Chapter I.2

Reconfigurable and Software-Defined Networks of Connectors and Components*

Roberto Bruni[1], Ugo Montanari[1], and Matteo Sammartino[1]

Department of Computer Science, University of Pisa, Italy

Abstract. The diffusion of adaptive systems motivate the study of models of software entities whose interaction capabilities can evolve dynamically. In this paper we overview the contributions in the ASCENS project in the area of software defined networks and of reconfigurable connectors. In particular we highlight: (i) the definition of the Network-conscious pi-calculus and its use in the modeling and verification of the PASTRY protocol, and (ii) the mutual correspondence between different frameworks for defining networks of connectors together with two suitable enhancements for addressing dynamically changing systems.

Keywords: Network-conscious pi-calculus, PASTRY, overlay networks, coalgebraic semantics, HD-automata, BIP, Petri nets with boundaries, algebras of connectors, tile model, reconfigurable connectors, dynamic connectors

1 Introduction

One of the research strands of the ASCENS project is concerned with the study of resource-aware infrastructures and networking middleware modeled in terms of advanced components, glues and connectors which can support different levels of guarantees, reliability, dynamics and integration to heterogeneous components. The study includes the development of foundational models and architectures for network-aware systems with a high degree of dynamism in the communication topology between components. Formal models must allow a separation between the detailed behavior of components and their overall coordination. This paper surveys two approaches to coordination whose focus is on Software-Defined networks and on component-based design, respectively:

- For the former, we have proposed a network-aware extension of classical π-calculus [27], called NCPi, that allows for expressing the creation and the activation of connections, and whose semantics deals with the possible routing paths. We show that NCPi looks more adequate than traditional process calculi to describe Software-Defined and overlay networks, their routing

* This research was supported by the European project IP 257414 (ASCENS) and by the Italian MIUR Project CINA (PRIN 2010/11).

M. Wirsing et al. (eds.): Collective Autonomic Systems, LNCS 8998, pp. 73–106, 2015.

mechanisms, and to verify their properties. In particular, we show how NCPi can support the formalization and verification of the PASTRY distributed hash table system of cloud systems.
- For the latter, we have related some of the most notable theories for expressing network of connectors between components, by defining mutual embeddings that reduce the fragmentation in the body of knowledge and the different notions and terminologies involving connectors, and then we have proposed some enhancements to address reconfigurability and dynamism.

Structure of the paper. Section 2 introduces Software-Defined networks and the PASTRY protocol. Section 3 presents the syntax and semantics of NCPi, including an extension with features reflecting real-life routing protocols. Section 4 shows the formalization of the Pastry overlay networks, then used to prove that each message eventually gets to its destination.

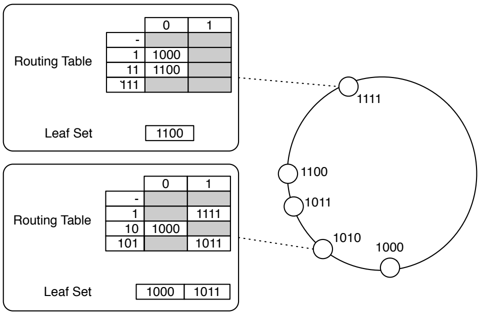
Section 5 explains the rationale for software architectures based on (networks of) connectors and components, and surveys some approaches from the literature. Section 6 presents the connection between algebras of connectors and nets with boundaries, a special flavor of Petri nets with interfaces. Section 7 recalls the BIP component framework and relates it with the models in Section 6. Section 8 introduces two enhanced models that allow a higher degree of dynamism.

2 Software-Defined and Overlay Networks

The trend in networking is going towards more “open” architectures, where the infrastructure can be manipulated in software. This trend started in the nineties, when OpenSig [15] and Active Networks [39] were presented, but neither gained wide acceptance due to security and performance problems. More recently, OpenFlow [26,32] or, more broadly, Software-Defined Networking has become the leading approach, supported by Google, Facebook, Microsoft and others. Software-Defined networks (SDNs) allow network administrators to control traffic via software installed on a centralized *controller*. This machine is connected to all switches in the network, and instructs them to install or uninstall forwarding rules and report traffic statistics.

