A survey on basic connectors and buffers*

Roberto Bruni[1], Hernán Melgratti[2], and Ugo Montanari[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Departamento de Computación, Universidad de Buenos Aires - Conicet, Argentina

Abstract. Recent years have witnessed an increasing interest about a rigorous modelling of (different classes of) connectors. Here, the term connector is used to name entities that can regulate the interaction of possibly heterogeneous components. Thus, connectors must take care of exogenous coordination, handling all those aspects that lie outside the scopes of individual components. This has led to the development of different frameworks that are used to specify, design, analyse, compare, prototype and implement connector-based middleware and a rigorous mathematical foundation of connectors is crucial for the analysis of exogenously coordinated systems. In this survey, we overview the main features of some notable theories of connectors, namely the algebra of stateless connectors, the tile model, Reo, BIP, nets with boundaries and the wire calculus. We discuss similarities, differences, mutual embedding and possible enhancements.

1 Introduction

The inherent complexity of modern distributed systems can only be tackled by modular engineering practices and methodologies that enhance the structural and logical blueprint of such systems. This way, it is possible to prove properties of the system either by construction, assembling well-behaving subsystems according to sound patterns, or by decomposition, dividing the systems and the property to be proved in smaller parts that can be analysed separately. *Component-based design* relies on the separation of concerns between coordination and computation. Component-based systems are built from sequential computational entities, the *components*, that should be loosely coupled w.r.t. the concurrent execution environments where they will be deployed. The component interfaces comprise the number, kind and peculiarities of communication ports. The communication media that make possible to interact are called *connectors*. Intuitively, they can be understood as (suitably decorated) channels or links among the ports of the components. Graphically, ports are represented as nodes and connectors as hyperarcs whose tentacles are attached to the ports they control. Several connectors can also be combined together by merging some of the ports their tentacles are attached to. Semantically, each connector imposes

* Research supported by the EU Integrated Project 257414 ASCENS, the Italian MIUR Project IPODS (PRIN 2008), ANPCyT Project BID-PICT-2008-00319, and UBACyT 20020090300122.

suitable constraints on the allowed communications among the components it links together. For example, a connector may impose handshaking between a sender component and a receiver component (Milner's CCS-like synchronization), or it may require an agreement on the action to be performed next by all components that it connects (Hoare's CSP-like synchronization). A different kind of connector may trigger the broadcasting of a message sent from one component to all the other linked components. The evolution of a network of components and connectors (just *network* for brevity) can be seen as if played in rounds: At each round, the components try to interact through their ports and the connectors allow/disallow some of the interactions selectively. A connector is called *stateless* when the interaction constraints that it imposes over its ports stay the same at each round; it is called *stateful* otherwise. To address composition and modularity of a system, networks are often decorated with (input and output) interfaces: in the simplest case, they consist of ports through which a network can interact. For example, two networks can be composed by merging the ports (i.e. nodes) they have in common. Ports that are not in the interface are typically private to the network and cannot be used to attach additional connectors. The distinction between input and output ports indicates in which direction the data should flow, but feedback is also possible through short-circuit connectors, which redirects some of the emitted output of a network to (some of) its input.

In this paper we survey some formal approaches to the modelling, composition and analysis of connectors, namely Reo [1], BIP [6], nets with boundaries [25], the algebra of stateless connectors [10], the tile model [17], and the wire calculus [24]. Although the approaches we shall consider are quite different in spirit, we will argue that they are different ways to look at the same entity. We briefly illustrate below the analysed frameworks by following the chronological order in which they were proposed. To expose the analogies and differences of the approaches, we shall use as a running example the modelling of compensation-based workflows typical of the area of business process modelling (see Section 2). We present the essential technical machinery underlying the considered theories in dedicated sections (the presentation order has been guided by practical dependencies arising in the descriptions of the different models). Some final considerations are reported in Section 8.

The algebra of stateless connectors and the tile model: An algebra consisting of five kinds of basic stateless connectors (plus their duals) has been presented in [10]. The connectors can be composed in series or in parallel. The operational, observational and denotational semantics of connectors are first formalised separately and then shown to coincide. Moreover, a complete normal-form axiomatisation is available for them.

The Tile Model [17, 9] offers a flexible and adequate semantic setting for concurrent systems [22, 15, 13] and also for defining the operational and abstract semantics of suitable classes of connectors, of which the algebra of stateless connectors is a particular instance. Tiles express the reactive behaviour of connectors in terms of $\langle \text{trigger}, \text{effect} \rangle$ pairs of labels. In this context, the

usual notion of equivalence is called *tile bisimilarity*. Tile bisimilarity is a congruence when a simple tile format is met by basic tiles [17].

The Reo coordination model: Reo [1] is an exogenous coordination model based on channel-like connectors that mediate the flow of data among components. Notably, a small set of point-to-point primitive connectors is sufficient to express a large variety of interesting constraints over the behaviour of connected components, including various forms of mutual exclusion, synchronization, alternation, and context-dependency. Components and primitive connectors can be composed into larger Reo circuits by disjoint union up-to the merging of shared Reo nodes. The semantics of Reo has been formalized in several ways, see [19] for a recent survey.

The BIP component framework: BIP [6] is a component framework for constructing systems by superposing three layers of modelling, called Behaviour, Interaction, and Priority. At the global level, the behaviour of a BIP system can be faithfully represented by a safe Petri net with priorities, whose single transitions are obtained by fusion of component transitions according to the permitted interactions, and priorities are assigned accordingly. An algebraic presentation of BIP connectors with vacuous priorities is given in [7]. One key feature of BIP is the so-called *correctness by construction*, which allows the specification of architecture transformations preserving certain properties of the underlying behaviour. For instance it is possible to provide (sufficient) conditions for compositionality and composability which guarantee deadlock-freedom. The BIP component framework has been implemented in a language and a tool-set. A compositional version of BIP systems is presented in [11].

Nets with boundaries and the wire calculus: Nets with boundaries takes inspiration from the open nets of [5]. The main idea is that nets are extended with input/output interfaces that can be used by transitions to synchronise their firings with the environment. C/E nets with boundaries can be composed in series and in parallel and come equipped with a labelled transition system that fixes their operational and bisimilarity semantics. The wire calculus [24] is a process algebra whose action prefixes come with an input/output arity typing. In [25, 12] a dialect of the wire calculus has been used to give an exact characterisation of a special class of (stateful) connectors that can be alternatively expressed in terms of nets with boundaries.

2 Running example

We will illustrate the different approaches surveyed in this paper by modelling the basic operator used for defining *Long Running Transactions* (LRT), i.e., transactions that may require long periods of time to complete. The implementation of LRT does not use locking (as usual for database transactions), but they rely instead on a weaker notion of atomicity based on compensations [18]. Compensations are activities programmed ad hoc to recover partial executions of transactional processes. Then, a LRT is a group of activities that must be all

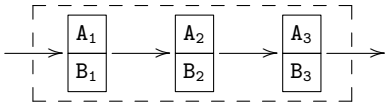


Fig. 1. A sequential saga.

