noting that t_o is in both cases the unique node with no incoming connections, thus satisfying $[\![-]\!]\mathsf{false}$.

7. Related work

The (essentially methodological) approach put forward in this paper is in debt to previous research on the reconfiguration of Reo connectors, in particular to the work of D. Clarke [24,25]. The first reference provides an axiomatisation of connector constructions (with a role similar to the primitive reconfiguration operations in the framework proposed here) to discuss connector equivalence. The second one introduces a basic modal logic to reason about reconfigurations, interpreted over constraint automata. What distinguishes our approach from this pioneering work is the graph-based representation of connectors (referred here as coordination patterns) and the independence with respect to the coordination model. Moreover, we distinguish behavioural from structural reasoning and stress the independence of the latter w.r.t. whatever behavioural semantic model is chosen for the coordination layer.

Structural and behavioural characterisations of dynamic update operations for reconfiguration of interacting systems also appear in the context of process calculi. M. Bravetti [26,27] shows that these different characterisations affect expressiveness and decidability/reachability results on adaptable processes. Our own approach also clearly separates the structural from the behavioural levels when analysing reconfigurations. Reconfigurations are characterised structurally (rather than behaviourally and structurally) but inspected under both perspectives.

Much work on reconfiguration has been conducted in graph-theoretical settings. Categorial accounts of graph grammars as formal frameworks for reasoning about software architectures and their dynamic changes are largely explored in the literature. Pioneer works [28,29] model software architectures as labelled graphs of components and connectors. The former uses productions on graph grammars to describe the style of an architecture and how it may evolve over time; the latter takes advantage of the double pushout approach [30,31] to express reconfigurations of a system architecture. R. Bruni and his collaborators [32] also adopt graph grammars applied to hypergraphs to both describe different notions of dynamic reconfiguration for software architectures.

D. Plump [33,34] introduces GP2, a programming language for solving graph problems, based on a notion of graph transformation and four operators shown to be Turing-complete. The language allows for the creation of programs over graphs at a high-level of abstraction. A program is a set of rules defining a transformation scheme based on the double pushout approach. Guards may be used to restrict the application of the program rules. A recent extension to Plump's work [35] introduces a Hoare logic for verifying graph programs. Although formulated in a different setting, the problem has several points of contact with our own effort to verify properties of coordination patterns.

The most recent and detailed work on architectural reconfigurations in a context close to ours is that of C. Krause [9, 36,37], in which Reo connectors are described by typed hypergraphs whose vertices are nodes and (typed hyper-)edges represent communication channels and components. High-level replacement systems, which form a powerful generalisation of algebraic graph-rewriting techniques to cope with enriched structures, from *e.g.* labelled graphs to Petri nets, are used to express reconfigurations. Those are specified as rewriting rules with a left-hand side (to be matched with a source structure) and a right-hand side (that specifies how the latter is changed). The authors claim several advantages of this approach based on graph rewriting: (*i*) the rewriting rules describe complex reconfigurations that are applied in an atomic step, rather than in a sequence of low-level operations; (*ii*) these rules are applied not only locally but also globally, wherever the left-hand

side of the rule matches the source structure; and (*iii*) by using positive or negative conditions associated to rewriting rules, it is possible to define in which specific situations the system should be changed. However, the approach is not fully compositional and, from our point of view, may become too complex to automate its application even in simple cases.

8. Conclusions

This paper introduces a framework for reasoning about reconfiguration of coordination patterns. Such patterns are described as graphs of primitive channels and their reconfigurations defined through the composition of a set of elementary operations: const, join, split, par and remove. Reconfigurations are assessed and compared according to two orthogonal perspectives: *behavioural*, based on the evaluation of the reconfigured pattern with respect to the underlying semantic model (Reo models were used for illustration purposes), and *structural*, in which properties of the interconnection graph are formulated in a variant of propositional hybrid logic.

This approach provides the basis to a systematic classification of reconfiguration strategies which are expected to lead to effective derivation of semi-automatic monitoring tools in the near future.