Another important example of programmable infrastructures are peer-to-peer systems. They provide the networking substrate for the execution of distributed applications, such as *Distributed Hash Tables* (DHTs). In peer-to-peer systems, *peers* interact over an application-level *overlay network*, built on top of the physical one. An overlay network is highly dynamic, as peers can join and leave it at any time, and this causes continuous reconfigurations of its topology.

In particular, we consider PASTRY [35], employed in the Science Cloud case-study (see Chapter IV.3 [25]). PASTRY is a peer-to-peer architecture where peers and keys have *identifiers*, regarded as arranged in clockwise order on a *ring*. The main service provided by PASTRY is *routing by key*: given a key k, PASTRY delivers the message to the peer which is *responsible for* k, i.e. the one whose identifier is numerically closest to k than all other peers. Routing is implemented



Routing Table

	0	1
-	▨	▨
1	1000	▨
11	1100	▨
111	▨	▨

Leaf Set

1100

Routing Table

	0	1
-	▨	▨
1	▨	1111
10	1000	▨
101	▨	1011

Leaf Set

1000	1011

Fig. 1. PASTRY example system

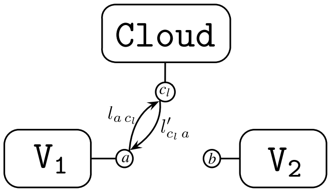
as follows. Each peer with identifier *id* maintains two data structures: a *routing table* and a *leaf-set.* The routing table contains peers that share a prefix with *id.* The leaf-set contains peers (*leaves*) with numerically closest smaller and larger identifiers, relative to *id.* Whenever *id* receives a message with target key k, it checks whether k belongs to the leaf-set range. If so, the message is forwarded to the leaf numerically closest to k (if it is not *id* itself). Otherwise, the routing table is used: the next hop is the peer sharing the longest prefix with k.

Example 1. An example system is in Figure 1, where identifiers are binary strings. Consider the peer with identifier 1010 and suppose 1100 is responsible for the key 1101. A message from 1010 with target key 1101 is routed as follows. Since 1101 does not belong to the interval $[1000, 1011]$ spanned by the leaf-set of 1010, the routing table is used: the longest prefix shared by 1010 and 1101 is 1, so the message is forwarded to 1111, the peer in the cell $(1, 1)$. Once 1111 receives the message, it discovers that 1101 is in its leaf-set range (the leaf-set has 1111 itself as upper bound, as no peer has larger identifier), so it forwards the message to the leaf closest to 1101, that is 1100.

3 Network Conscious π-Calculus (NCPi)

Traditional process calculi, such as π-calculus [27], seem inadequate to describe Software-Defined and overlay networks, their routing mechanisms, and to verify their properties. In fact, they abstract away from network details. Complex infrastructural elements, such as network links, could be described in terms of processes, and routing protocols in terms of consecutive step-by-step forwardings. However, end-to-end routing behavior could not be observed in a single transition, e.g. the path of a DHT lookup request through a peer-to-peer overlay. This information can be useful for the analysis of routing algorithms, e.g., to determine whether they are always able to construct a valid/optimal path for given source and destination.

In order to model network architectures in a more explicit way, in [30,36,31] we have introduced the *Network Conscious π-calculus* (NCPi), an extension of



$$\begin{aligned}
\mathtt{Cloud} &\overset{\text{def}}{=} c_l(x).c_l(y).(l_{xy})(\overline{c_l}l_{xy}.\mathtt{Cloud})\\
\mathtt{V}_1 &\overset{\text{def}}{=} \overline{a}a.\overline{a}b.a(l_{ab}).(\overline{a}c.\mathtt{V}_1{}' \mid \mathtt{L}(l_{ab}))\\
\mathtt{V}_2 &\overset{\text{def}}{=} b(x).\mathtt{V}_2{}'\\
\mathtt{L}(l_{xy}) &\overset{\text{def}}{=} l_{xy}.\mathtt{L}(l_{xy})\\
\mathtt{S} &\overset{\text{def}}{=} \mathtt{V}_1 \mid \mathtt{Cloud} \mid \mathtt{V}_2 \mid \mathtt{L}(l_{a\,c_l}) \mid \mathtt{L}(l'_{c_l\,a})
\end{aligned}$$

Fig. 2. Cloud example system

the π-calculus with a natural notion of network: nodes and links are regarded as computational resources that can be created, passed and used to transmit, so they are represented as names, following the π-calculus methodology.