Fig. 2. Graphical representation of Reo basic connectors

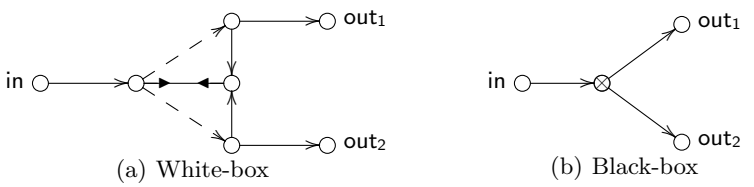
either successfully executed or compensated otherwise. Consider the LRT presented in Fig. 1, where the transaction consists in the sequential execution of the activities $\mathtt{A}_1$, $\mathtt{A}_2$ and $\mathtt{A}_3$, that can be compensated respectively by $\mathtt{B}_1$, $\mathtt{B}_2$ and $\mathtt{B}_3$. Suppose now that the activity $\mathtt{A}_1$ completes successfully while the activity $\mathtt{A}_2$ fails. In this case, the failure of $\mathtt{A}_2$ activates the execution of the compensation $\mathtt{B}_1$ to undo as much as possible the effects of $\mathtt{A}_1$, because the transaction failed as a whole. Note that $\mathtt{B}_2$ is not executed, because $\mathtt{A}_2$ has not completed. Differently, if both $\mathtt{A}_1$ and $\mathtt{A}_2$ succeed while $\mathtt{A}_3$ fails, then the compensations will be executed in the reverse order, i.e., first $\mathtt{B}_2$ and then $\mathtt{B}_1$.

Recent years have seen an increasing interest in compensation-based languages for LRT, especially in the area of business process modelling [26], mostly exploiting standard form of composition (sequential, branching, parallel). The *compensation pair* $\mathtt{A}\%\mathtt{B}$ is one key operation common to most of them, whose modelling as connector middleware shall be our running example.

3 The Reo coordination model

Reo [1] is a connector-based exogenous coordination model. Connectors are essentially graphs where the edges are user-defined communication channels and the nodes implement a fixed routing policy. Reo *channels* are entities that have exactly two ends, also referred to as *ports*, which can be either source or sink ends: Source ends accept data into, and sink ends dispense data out of their channels. Typical primitive connectors are: (i) the Sync channel, which allows a data item to flow from its source end to its sink end when the latter is able to accept it; (ii) the LossySync channel, similarly to the Sync channel but the data item is lost when the sink end is not ready to accept it; (iii) the SyncDrain channel, which is a channel with two source ends that accept data simultaneously and dispense them subsequently; (iv) the FIFO channel, which is an asynchronous channel with a buffer of capacity one. The set of primitive channels is completed with AsyncDrain, Filter, Transformer, Timer (their definition can be found at [1]). The graphical representation of basic channels is shown in Fig. 2.

Components and primitive connectors can be composed into larger Reo circuits by disjoint union, up-to the merging of shared Reo nodes with the same



(a) White-box (b) Black-box

Fig. 3. Reo exclusive router connector

name (this operation is called *join*). Note that a joint node behaves asymmetrically: in input, the node takes non-deterministically a message from one of its incoming channels (the other channels must remain idle); in output, the selected data is written simultaneously to all outgoing channels (that must be able to accept the message). Nodes of a connector can be hidden before composition in order to avoid further joins over those particular nodes. In graphical representation we will leave all hidden nodes unnamed. Figure 3(a) illustrates a well-known composite Reo connector, called *exclusive router*. It joins five Sync, two LossySync and one SyncDrain. The connector provides three visible nodes in, out_1 and out_2. Any data item read on the input port in is written in only one of its output ports out_1 or out_2, depending on which one is ready to consume it. When both out_1 and out_2 are ready to read, then the connector chooses non deterministically one of them. We remark that an input data is never replicated to more than one of its output ports. As a shorthand, we will represent the exclusive router connector as shown in Fig. 3(b).

The semantics of Reo has been formalized in several ways, exploiting, e.g., coalgebraic techniques [3], constraint-automata [4], colouring tables [14], and the tile model [2]. We illustrate here the denotational approach called the two-colour semantics. The two-colour semantics relies on two colours to denote the presence and absence of a message on a port (1 and 0 respectively). The semantics of a connector is defined in terms of the valid assignments of colours to its ports. The tables in Fig. 4 show the valid assignments for some basic connectors. The definition is straightforward for stateless connectors such as Sync, LossySync and SyncDrain. For stateful connectors, constraints have to be provided for any possible state of the connector. Note that the semantics of the FIFO connector is given in terms of two different states, i.e., empty and full. Moreover, the semantic tables define also the state transitions of the connector. Finally, the semantics of a complex Reo circuit corresponds to the set of all possible colour assignments that are consistent with the colouring tables of the involved connectors and reflect the behaviour of joint nodes (i.e., at most one incoming arc has colour 1 and all ongoing arcs have the same colour: 0 when no incoming arc is coloured with 1 and 1 otherwise). In this way, we derive the semantics of complex connectors. For example, valid assignments for $(\mathsf{in}, \mathsf{out}_1, \mathsf{out}_2)$ in the exclusive router are $(0,0,0)$, $(1,0,1)$ and $(1,1,0)$.

Sync

in	out
0	0
1	1

LossySync

in	out
0	0
1	0
1	1

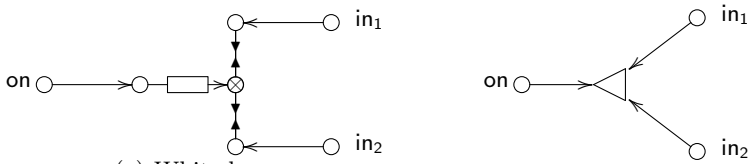
SyncDrain

in	out
0	0
1	1

FIFO

empty			full		
in	out	state	in	out	state
0	0	empty	0	0	full
1	0	full	0	1	empty

Fig. 4. Two-colour semantics of Reo basic connectors



(a) White-box (b) Black-box

Fig. 5. The stateful selector

3.1 Compensation Pair in Reo

We report here the definition of a compensation pair in Reo as proposed in [21]. We start by presenting an auxiliary connector, named *stateful selector*, which will be used for encoding a compensation pair into a Reo circuit. The stateful selector, depicted in Fig. 5(a), behaves as follows: At the initial state this connector can only accept a message over port on because a synchronization over in_1 and in_2 would require a message in the FIFO channel. A synchronization over on puts a message in the FIFO channel that next enables the synchronization over just one of the ports in_1 and in_2. Note that in_1 and in_2 are in mutual exclusion due to the connector $\otimes$, and hence, the connector can accept just one message on either in_1 or in_2. The connector returns to its initial state after this synchronization. As a shorthand we will depict the stateful selector as shown in Fig. 5(b).

The Reo circuit modelling the compensation pair A%B is shown in Fig. 6. The execution flow starts when a message is written in channel Start, which activates the execution of the activity A. After completion, A will write a message on its output port, which will set the stateful selector and write a message on port Performed for signalling that the activity has been successfully executed (this could serve for instance to activate the next activity in the flow). Eventually, the performed task will be cancelled or committed, after which the effects of a committed task cannot be undone or cancelled anymore. If a Cancel message arrives, the compensation activity B is executed and the task A is considered to be cancelled (this is signalled by sending a message in port Cancelled). If a Commit message arrives, the port Committed emits also a message. The messages to commit or cancel the task are generated from the controller of the transaction. (For simplicity, we omit details here and refer interested readers to [21]).