Although exposition resorted to Reo, as a concrete coordination setting, for illustration purposes, the basic ideas have a wider application. Recent experiments with BIP [38,39] show that the approach can easily be reused in the context of different coordination models, provided that they offer a well-defined notion of a software connector.

The reader may wonder why we have chosen the particular class of primitive reconfigurations introduced in Section 3 or the specific definitions of behavioural or structural equivalence put forward in Sections 4 and 5. The choice was essentially motivated by pragmatic considerations, leading to a small number of primitives upon which complex reconfigurations can be built compositionally and notions of equivalence intuitive for a software architect. Other options could have been considered, however. An important example, we would like to briefly discuss here, concerns the definition of structural properties.

Actually, the hybrid logic proposed in Section 5 to reason about reconfigurations from a structural point of view, is interpreted over a transition system whose transitions express the possibility of data flowing from one node to another through a channel of a certain type. Of course, for a coordination pattern ρ, the behavioural semantics assigned to connectors may prevent a specific sort of data flowing allowed by $\mathcal{G}(\rho)$. For example, a lossy channel gives rise to a transition from the node in which its source end participates to the one where its sink end lies, indicating the possible flow direction. However, data will only flow when a data request is present at the sink end. The structural view is blind with respect to this sort of behavioural restrictions.

We could, however, go a step further and totally forget the possible flow direction. For this the underlying transition system must connect by a transition of type t all nodes in which the ends of the corresponding channel $\langle \mathcal{S}, i, t, \mathcal{K} \rangle$ participate. Note that, a synchronous drain is ruled out from the *may-flow graph*, because data simply does not flow from one of its nodes to the other, but gives rise, in this new graph, to two transitions in opposite direction connecting the corresponding nodes. Note that this sort of double connection is always present if one adopts such a purely structural point of view: in other words, the transition system becomes a symmetric graph. Formally, it is built as

$$\bigcup_{\langle \mathcal{S},i,t,\mathcal{K}\rangle \in \mathcal{C}_\rho} \{\langle m, t, n\rangle \mid m, n \in \mathcal{N}_\rho \wedge m \cap (\mathcal{S} \cup \mathcal{K}) \neq \emptyset \wedge n \cap (\mathcal{S} \cup \mathcal{K}) \neq \emptyset \wedge n \neq m\}$$

Symmetry has a number of advantages: in particular it frees us from bothering with *past* modalities, *i.e.,* defined over the converse of $\mathcal{G}(\rho)$, as well as to consider this converse case in the definition of bisimulation. Apart from this simplification of the logic, all constructions and results remain. Whether such a purely structural view, capturing simply the existence of a connector between two nodes, irrespective of the data flow direction, is useful in practice for reasoning about coordination patterns, is debatable. It should be emphasised, however, that all tools are available for use with it, if such is the case.

A lot of work remains to be done. Providing tool support for both sorts of analysis is in order as well as classifying and organising reconfigurations in a suitable ontology. Tool support may build on already available tools either for the behavioural analysis of Reo circuits (*e.g.*, for CA, Vereofy [17]) or for hybrid logic verification (*e.g.*, HyloRes [22]).

Another line of enquire concerns *dynamic* reconfiguration, *i.e.,* the principles and mechanisms that may govern the application of reconfigurations at runtime. This is partially addressed by C. Krause [9]. In our case, as reconfigurations are defined as sequences of low-level (graph) operations, it will be mandatory to require their execution as transactions, with suitable roll-back mechanisms. Their definition is still current work.

Finally, it would also be interesting to extend the overall framework to deal with *quantitative* measures and express typical QoS constraints. We are currently working on this topic taking into account, on a reconfiguration, suitable measures of QoS levels attached to coordination patterns [40] and to their deployment context.

References

[1] F. Arbab, F. Mavaddat, Coordination through channel composition, in: F. Arbab, C. Talcott (Eds.), Coordination Models and Languages, in: Lecture Notes in Computer Science, vol. 2315, Springer, Berlin, Heidelberg, 2002, pp. 275–297, Ch. 6.

[2] F. Arbab, Reo: a channel-based coordination model for component composition, Math. Struct. Comput. Sci. 14 (3) (2004) 329–366.