3.1 Illustrative Example

To have a first look at the calculus, consider the system in Figure 2. It represents the network level of a cloud system, made of (virtual) machines and (virtual) links between them. *Site names* a, b, c_l represent network interfaces; *link names* $l_{a\,c_l}$ and $l'_{c_l\,a}$ represent directed links from a to c_l and viceversa, respectively. We have a *cloud manager* Cloud, capable of creating new links between virtual machines and granting access to them, and two virtual machines $\mathtt{V}_1$ and $\mathtt{V}_2$. We model a situation where the machine $\mathtt{V}_1$ wants to exchange data with $\mathtt{V}_2$, but no links exists between a and b, so $\mathtt{V}_1$ will ask Cloud to create such link.

The formal definition says that Cloud can receive two sites x and y at c_l, create a new link between them and emit it at c_l. The process $\mathtt{V}_1$ can send a and b from a, wait for a link at a and then become the parallel composition of two components: the first one can send c from a; the second one invokes the process L to activate the link l_{ab}. This activation is expressed as the *link prefix* $l_{xy}.-$ in the definition of L: when consumed, it spawns a *transportation service* over l_{xy}, which can be used to forward a datum from x to y. The link prefix expresses a single activation of the link, as input/output prefixes in the π-calculus express a single usage of their subject channel. The recursive definition of L is needed to model a persistent connection. The process $\mathtt{V}_2$ simply waits for a datum at b. Finally, the whole system S is the parallel composition of $\mathtt{V}_1$, $\mathtt{V}_2$, Cloud and two processes modeling a bidirectional persistent connection between $\mathtt{V}_1$ and Cloud.

As in the π-calculus, we have observations representing inputs, output and complete communications. However, since NCPi allows for remote communications, they all include the (possibly empty) sequence of links that are traversed in the communication. For instance, the process $\mathtt{V}_1$ can emit a at a as follows

$$\mathtt{V}_1 \xrightarrow{\bullet;\overline{a}a} \overline{a}b.a(l_{ab}).(\overline{a}c.\mathtt{V}_1{}' \mid \mathtt{L}(l_{ab}))$$

The label $\bullet;\overline{a}a$ is a zero-length (i.e. with empty sequence of links) *output path*, which can be seen as the π-calculus action $\overline{a}a$. The symbol $\bullet$ is syntactic sugar, indicating where the path starts. In general, there may be a list of links W between $\bullet$ and $\overline{a}a$: $\bullet; W; \overline{a}a$ means that a went through W before being emitted. The syntax also include *bound output paths* of the form $\bullet; W; a(b)$, representing the publication of a previously bound name b (its *extrusion*).

Symmetrically, `Cloud` can receive a at c_l

$$\mathtt{Cloud} \xrightarrow{c_l a;\bullet} c_l(y)(l_{ay})\overline{c_l}l_{ay}.\mathtt{Cloud}$$

where $c_l a; \bullet$ is an *input path*, analogous to the early π-calculus input action $c_l a$. Input paths always have length zero, as we only allow local receptions (this restriction will be dropped for the concurrent version of `NCPi`).

Next we introduce *service paths*, which have no counterpart in the π-calculus. A service path has the form $a; W; b$, where W is a sequence of links. It represents a transportation service that can be used to route a datum from a to b. For instance $a; l_{a\,c_l}; c_l$ is a service path from a to c_l over $l_{a\,c_l}$ and we have

$$\mathtt{L}(l_{a\,c_l}) \xrightarrow{a;l_{a\,c_l};c_l} \mathtt{L}(l_{a\,c_l}).$$

Finally, we have complete communication, denoted by a *complete path* $\bullet; W; \bullet$. Unlike the π-calculus τ-action, this observation is not silent, as the path W of the transmitted datum is observed; the datum itself remains unobservable. Another difference is that a complete path is usually produced by more than one synchronization, each one concatenating a compatible pair of paths. For instance, in order for $\mathtt{V}_1$ to communicate a to `Cloud`, there must be a first synchronization between $\mathtt{V}_1$ and $\mathtt{L}(l_{ab})$, causing $\bullet; \overline{a}a$ and $a; l_{a\,c_l}; c_l$ to be concatenated

$$\mathtt{V}_1 \,|\, \mathtt{L}(l_{a\,c_l}) \xrightarrow{\bullet;l_{a\,c_l};\overline{c_l}a} \ldots$$