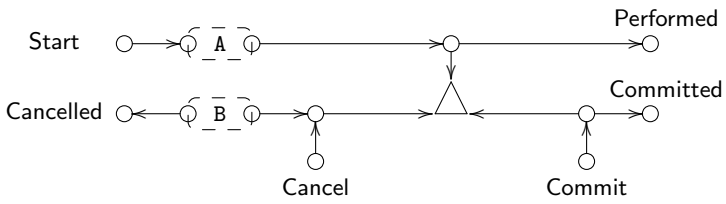


Fig. 6. Reo circuit for the compensation pair A%B

4 The BIP component framework, and BI(P)

BIP [6] is a component framework that exploits a three-layered architecture: 1) the lower level is called *Behaviour* and it fixes the activities of individual atomic components; 2) the middle layer is called *Interaction* and it defines the handshaking mechanisms between components; and 3) the top level is called *Priority* and it assigns a partial order of preferences to the admissible interactions. This section recalls the formal definition of BIP using the notation from [8]. Here we disregard priorities for simplicity, and thus we name BI(P) the presented framework.

The lower layer consists of a set of atomic components with ports. The sets of ports of components are pairwise disjoint, i.e., each port is uniquely assigned to a component. Components are modelled as automata whose transitions are labelled by sets of ports.

Definition 1 (Component). *A* component $B = (Q, P, \rightarrow)$ *is a transition system where Q is a set of states, P is a set of ports, and $\rightarrow \subseteq Q \times 2^P \times Q$ is the set of labelled transitions.*

As usual, we write $q \xrightarrow{a} q'$ to denote the transition $(q, a, q') \in \rightarrow$. We say that a is enabled in q, denoted $q \xrightarrow{a}$, iff there exists q' s.t. $q \xrightarrow{a} q'$. We assume that for all q, q' it holds $q \xrightarrow{\varnothing} q'$ iff $q = q'$.

The second layer consists of connectors that specify the allowed interactions between components.

Definition 2 (Interaction). *Given a set of ports P, an* interaction *over P is a non-empty subset $a \subseteq P$.*

We write an interaction $\{p_1, p_2, \ldots, p_n\}$ as $p_1 p_2 \ldots p_n$ and $a \downarrow_{P_i}$ for the projection of $a \subseteq P$ over the set of ports $P_i \subseteq P$, i.e., $a \downarrow_{P_i} = a \cap P_i$.

Definition 3 (BI(P) system). *A* BI(P) system $B = \gamma(B_1, \ldots, B_n)$ *is the composition of a finite set $\{B_i\}_{i=1}^n$ of transitions systems $B_i = (Q_i, P_i, \rightarrow_i)$ such that their sets of ports are pairwise disjoint, i.e., $P_i \cap P_j = \varnothing$ for $i \neq j$, parametrized by a set $\gamma \subset 2^P$ of interactions over the set of ports $P = \biguplus_{i=1}^n P_i$. We call P the underlying set of ports of B, written $\iota(B)$.*

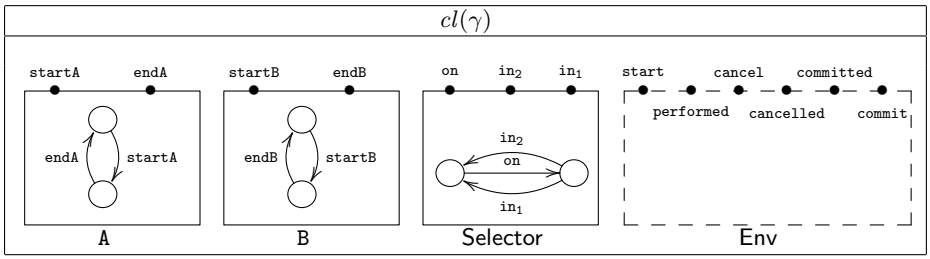


Fig. 7. A simple BIP for compensation pair A%B

The semantics of a BI(P) system $\gamma(B_1, \ldots, B_n)$ is given by the transition system $(Q, P, \rightarrow_\gamma)$, with $Q = \Pi_i Q_i$, $P = \biguplus_{i=1}^{n} P_i$ and $\rightarrow_\gamma \subseteq Q \times 2^P \times Q$ is the least set of transitions satisfying the following inference rule

$$\frac{a \in \gamma \qquad \forall i \in 1..n : q_i \xrightarrow{a\downarrow_{P_i}} q_i'}{(q_1, \ldots, q_n) \xrightarrow{a}_\gamma (q_1', \ldots, q_n')}$$

Note that the interactions in γ are pairwise mutually exclusive, e.g., is $a, b \in \gamma$ it is not necessarily the case that $ab \in \gamma$. We find it convenient to introduce the shorthand $cl(\gamma)$ as the closure of γ w.r.t. set union, i.e., $cl(\gamma)$ is the least set such that $\gamma \in cl(\gamma)$ and $\forall a, a' \in cl(\gamma).\ aa' \in cl(\gamma)$.

4.1 Compensation Pair in BI(P)

A BI(P) system modelling the behaviour of a compensation pair A%B is shown in Fig. 7, where

$$\gamma = \{\ \{\mathtt{start}, \mathtt{startA}\}, \{\mathtt{performed}, \mathtt{endA}, \mathtt{on}\}, \{\mathtt{cancel}, \mathtt{in}_1, \mathtt{startB}\},$$
$$\{\mathtt{endB}, \mathtt{cancelled}\}, \{\mathtt{commit}, \mathtt{in}_2, \mathtt{committed}\}\ \}.$$

As in previous cases, we assume basic activities A and B to be defined as components with two ports: one for activating its execution (named, startA and startB, respectively) and other for signalling the completion of the activity (named, endA and endB, respectively). Moreover, we assume that any initiated execution is completed before starting another execution. Hence, basic activities are modelled as automata with just two states (see Fig. 7). We also rely on a component Selector, which behaves analogously to the stateful selector defined in Reo. In addition, we consider a fourth component Env representing the environment in which the compensation pair will execute. This component acts as the transaction manager that coordinates the execution of the whole transaction. As such, it is in charge of starting the execution of the compensation pair at the proper moment (action start) and then it decides whether to commit or to

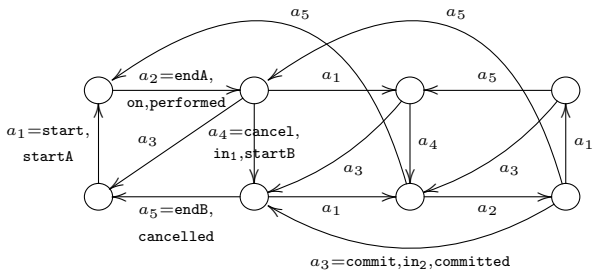


Fig. 8. Synthesized behaviour of the BI(P) system for A%B

cancel the already executed activity. The synchronization set $cl(\gamma)$ in Fig. 7 defines all the allowed movements of the system. For instance, γ contains just one synchronization for the action start, which requires the simultaneous execution of startA, i.e., the environment may perform start only when the component A is able to perform startA. Similarly, synchronization {endA, on, performed} implies that the termination of A enables the component Selector and makes the environment to move with action performed. Assuming the more liberal definition of Env (that does not introduce deadlocks), the behaviour of the system can be summarized with the LTS in Fig. 8. Note that, if Env guarantees that multiple instances of the compensation pair are serialized, then only the four leftmost states are meaningful. Otherwise, it is possible to handle simultaneously up to three instances of the pair: executing one instance of A (third serve), one of B (first serve) with selector on (second serve). We remark that the same behaviour arises in the Reo circuit for the compensation pair when we replace A and B by FIFO connectors.