[3] H. Gomaa, M. Hussein, Software reconfiguration patterns for dynamic evolution of software architectures, in: Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture, 2004, WICSA 2004, 2004, pp. 79–88.

[4] A.J. Ramirez, B.H.C. Cheng, Design patterns for developing dynamically adaptive systems, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '10, ACM, New York, NY, USA, 2010, pp. 49–58.
[5] P. Oreizy, N. Medvidovic, R.N. Taylor, Architecture-based runtime software evolution, in: Proceedings of the 20th International Conference on Software Engineering, ICSE '98, IEEE Computer Society, Washington, DC, USA, 1998, pp. 177–186.
[6] P. Hnětynka, F. Plášil, Dynamic reconfiguration and access to services in hierarchical component models, in: I. Gorton, G.T. Heineman, I. Crnković, H.W. Schmidt, J.A. Stafford, C. Szyperski, K. Wallnau (Eds.), Component-Based Software Engineering, in: Lecture Notes in Computer Science, vol. 4063, Springer, Berlin, Heidelberg, 2006, pp. 352–359, Ch. 27.
[7] M. Malohlava, T. Bures, Language for reconfiguring runtime infrastructure of component-based systems, in: Proceedings of MEMICS 2008, Znojmo, Czech Republic, 2008, pp. 1–8.
[8] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, J. Stefani, A component-based middleware platform for reconfigurable service-oriented architectures, Softw. Pract. Exp. 42 (5) (2012) 559–583.
[9] C. Krause, Reconfigurable component connectors, Ph.D. thesis, Leiden University, Amsterdam, The Netherlands, 2011.
[10] N. Oliveira, L.S. Barbosa, Reconfiguration mechanisms for service coordination, in: M. ter Beek, N. Lohmann (Eds.), Web Services and Formal Methods, in: Lecture Notes in Computer Science, vol. 7843, Springer, Berlin Heidelberg, 2013, pp. 134–149.
[11] N. Oliveira, L.S. Barbosa, On the reconfiguration of software connectors, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, Vol. 2, SAC '13, ACM, New York, NY, USA, 2013, pp. 1885–1892.
[12] S.-S.T.Q. Jongmans, F. Arbab, Overview of thirty semantic formalisms for Reo, Sci. Ann. Comput. Sci. 22 (1) (2012) 201–251.
[13] F. Arbab, Abstract behavior types: a foundation model for components and their composition, in: F.S. de Boer, M.M. Bonsangue, S. Graf, W.-P. de Roever (Eds.), Formal Methods for Components and Objects, in: Lecture Notes in Computer Science, vol. 2852, Springer, Berlin, Heidelberg, 2003, pp. 33–70, Ch. 2.
[14] D. Costa, Formal models for component connectors, Ph.D. thesis, Vrije University, Amsterdam, Oct. 2010.
[15] D. Clarke, D. Costa, F. Arbab, Connector colouring I: synchronisation and context dependency, Electron. Notes Theor. Comput. Sci. 154 (2006) 101–119.
[16] C. Baier, M. Sirjani, F. Arbab, J. Rutten, Modeling component connectors in Reo by constraint automata, Sci. Comput. Program. 61 (2) (2006) 75–113.
[17] C. Baier, T. Blechmann, J. Klein, S. Klüppelholz, A uniform framework for modeling and verifying components and connectors, in: Proc. 11th Int. Conf on Coordination Models and Languages, in: Lecture Notes Computer Science, vol. 5521, Springer, 2009, pp. 247–267.
[18] M. Bonsangue, D. Clarke, A. Silva, Automata for context-dependent connectors, in: Proceedings of the 11th International Conference on Coordination Models and Languages, COORDINATION '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 184–203.
[19] M.M. Bonsangue, D. Clarke, A. Silva, A model of context-dependent component connectors, Sci. Comput. Program. 77 (6) (2012) 685–706.
[20] P. Blackburn, Representation, reasoning, and relational structures: a hybrid logic manifesto, Log. J. IGPL 8 (3) (2000) 339–365.
[21] T. Brauner, Hybrid Logic and Its Proof-Theory, Applied Logic Series, Springer, 2010.