Here the continuation is the parallel composition of those shown above, and $\bullet; l_{a\,c_l}; \overline{c_l}a$ is an output path where a is emitted at c_l after traversing $l_{a\,c_l}$. A complete path is produced by another, final synchronization:

$$\mathtt{V}_1 \,|\, \mathtt{L}(l_{a\,c_l}) \,|\, \mathtt{Cloud} \xrightarrow{\bullet;l_{a\,c_l};\bullet} \ldots$$

meaning that a complete communication over l_{ac_l} has happened.

Now we overview the steps the entire system `S` can perform:

1. **$\mathtt{V}_1$ communicates to `Cloud` the endpoints a and b of the link to be created:** it is observed as two consecutive occurrences of $\bullet; l_{ac_l}; \bullet$. The state of the system after these interactions is

$$a(l_{xy}).(\mathtt{L}(l_{xy}) \,|\, \overline{a}c.\mathtt{V_1}') \,|\, (l_{ab})(\overline{c_l}l_{ab}.\mathtt{Cloud}) \,|\, \mathtt{V}_2 \,|\, \mathtt{L}(l_{a\,c_l}) \,|\, \mathtt{L}(l'_{c_l\,a}) \ .$$

2. **$\overline{c_l}l_{ab}.\mathtt{Cloud}$ communicates l_{ab} to $a(l_{xy}).(\mathtt{L}(l_{xy}) \,|\, \overline{a}c.\mathtt{V_1}')$:** we first rearrange the processes using structural congruence

$$(l_{ab})(\ a(l_{xy}).(\mathtt{L}(l_{xy}) \,|\, \overline{a}c.\mathtt{V_1}') \,|\, \overline{c_l}l_{ab}.\mathtt{Cloud} \,|\, \mathtt{L}(l'_{c_l\,a})\ ) \,|\, \mathtt{V}_2 \,|\, \mathtt{L}(l_{a\,c_l}) \ .$$

Now the processes within the scope of l_{ab} can interact, and the resulting observation is $\bullet; l'_{c_l\, a}; \bullet$, with continuation

$$(l_{ab})(\mathtt{L}(l_{ab}) \,|\, \overline{a}c.\mathtt{V_1}' \,|\, \mathtt{Cloud} \,|\, \mathtt{V_2}) \,|\, \mathtt{L}(l_{a\, c_l}) \,|\, \mathtt{L}(l'_{c_l\, a}) \ .$$

3. $\overline{a}c.\mathtt{V_1}'$ **communicates** c **to** $\mathtt{V_2}$**:** in this case, despite l_{ab} is used for the transmission, only $\bullet; \bullet$ can be observed, because such link is restricted. This is analogous to the π-calculus τ action. The continuation is

$$(l_{ab})(\mathtt{L}(l_{ab}) \,|\, \mathtt{V_1}' \,|\, \mathtt{Cloud} \,|\, \mathtt{V_2}'[{}^{c}\!/\!_{x}]) \,|\, \mathtt{L}(l_{a\, c_l}) \,|\, \mathtt{L}(l'_{c_l\, a}) \ .$$

3.2 Syntax and Semantics

We assume an enumerable set of site names $\mathcal{S}$ (or just sites) and an enumerable set of link names $\mathcal{L}$ (or just links), equipped with two functions $s, t \colon \mathcal{L} \to \mathcal{S}$, telling source and target of each link. We denote by l_{ab} a link l such that $s(l) = a$ and $t(l) = b$. We write $\mathcal{L}_{ab}$ for the set of links of the form l_{ab} and $\mathcal{L}_a$ for the union of all $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$, for all b.

As shown in the previous section, the syntax of `NCPi` processes is an extension of the π-calculus one: prefixes can also express input/output of links, and we have a *link prefix* $l_{ab}.p$, meaning that this process can offer to the environment the service of transporting a datum from a to b through l and then continue as p.