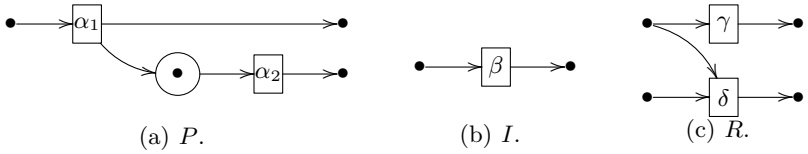
5 Nets with Boundaries

This section summarizes the basic of C/E nets with boundaries introduced in [25]. C/E nets with boundaries are a compositional version of C/E nets that come equipped with a notion of sequential and parallel composition. Contrary to previous proposals in the literature for composing Petri nets, the ports in their boundaries are neither places nor transitions, but rather handshaking points.

We start by describing elemental Petri nets and then we introduce nets with boundaries. Petri nets [23] consist of *places*, which are repositories of *tokens*, and *transitions* that remove and produce tokens.

Definition 4 (Net). *A* net N *is a 4-tuple* $N = (S_N, T_N, {}^\circ{-}_N, {-}^\circ_N)$ *where* S_N *is the (nonempty) set of places,* $\mathtt{a}, \mathtt{a}', \ldots$, T_N *is the set of transitions,* $\mathtt{t}, \mathtt{t}', \ldots$ *(with* $S_N \cap T_N = \varnothing$*), and the functions* ${}^\circ{-}_N, {-}^\circ_N : T_N \to 2^{S_N}$ *assign finite sets of places, called respectively source and target, to each transition.*

Transitions t, u are independent when ${}^\circ t \cap {}^\circ u = t^\circ \cap u^\circ = \varnothing$. This notion of independence allows so-called contact situations. Moreover, it also allows



(a) P. (b) I. (c) R.

Fig. 9. Three nets with boundaries

consume/produce loops, i.e., a place p can be both in ${}^\circ t$ and t°. A set U of transitions is mutually independent when, for all $t, u \in U$, if $t \neq u$ then t and u are independent. Given a set of transitions U let ${}^\circ U = \cup_{u \in U} {}^\circ u$ and $U^\circ = \cup_{u \in U} u^\circ$.

Definition 5 (Semantics). *Let $N = (P, T, {}^\circ -, -^\circ)$ be a net, $X, Y \subseteq P$ and $t \in T$. Write:*

$$(N, X) \to_{\{t\}} (N, Y) \stackrel{\mathsf{def}}{=} {}^\circ t \subseteq X \wedge t^\circ \subseteq Y \wedge X \backslash {}^\circ t = Y \backslash t^\circ$$

For $U \subseteq T$ a set of mutually independent transitions, write:

$$(N, X) \to_{U} (N, Y) \stackrel{\mathsf{def}}{=} {}^\circ U \subseteq X \wedge U^\circ \subseteq Y \wedge X \backslash {}^\circ U = Y \backslash U^\circ$$

Note that, for any $X \subseteq P$, $(N, X) \to_{\varnothing} (N, X)$. States of this transition system will be referred to as markings of N.

For the definition of nets with boundaries we let $\underline{k}$, $\underline{l}$, $\underline{m}$, $\underline{n}$ range over finite ordinals: $\underline{n} \stackrel{\mathsf{def}}{=} \{0, 1, \ldots, n-1\}$.

Definition 6 (Nets with boundaries). *Let $m, n \in \mathbb{N}$. A net with boundaries $N : m \to n$ is a tuple $N = (S, T, {}^\circ -, -^\circ, {}^\bullet -, -^\bullet)$ where $(S, T, {}^\circ -, -^\circ)$ is a net and functions ${}^\bullet - : T \to 2^{\underline{m}}$ and $-^\bullet : T \to 2^{\underline{n}}$ assign transitions to the left and right boundaries of N, respectively.*

The representation of the left and right boundaries as ordinals is just a notational convenience. In particular, we remark that the left and the right boundaries of a net are always disjoint.

The notion of independence of transitions extends to nets with boundaries in the obvious way: $t, u \in T$ are said to be *independent* when

$${}^\circ t \cap {}^\circ u = \varnothing \ \wedge \ t^\circ \cap u^\circ = \varnothing \ \wedge \ {}^\bullet t \cap {}^\bullet u = \varnothing \ \wedge \ t^\bullet \cap u^\bullet = \varnothing$$

Example 1. Figure 9 shows three different nets with boundaries. Places are circles and a marking is represented by the presence or absence of tokens; rectangles are transitions and arcs stand for pre and postset relations. The left interface (right interface) is depicted by points situated on the left (respectively, on the right). Figure 9(a) shows the net $P : 1 \to 2$ containing one place, two transitions and one token. Nets $I : 1 \to 1$ and $R : 3 \to 2$ have no places: the former, called *identity*, forwards tokens received on its input port to the output port; the latter has two competing transitions γ and δ for the tokens arriving on the top-positioned input port, and δ requires also a token from the bottom-positioned input port.

Nets with boundaries can be composed in parallel and in series. Given $N : m \to n$ and $M : k \to l$, their tensor product is the net $N \otimes M : m + k \to n + l$ whose sets of places and transitions are the disjoint union of the corresponding sets in N and M, whose maps ${}^\circ{-}, {-}^\circ, {}^\bullet{-}, {-}^\bullet$ are defined according to the maps in N and M and whose initial marking is $m_{0N} \oplus m_{0M}$. Intuitively, the tensor product corresponds to draw the nets N and M one above the other.

The sequential composition $N; M : m \to k$ of $N : m \to n$ and $M : n \to k$ is slightly more involved and relies on the following notion of synchronization: a pair (U, V) with $U \subseteq T_N$ and $V \subseteq T_M$ mutually independent sets of transitions such that: (1) $U \cup V \neq \varnothing$ and (2) $U^\bullet = {}^\bullet V$.

The set of synchronisations inherits an ordering from the subset relation, i.e. $(U, V) \subseteq (U', V')$ when $U \subseteq U'$ and $V \subseteq V'$. A synchronisation is said to be minimal when it is minimal with respect to this order. Let

$$T_{N;M} \stackrel{\mathsf{def}}{=} \{(U, V) | U \subseteq T_N, V \subseteq T_M, (U, V) \text{ a minimal synchronisation}\}$$

Notice that any transition t in N (respectively t' in M) not connected to the shared boundary n defines a minimal synchronisation $(\{t\}, \varnothing)$ (respectively $(\varnothing, \{t'\})$) in the above sense. The sequential composition of N and M is written $N; M : m \to k$ and defined as $(S_N \uplus S_M, T_{N;M}, {}^\circ{-}_{N;M}, {-}^\circ_{N;M}, {}^\bullet{-}_{N;M}, {-}^\bullet_{N;M})$, where pre- and post-sets of synchronizations are defined as

- ${}^\circ(U, V)_{N;M} = {}^\circ(U)_N \uplus {}^\circ(V)_M$ and $(U, V)^\circ_{N;M} = (U)^\circ_N \uplus (V)^\circ_M$
- ${}^\bullet(U, V)_{N;M} = {}^\bullet(U)_N$ and $(U, V)^\bullet_{N;M} = (V)^\bullet_M$.