[22] C. Areces, J. Heguiabehere, HyLoRes 1.0: direct resolution for hybrid logics, in: A. Voronkov (Ed.), Automated Deduction—CADE-18, in: Lecture Notes in Computer Science, vol. 2392, Springer, Berlin Heidelberg, 2002, pp. 156–160.
[23] D. Kozen, Results on the propositional μ-calculus, Theor. Comput. Sci. 27 (1983) 333–354.
[24] D. Clarke, Reasoning about connector reconfiguration I: equivalence of constructions, Tech. rep., CWI-Centrum voor Wiskunde en Informatique, Amsterdam, Feb. 2005.
[25] D. Clarke, A basic logic for reasoning about connector reconfiguration, Fundam. Inform. 82 (2008) 361–390.
[26] M. Bravetti, C. Di Giusto, J.A. Pérez, G. Zavattaro, Adaptable processes, Log. Methods Comput. Sci. 8 (4) (2012) 1–71.
[27] M. Bravetti, C. Giusto, J. Pérez, G. Zavattaro, Towards the verification of adaptable processes, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, in: Lecture Notes in Computer Science, vol. 7609, Springer, Berlin Heidelberg, 2012, pp. 269–283.
[28] D. Hirsch, P. Inverardi, U. Montanari, Modeling software architectures and styles with graph grammars and constraint solving, in: P. Donohoe (Ed.), IFIP—The International Federation for Information Processing, in: Software Architecture, vol. 12, Springer US, 1999, pp. 127–143.
[29] M. Wermelinger, J.L. Fiadeiro, Algebraic software architecture reconfiguration, in: Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-7, Springer-Verlag, London, UK, 1999, pp. 393–409.
[30] H. Ehrig, M. Pfender, H.J. Schneider, Graph-grammars: an algebraic approach, in: 14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15–17, 1973, 1973, pp. 167–180.
[31] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, M. Löwe, Algebraic approaches to graph transformation—part I: basic concepts and double pushout approach, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations, World Scientific, 1997, pp. 163–246.
[32] R. Bruni, A. Bucchiarone, S. Gnesi, H. Melgratti, Modelling dynamic software architectures using typed graph grammars, Electron. Notes Theor. Comput. Sci. 213 (1) (2008) 39–53.
[33] D. Plump, The graph programming language GP, in: S. Bozapalidis, G. Rahonis (Eds.), Algebraic Informatics, in: Lecture Notes in Computer Science, vol. 5725, Springer, Berlin, Heidelberg, 2009, pp. 99–122, Ch. 6.
[34] D. Plump, The design of GP 2, in: S. Escobar (Ed.), Proceedings of the 10th International Workshop on Reduction Strategies in Rewriting and Programming, Novi Sad, Serbia, in: Electronic Proceedings in Theoretical Computer Science, vol. 82, 2011, pp. 1–16.
[35] C.M. Poskitt, D. Plump, Verifying total correctness of graph programs, Electron. Commun. EASST 61 (2013) 1–20.
[36] C. Koehler, A. Lazovik, F. Arbab, Connector rewriting with high-level replacement systems, Electron. Notes Theor. Comput. Sci. 194 (4) (2008) 77–92.
[37] C. Krause, Z. Maraikar, A. Lazovik, F. Arbab, Modeling dynamic reconfigurations in Reo using high-level replacement systems, Sci. Comput. Program. 76 (1) (2011) 23–36.
[38] A. Basu, M. Bozga, J. Sifakis, Modeling heterogeneous real-time components in BIP, in: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods, SEFM '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 3–12.
[39] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, J. Sifakis, Rigorous component-based system design using the BIP framework, IEEE Softw. 28 (3) (2011) 41–48.
[40] F. Arbab, T. Chothia, R. van der Mei, S. Meng, Y. Moon, C. Verhoef, From coordination to stochastic models of QoS, in: J. Field, V. Vasconcelos (Eds.), Coordination Models and Languages, in: Lecture Notes in Computer Science, vol. 5521, Springer, Berlin, Heidelberg, 2009, pp. 268–287, Ch. 14.