The free names $\mathrm{fn}(p)$ describe the network available to p, in the form of a multigraph made of sites and links. They are defined as expected. For links, if l_{ab} is not the argument of a top-level binder, then $\mathrm{fn}(p)$ includes $\{a, b, l_{ab}\}$; otherwise, for instance in $a(l_{bc}).p$, only l_{bc} is bound, whereas its endpoints are free, namely $\mathrm{fn}(a(l_{bc}).p) := \{a, b, c\} \cup \mathrm{fn}(p) \smallsetminus \{l_{bc}\}$. The interesting cases are:

$$\mathrm{fn}(b(a).p) := \{b\} \cup (\mathrm{fn}(p) \smallsetminus (\{a\} \cup \mathcal{L}_a)) \qquad \mathrm{fn}((a)p) := \mathrm{fn}(p) \smallsetminus (\{a\} \cup \mathcal{L}_a)$$

where a free link in p having a as endpoint is considered bound in $(a)p$ and $b(a).p$. This intuitively means that a global link cannot have private endpoints. Given a name r, we shall write $r \,\#\, p$ to indicate that r is *fresh* w.r.t. p, i.e. $r \notin \mathrm{fn}(p)$; $N \,\#\, p$, with N a set of names, has the expected meaning.

Now we define *renamings* and their extensions to processes. Since names describe graphs, we require renaming to respect their structure, i.e. to be *graph homomorphisms*. In order to define the extension of renamings to processes, we need a notion of α-conversion that establishes how to avoid captures. For reasons that will become clear later, α-conversion can only be defined for processes where bound links are bound explicitly, and not as a side-effect of binding a site. We call such processes *well-formed*. For instance, $a(b).l_{bc}.p$ is not well-formed because l_{bc} is implicitly bound by $a(b)$.

Definition 1 (Well-formed process). *A `NCPi` process p is* well-formed *if for every subterm q: (i) $q = (a)p'$ implies* $\mathrm{fn}(q) = \mathrm{fn}(p') \smallsetminus \{a\}$*; (ii) $q = b(a).p'$ implies* $\mathrm{fn}(q) = \{b\} \cup \mathrm{fn}(p') \smallsetminus \{a\}$*;*

Table 1. Free names, bound names, objects and interaction sites of a path α

path α	fn	bn	obj	is
$a; W; b$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\{a, b\}$
$\bullet; W; \bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\varnothing$	$\varnothing$
$\bullet; W; \overline{a}r$	$\mathrm{n}(\alpha)$	$\varnothing$	$\mathrm{n}(r)$	$\{a\}$
$\bullet; W; a(r)$	$\mathrm{n}(\alpha) \smallsetminus \{r\}$	$\{r\}$	$\mathrm{n}(r) \smallsetminus \{r\}$	$\{a\}$
$ar; \bullet$	$\mathrm{n}(\alpha)$	$\varnothing$	$\mathrm{n}(r)$	$\{a\}$

Structural congruence axioms for well-formed processes contains the usual commutative monoid laws for $\mid$ and $+$, scope extension and unfolding for process definitions. The interesting case is α-conversion:

$$\begin{array}{lll} (a)p \equiv (a')p[{}^{a'}/_{a}] & b(a).p \equiv b(a').p[{}^{a'}/_{a}] & a' \# (a)p \\ (l_{ab})p \equiv (l'_{ab})p[{}^{l'_{ab}}/_{l_{ab}}] & c(l_{ab}).p \equiv c(l'_{ab}).p[{}^{l'_{ab}}/_{l_{ab}}] & l'_{ab} \# (l_{ab})p \end{array}$$

When α-converting $(a)p$, $[{}^{a'}/_{a}]$ is never applied to a link l_{ab}, since such link cannot be free in p by well-formedness. Indeed, $[{}^{a'}/_{a}]$ does not uniquely characterize a renaming if l_{ab} is free; if it is bound, i.e. if $(l_{ab})p'$ is a subprocess of p, then we simply have inductively $((l_{ab})p')[{}^{a'}/_{a}] \equiv (l'_{a'b})p'[{}^{l'_{a'b}}/_{l_{ab}}][{}^{a'}/_{a}]$, for any $l'_{a'b}$ fresh w.r.t. p. The axioms' side conditions guarantee preservation of well-formedness.