Intuitively, transitions attached to the left or right boundaries can be seen as transition fragments, that can be completed by attaching other complementary fragments to that boundary. When two transition fragments in N share a boundary node, then they are two mutually exclusive options for completing a fragment of M attached to the same boundary node. Thus, the idea is to combine the transitions of N with that of M when they share a common boundary, as if their firings were synchronized. As in general several combinations are possible, only minimal synchronizations are selected.

Example 2. Let P, I and R be the nets in Fig. 9. Then, the net $(P \otimes I); (I \otimes R)$ obtained as the composition of P, R and two copies of I is shown in Fig. 10.

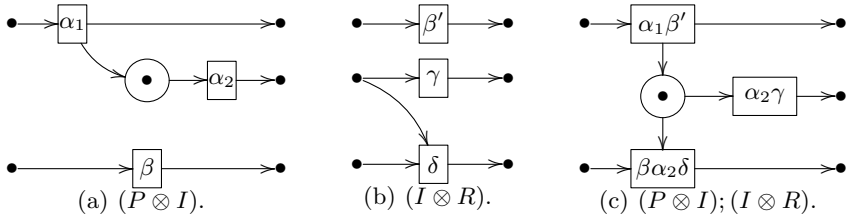
Sometimes we find convenient to write $N = (S, T, {}^\circ{-}, {-}^\circ, {}^\bullet{-}, {-}^\bullet, X)$ with $X \subseteq S$ for the net $(S, T, {}^\circ{-}, {-}^\circ, {}^\bullet{-}, {-}^\bullet)$ with initial marking X and extend the sequential and parallel composition to nets with initial marking by taking the union of the initial markings.

For any $k \in \mathbb{N}$, there is a bijection $\ulcorner \_ \urcorner : 2^{\underline{k}} \to \{0, 1\}^k$ with

$$\ulcorner U \urcorner_i \stackrel{\mathsf{def}}{=} \begin{cases} 1 \text{ if } i \in U \\ 0 \text{ otherwise} \end{cases}$$

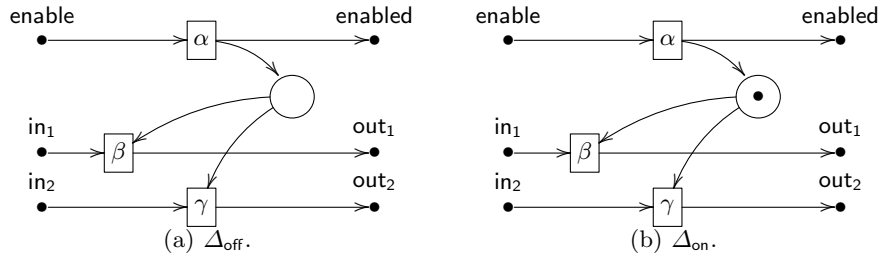
Definition 7 (Semantics). *Let $N : n \to n$ be a net and $X, Y \subseteq P_N$. Write:*

$$(N, X) \xrightarrow[\beta]{\alpha} (N, Y) \stackrel{\mathsf{def}}{=} \exists \;\; \textit{mutually independent } U \subseteq T_N \;\; \textit{s.t}$$
$$(N, X) \to_U (N, Y), \alpha = \ulcorner {}^\bullet U \urcorner, \;\; \textit{and } \beta = \ulcorner U^\bullet \urcorner$$



(a) $(P \otimes I)$. (b) $(I \otimes R)$. (c) $(P \otimes I); (I \otimes R)$.

Fig. 10. Composition of nets with boundaries



(a) Δ_{off}. (b) Δ_{on}.

Fig. 11. Net with boundaries for the stateful selector

5.1 Compensation Pair as a Net with Boundaries

We start by introducing an auxiliary net that will be used for encoding a compensation pair into a net with boundaries. Figure 11 depicts a net modelling a stateful selector, analogous to the Reo circuit introduced in Section 3.1. The main difference with the Reo circuit is that we distinguish here between input and output ports (note that nodes in Reo allow both input and output actions). Consequently, we use three input ports enable, in_1 and in_2, which correspond to to the input behaviour of the homonymous nodes in Fig. 5(a). Similarly, the output ports $\mathsf{enabled}$, out_1 and out_2 are considered. Figure 11(a) depicts the initial state of the connector, abbreviated as Δ_{off}, in which selection is not enabled, while Fig. 11(b) shows the state Δ_{on} in which selection is enabled. The allowed movements of the connector are:

1. $\Delta_{\mathsf{off}} \xrightarrow[000]{000} \Delta_{\mathsf{off}}$, i.e., the connector is idle;
2. $\Delta_{\mathsf{off}} \xrightarrow[100]{100} \Delta_{\mathsf{on}}$, i.e., selection has been enabled.
3. $\Delta_{\mathsf{on}} \xrightarrow[000]{000} \Delta_{\mathsf{on}}$, i.e., the connector remains idle;
4. $\Delta_{\mathsf{on}} \xrightarrow[010]{010} \Delta_{\mathsf{off}}$, i.e., input in_1 is chosen;
5. $\Delta_{\mathsf{on}} \xrightarrow[001]{001} \Delta_{\mathsf{off}}$, i.e., input in_2 is chosen.

We remark that Δ_{on} returns to the initial state Δ_{off} after a selection takes place.

We assume any activity A to be modelled as a net with boundaries $[\![A]\!]$ with just one input and one output port, i.e., $[\![A]\!] : \underline{1} \to \underline{1}$. In addition, we assume

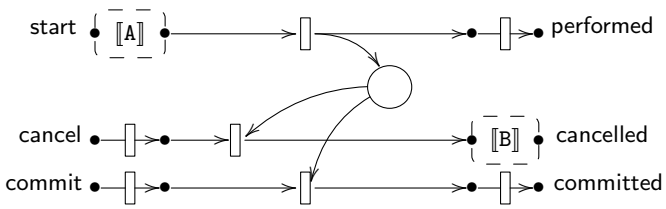


Fig. 12. Net with boundaries for ⟦A%B⟧

that $[\![A]\!]$ is well-defined, and that every started execution ends by signalling the completion of the task with a signal over the output port. We will also use the identity net introduced in Fig. 9(b). Finally, the net corresponding to the compensation pair A%B is the net $[\![\mathtt{A\%B}]\!] : \underline{3} \to \underline{3}$ defined as follows

$$[\![\mathtt{A\%B}]\!] = ([\![A]\!] \otimes I \otimes I); \Delta_{\mathsf{off}}; (I \otimes [\![B]\!] \otimes I)$$

A graphical representation of $[\![\mathtt{A\%B}]\!]$ is in Fig. 12. Initially, the only allowed movement of the net is the one that initiates the execution of $[\![\mathtt{A}]\!]$ (i.e., a message received on port start). Subsequently, the completion of $[\![\mathtt{A}]\!]$ will be signalled on the output port of the net representing the task. This will fire the top leftmost transition of the net, and consequently a token will be produce in the unique place of the net and a signal will be emitted on port performed. Afterwards, the net will be able to accept one signal on either cancel or commit port. A signal on port cancel will activate the execution of the compensation $[\![\mathtt{B}]\!]$, which will eventually complete and a signal on port cancelled will be produced.

6 Tiles, Wires and the Petri Calculus

The Petri calculus [25] is an algebra of stateful connectors, which basically extends the algebra of stateless connectors from [10] with one-place buffers. It can also be seen as an instance of the tile model or of the wire calculus.

The algebra of stateless connectors [10] consists of five kinds of basic connectors (plus their duals), namely symmetry, synchronization, mutual exclusion, hiding and inaction. The connectors can be composed in series or in parallel. The operational, observational and denotational semantics of connectors are first formalised separately and then shown to coincide. Moreover, a complete normal-form axiomatisation is available for them. These networks are quite expressive: for instance it is shown [10] that they can model all the (stateless) connectors of the architectural design language CommUnity [16]. This result is of particular interest, because it reconciles the algebraic and categorical approaches to system modelling, of which the algebra of stateless connectors and CommUnity are suitable representatives. The algebraic approach models systems as terms in a suitable algebra. Operational and abstract semantics are then usually based on

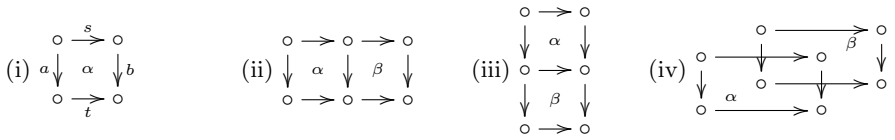


Fig. 13. Examples of tiles and their composition

inductively defined labelled transition systems. The categorical approach models systems as objects in a category, with morphisms defining relations such as subsystem or refinement. Complex software architectures can be modelled as diagrams in the category, with universal constructions, such as colimit, building an object in the same category that behaves as the whole system and that is uniquely determined up to isomorphisms. While in the algebraic approach equivalence classes are usually abstract entities, having a normal form gives a concrete representation that matches a nice feature of the categorical approach, namely that the colimit of a diagram is its best concrete representative.

The tile model [17, 9] offers a convenient framework for defining the operational and abstract semantics of connectors. For example, the operational semantics of the algebra of stateless connectors is given in terms of the tile model, which has later been extended to deal with one place buffers in [2]. Also the operational semantics of the Petri calculus, originally defined as a dialect the wire calculus [24], can be straightforwardly represented in the tile model. Tile bisimilarity provided a standard observational congruence in all the above cases.

The name 'tile' is due to the graphical representation of such rules (see Fig. 13). A tile $\alpha : s \xrightarrow[b]{a} t$ is a rewrite rule stating that the *initial configuration* s can evolve to the *final configuration* t via α, producing the *effect* b; but the step is allowed only if the 'arguments' of s can contribute by producing a, which acts as the *trigger* of α (see Fig. 13(i)). Triggers and effects are called *observations* and tile vertices are called *interfaces*.

Roughly, the semantics of component-based systems can be expressed via tiles when: i) components and connectors are equipped with sequential composition $s; t$ (defined when the output interface of s matches the input interface of t), with identities for each interface and with a monoidal tensor product $s \otimes t$ (associative, with unit and distributing over sequential composition); ii) observations have analogous structure $a; b$ and $a \otimes b$. Technically, we rely on configurations and observations that are taken in two monoidal categories that are freely generated from suitable signatures of constructors (i.e. the basic elements of which systems are composed of) and have the same underlying set of objects.

Tiles can be composed horizontally, in parallel, or vertically to generate larger rules. Horizontal composition $\alpha; \beta$ coordinates the evolution of the initial configuration of α with that of β, yielding the 'synchronization' of the two rewrites (see Fig. 13(ii)). Vertical composition is just the sequential composition of com-

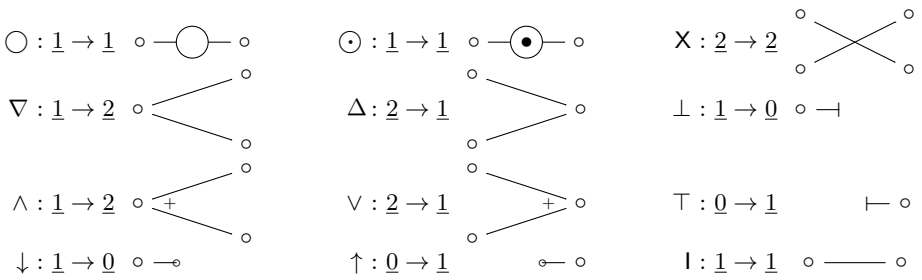


Fig. 14. Graphical representation of Petri calculus constants

putations (see Fig. 13(iii)). The parallel composition builds concurrent steps (see Fig. 13(iv)).

Tiles express the reactive behaviour of connectors in terms of $\langle \text{trigger}, \text{effect} \rangle$ pairs of labels. In this context, the usual notion of bisimilarity over the derived Labelled Transition System is called *tile bisimilarity*. Tile bisimilarity is a congruence (w.r.t. composition in series and parallel) when a simple tile format is met by basic tiles [17].

The wire calculus [24] builds on ideas from [20] to propose a process algebra whose distinctive features are: actions and processes come with an input/output arity typing (that depends on the ports they are using); independent concurrent systems are assembled together using a tensor product $(\cdot \otimes \cdot)$; ports are used instead of channels and communication is possible when ports are linked together by sequential composition $(\cdot;\cdot)$.

Roughly, we write $\vdash P : (n, m)$ for P with n input ports and m output ports and the usual action prefixes $a.P$ of process algebras are extended in the wire calculus by the simultaneous input of a trigger a and output of an effect b, written $\frac{a}{b}.P$, where a (resp. b) is a string of actions, one for each input port (resp. output port) of the process.

6.1 The Petri Calculus

Terms of the Petri Calculus are defined by the grammar in Fig. 15. It consists of the following constants plus parallel and sequential composition: the empty place $\bigcirc$, the full place $\odot$, the identity wire I, the twist (also swap, or symmetry) X, the duplicator (also sync) ∇ and its dual Δ, the mutex (also choice) $\wedge$ and its dual $\vee$, the hiding (also bang) $\bot$ and its dual $\top$, the inaction $\downarrow$ and its dual $\uparrow$. The graphical representation of Petri calculus constants are in Fig. 14.

Any term has a unique associated *sort* (also called *type*) (k, l) with $k, l \in \mathbb{N}$, that fixes the size k of the left (input) interface and the size l of the right (output) interface of P. The type of constants are as follows: $\bigcirc$, $\odot$, and I have type $(1, 1)$, $\mathsf{X} : (2, 2)$, ∇ and $\wedge$ have type $(1, 2)$ and their duals Δ and $\vee$ have type $(2, 1)$,

$$R ::= \ \bigcirc \mid \odot \mid \mathsf{I} \mid \mathsf{X} \mid \nabla \mid \Delta \mid \bot \mid \top \mid \wedge \mid \vee \mid \downarrow \mid \uparrow \mid R \otimes R \mid R; R$$

Fig. 15. Petri calculus grammar

$$\frac{R : (k,l) \quad R' : (m,n)}{R \otimes R' : (k+m, l+n)} \qquad \frac{R : (k,n) \quad R' : (n,l)}{R; R' : (k,l)}$$

Fig. 16. Sort inference rules

$$\overline{\bigcirc \xrightarrow[0]{1} \odot} \quad \overline{\odot \xrightarrow[1]{0} \bigcirc} \quad \overline{\odot \xrightarrow[1]{1} \odot} \quad \overline{\mathsf{I} \xrightarrow[1]{1} \mathsf{I}} \quad \overline{\nabla \xrightarrow[11]{1} \nabla} \quad \overline{\Delta \xrightarrow[1]{11} \Delta} \quad \overline{\bot \xrightarrow{1} \bot} \quad \overline{\top \xrightarrow[1]{} \top}$$

$$\overline{\mathsf{X} \xrightarrow[yx]{xy} \mathsf{X}} \quad \overline{\wedge \xrightarrow[x\overline{x}]{1} \wedge} \quad \overline{\vee \xrightarrow[1]{x\overline{x}} \vee} \quad \frac{R_1 \xrightarrow[\sigma]{\alpha} R_2 \quad R_1' \xrightarrow[\beta]{\sigma} R_2'}{R_1; R_1' \xrightarrow[\beta]{\alpha} R_2; R_2'} \quad \frac{R_1 \xrightarrow[\beta]{\alpha} R_2 \quad R_1' \xrightarrow[\sigma]{\rho} R_2'}{R_1 \otimes R_1' \xrightarrow[\beta\rho]{\alpha\sigma} R_2 \otimes R_2'} \quad \frac{R : (m,n)}{R \xrightarrow[0^n]{0^m} R}$$

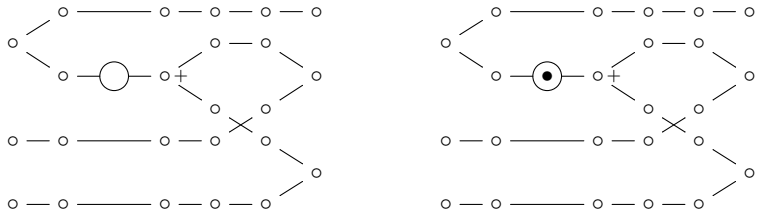
Fig. 17. Operational semantics for the Petri Calculus

$\bot$ and $\downarrow$ have type $(1,0)$ and their duals $\top$ and $\uparrow$ have type $(0,1)$. The sort inference rules for composed processes are in Fig. 16.

The operational semantics is defined by the tiles in Fig. 17, where $x, y \in \{0,1\}$ and we let $\overline{x} = 1 - x$. The labels $\alpha, \beta, \rho, \sigma$ of transitions are binary strings, all transitions are sort-preserving, and if $R \xrightarrow[\beta]{\alpha} R'$ with $R, R' : (n,m)$, then $|\alpha| = n$ and $|\beta| = m$. Notably, bisimilarity induced by such a transition system is a congruence.

Example 3. For example, let $P \stackrel{\mathsf{def}}{=} \nabla; (\mathsf{I} \otimes \odot)$ and $Q \stackrel{\mathsf{def}}{=} \nabla; (\mathsf{I} \otimes \bigcirc)$. It is immediate to check that P and Q have both sort $(1,2)$, in fact we have: $\nabla : (1,2)$, $\mathsf{I} \otimes \odot : (2,2)$ and $\mathsf{I} \otimes \bigcirc : (2,2)$. The only moves for P are $P \xrightarrow[00]{0} P$, $P \xrightarrow[01]{0} Q$ and $P \xrightarrow[11]{1} P$, while the only moves for Q are $Q \xrightarrow[00]{0} Q$ and $Q \xrightarrow[10]{1} P$. It is immediate to note that P is a term analogous to the net in Fig. 9(a).

A close correspondence between nets with boundaries and Petri calculus terms is established in [25], by providing mutual encodings with tight semantics correspondence. First, it is shown that any net $N : m \to n$ with initial marking X can be associated with a term $T_{N,X} : (m,n)$ that preserves and reflects the semantics of N. Conversely, for any term $T : (m,n)$ of the Petri calculus there exists a bisimilar net $N_T : m \to n$. Due to space limitation we omit details here and refer the interested reader to [25].



(a) Δ_{Off} (b) Δ_{On}

Fig. 18. Petri calculus term for the stateful selector

6.2 Compensation Pair in the Petri Calculus

We start by presenting the Petri calculus term equivalent to the stateful selector, which can be defined as follows:

$$\Delta_{\mathtt{off}} = (\nabla \oplus \mathsf{I} \oplus \mathsf{I}); (\mathsf{I} \oplus \bigcirc \oplus \mathsf{I} \oplus \mathsf{I}); (\mathsf{I} \oplus \wedge \oplus \mathsf{I} \oplus \mathsf{I}); (\mathsf{I} \oplus \mathsf{I} \oplus \mathsf{X} \oplus \mathsf{I}) : (\mathsf{I} \oplus \Delta \oplus \Delta)$$

Its graphical representation is in Fig. 18(a). It can be shown that the reductions of the connector coincide with the movements of the net with boundaries presented in Fig. 11. Moreover, Fig. 18(b) corresponds to the enabled state of the selector. We remark that the Petri calculus term equivalent to the net with boundaries for the stateful selector can be directly obtained by using the encoding defined in [25]. To handle all possible cases, the encoding uses a canonical representation of nets and, as a consequence, obtained terms can become more complex than necessary. For the sake of the simplicity, we prefer to present here a simpler term that is bisimilar to the one produced by the encoding.

Finally, the term representing the compensation pair $\mathtt{A\%B}$ can be defined analogously to the case of net with boundaries, i.e.,

$$[\![\mathtt{A\%B}]\!] = ([\![A]\!] \otimes \mathsf{I} \otimes \mathsf{I}); \Delta_{\mathsf{off}}; (\mathsf{I} \otimes [\![B]\!] \otimes \mathsf{I})$$

where $[\![A]\!]$ and $[\![B]\!]$ are Petri calculus terms with sort $(1, 1)$ describing the behaviour of components A and B, respectively.

7 Comparison

BIP and Reo are two prominent approaches for coordination that rely on (apparently) quite unrelated semantic models. In this section we link both models by taking advantage of several results appeared in the literature that formally state correspondences among the approaches presented in the previous sections.

BI(P) and Nets with boundaries The formal relation between BI(P) and nets with boundaries has been studied in [11]. Firstly, it is shown that any BI(P)

system can be mapped into a 1-safe Petri net that preserves computations. Intuitively, the places of the net are in one-to-one correspondence with the states of the components, while the transitions of the net represent the synchronized execution of the transitions of the components. In addition, [11] introduces a composition operation for BI(P) systems that enables the hierarchical definition of systems in which any BI(P) system can taken as a component of a more complex system. Then, this compositional version of BI(P) systems is used to define a compositional mapping of BI(P) systems into bisimilar nets with boundaries. Finally, it is shown that any net with boundaries without left interface can be encoded as a BI(P) system consisting on just one component. It is in this sense that BI(P) systems and nets with boundaries are retained equivalent.

Nets with boundaries and the Petri Calculus The technical contribution in [25] enlightens a tight semantics correspondence between these two approaches: it is shown that a Petri calculus process can be defined for each net such that the translation preserves and reflects operational semantics (and thus also bisimilarity). The second result provides the converse translation, from Petri calculus to nets, which requires some technical ingenuity.

Petri Calculus, Wires and Tiles The wire calculus [24] shares strong similarities with the tile model, in the sense that it has sequential and parallel compositions and exploits trigger-effect pairs labels as observations. However it is presented as a process algebra instead of via monoidal categories and it exploits a different kind of vertical composition. The usual action prefixes $a.P$ of process algebras are extended in the wire calculus by the simultaneous input of a trigger a and output of an effect b, written $\frac{a}{b}.P$, where a (resp. b) is a string of actions, one for each input port (resp. output port) of the process. The Petri calculus is a suitable instance of the wire calculus that roughly models circuit diagrams with one-place buffers and interfaces. An alternative characterization of the Petri calculus as tiles has been given in [12]

Algebra of stateless connectors, Tiles and Wires The algebra of stateless connectors in [10] can be regarded as a peculiar kind of tile model where all basic tiles have identical initial and final connectors, i.e. they are of the form $s \xrightarrow[b]{a} s$. In terms of the wire calculus, this means that only recursive processes of the form $\mathbf{rec}\, X.\frac{a}{b}.X$ are considered for composing larger networks of connectors.

Tiles and Reo Differently from the stateless connectors of [10], Reo connectors are stateful (in particular due to the asynchronous one-place buffer connector). Nevertheless, it has been shown in [2] that the two-colour semantics of Reo connectors can be recovered into the setting of the basic algebra of connectors and in the tile approach by adding a connector and a tile for the one-state buffer. It is worth mentioning that, in addition, the tile semantics of Reo connectors provides a description for full computations instead of just single steps (as considered in the original two-colour semantics) and makes evident the evolution of the connector state (particularly, whether buffers get full or become empty).

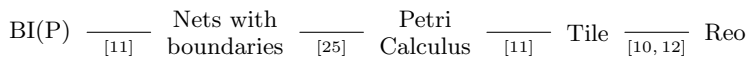


Fig. 19. Relation among the different models of connectors& buffers

The main results stating the correspondence among considered approaches are summarized in Fig. 19.

8 Conclusion and future work

One of the main limitations of the state-of-the-art theories of connectors is the lack of a reference paradigm for describing and analysing the information flow to be imposed over components for proper coordination. Such a paradigm would allow designers, analysts and programmers to rely on well-founded and standard concepts instead of using all kinds of heterogeneous mechanisms, like semaphores, monitors, message passing primitives, event notification, remote call, etc. Moreover, a reference paradigm would facilitate the comparison and evaluation of otherwise unrelated architectural approaches as well as the development of code libraries for distributed connectors.

Still, some kind of models can be more convenient than others for particular purposes, e.g., if modularity and mantainance is a key issue rather than efficient analysis or automatic synthesis out of requirements. So, we think that having links to move from one model to the other can be as important as having a referential model and, to some extent, it may be more practical.

Some interesting research avenues for future work are (i) the study of suitable extensions of BIP interaction model accounting for dynamically changing topologies of interactions; and (ii) the representation of priorities in approaches such as the algebra of connectors and the tile model. In the former direction, we are studying the possibility to define BIP components whose interfaces can be changed at run time and whose evolution can spawn new component instances and new interaction constraints. This would increase the expressive power of BIP. In the latter direction, we are studying the condition under which global priorities can be safely distributed among connectors. This would allow a modular, inconsistency-free assignment of priorities.

References

1. F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
2. F. Arbab, R. Bruni, D. Clarke, I. Lanese, and U. Montanari. Tiles for Reo. In *WADT*, volume 5486 of *LNCS*, pages 37–55. Springer, 2009.
3. F. Arbab and J. Rutten. A coinductive calculus of component connectors. In *WADT 2002*, volume 2755 of *LNCS*, pages 34–55. Springer, 2002.

4. C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in Reo by constraint automata. *Sci. Comput. Program*, 61(2):75–113, 2006.
5. P. Baldan, A. Corradini, H. Ehrig, and R. Heckel. Compositional semantics for open Petri nets based on deterministic processe. *Mathematical Structures in Computer Science*, 15(1):1–35, 2005.
6. A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006)*, pages 3–12. IEEE Computer Society, 2006.
7. S. Bliudze and J. Sifakis. The algebra of connectors - structuring interaction in BIP. *IEEE Trans. Computers*, 57(10):1315–1330, 2008.
8. S. Bliudze and J. Sifakis. Causal semantics for the algebra of connectors. *Formal Methods in System Design*, 36(2):167–194, 2010.
9. R. Bruni. *Tile Logic for Synchronized Rewriting of Concurrent Systems*. PhD thesis, Computer Science Department, University of Pisa, 1999.
10. R. Bruni, I. Lanese, and U. Montanari. A basic algebra of stateless connectors. *Theor. Comput. Sci.*, 366(1-2):98–120, 2006.
11. R. Bruni, H. Melgratti, and U. Montanari. Connector algebras, Petri nets, and BIP. In *Proceedings of the Ershov Informatics Conference PSI 2011*, volume 7162 of *LNCS*, pages 19–38. Springer, 2012.
12. R. Bruni, H. C. Melgratti, and U. Montanari. A connector algebra for p/t nets interactions. In *CONCUR*, volume 6901 of *LNCS*, pages 312–326. Springer, 2011.
13. R. Bruni and U. Montanari. Dynamic connectors for concurrency. *Theor. Comput. Sci.*, 281(1-2):131–176, 2002.
14. D. Clarke, D. Costa, and F. Arbab. Connector colouring I: Synchronisation and context dependency. *Sci. Comput. Program*, 66(3):205–225, 2007.
15. G. L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Inf. Comput.*, 156(1-2):173–235, 2000.
16. J. L. Fiadeiro and T. S. E. Maibaum. Categorical semantics of parallel program design. *Sci. Comput. Program.*, 28(2-3):111–138, 1997.
17. F. Gadducci and U. Montanari. The tile model. In *Proof, Language, and Interaction*, pages 133–166. The MIT Press, 2000.
18. H. Garcia-Molina and K. Salem. Sagas. In *Proceedings of the ACM Special Interest Group on Management of Data Annual Conference*, pages 249–259, 1987.
19. S.-S. T. Jongmans and F. Arbab. Overview of thirty semantic formalisms for Reo. *Scientific Annals of Computer Science*, 22(1):201–251, 2012.
20. P. Katis, N. Sabadini, and R. F. C. Walters. Representing place/transition nets in Span(Graph). In *AMAST*, volume 1349 of *LNCS*, pages 322–336. Springer, 1997.
21. N. Kokash and F. Arbab. Applying Reo to service coordination in long-running business transactions. In *SAC*, pages 1381–1382, 2009.
22. U. Montanari and F. Rossi. Graph rewriting, constraint solving and tiles for coordinating distributed systems. *Applied Categorical Structures*, 7(4):333–370, 1999.
23. C. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
24. P. Sobocinski. A non-interleaving process calculus for multi-party synchronisation. In *ICE*, volume 12 of *EPTCS*, pages 87–98, 2009.
25. P. Sobocinski. Representations of Petri net interactions. In *CONCUR*, volume 6269 of *LNCS*, pages 554–568. Springer, 2010.
26. P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of web services composition languages: The case of BPEL4WS. In *ER*, volume 2813 of *LNCS*, pages 200–215. Springer, 2003.