We now introduce the operational semantics. As mentioned, observations represent routing paths. We denote them by α. Table 1 introduces some notation for paths: the *interaction sites* of α, written $\mathrm{is}(\alpha)$, are those sites where the interaction with another process may happen, similarly to subjects of the π-calculus. We also have free names fn(), bound names bn() and objects obj() of α. Given a list of links W, we write W/r for W after removing each occurrence of $r \in \mathcal{L}$, and α/r for α with $/r$ applied to its list of links.

Definition 2 (NCPi transition system). *The* NCPi transition system *is the smallest transition system generated from the rules in Figure 3. We assume that structurally congruent processes have the same transitions.*

We briefly explain the rules. (OUT) and (IN) infer a zero-length path representing, respectively, the beginning and the end of a transmission. As in the early π-calculus, a renaming must be applied to the continuation in the free input case; if the input object is a site a, then we have a substitution between sites, which can be turned into a proper renaming by well-formedness. (LINK) infers a service path made of one link. (INT) infers an internal action, represented as a complete path where everything is unobservable. (RES) computes the paths of a process with an additional restriction (r) from those of the unrestricted process, provided that r is not already bound and is not an object or an interaction site. This side condition reflects that of the corresponding π-calculus rule. (OPEN) treats the case, excluded by (RES), when r is the object of a free output path: such path is turned into a bound output path, again rendering r unobservable when needed. (SUM) and (PAR) are as expected. (ROUTE), (COMP) and (COM) concatenate

$$(\textsc{out})\quad \overline{a}r.p \xrightarrow{\bullet;\overline{a}r} p \qquad\qquad (\textsc{open})\quad \frac{p \xrightarrow{\bullet;W;\overline{a}r} q}{(r)p \xrightarrow{\bullet;W/r;\overline{a}(r)} q}\ \ r{\neq}a$$

$$(\textsc{in})\quad a(r).p \xrightarrow{ar';\bullet} p[{}^{r'}\!/_{r}] \qquad\qquad (\textsc{par})\quad \frac{p_1 \xrightarrow{\alpha} q_1}{p_1 \mid p_2 \xrightarrow{\alpha} q_1 \mid p_2}\ \ \mathrm{bn}(\alpha) \# p_2$$

$$(\textsc{link})\quad l_{ab}.p \xrightarrow{a;l_{ab};b} p \qquad\qquad (\textsc{route})\quad \frac{p_1 \xrightarrow{\bullet;W;\overline{a}x} q_1 \qquad p_2 \xrightarrow{a;W';b} q_2}{p_1 \mid p_2 \xrightarrow{\bullet;W;W';\overline{b}x} q_1 \mid q_2}\ \ \mathrm{bn}(x) \# p_2$$

$$(\textsc{int})\quad \tau.p \xrightarrow{\bullet;\bullet} p \qquad\qquad (\textsc{comp})\quad \frac{p_1 \xrightarrow{a;W;b} q_1 \qquad p_2 \xrightarrow{b;W';c} q_2}{p_1 \mid p_2 \xrightarrow{a;W;W';c} q_1 \mid q_2}$$

$$(\textsc{sum})\quad \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad\qquad (\textsc{com})\quad \frac{p_1 \xrightarrow{\bullet;W;\overline{a}r} q_1 \qquad p_2 \xrightarrow{ar;\bullet} q_2}{p_1 \mid p_2 \xrightarrow{\bullet;W;\bullet} q_1 \mid q_2}$$

$$(\textsc{res})\quad \frac{p \xrightarrow{\alpha} q}{(r)p \xrightarrow{\alpha/r} (r)q}\ \ \begin{array}{c}\mathrm{bn}(\alpha)\\ r{\notin} \cup \mathrm{obj}(\alpha)\\ \cup\, \mathrm{is}(\alpha)\end{array}$$

Fig. 3. `NCPi` SOS rules

paths that meet at an interaction site: (ROUTE) extends an output path, provided that the transported name, whenever bound, is fresh w.r.t. the process that offers the transportation service; (COMP) composes two service paths; (COM) completes a communication. It is easy to see that the π-calculus is included in `NCPi`: we have just to forbid links in processes. The notion of behavioral equivalence is the following one, called *network conscious bisimulation.*

Definition 3 (Network conscious bisimulation). *A binary, symmetric and reflexive relation R is a* network conscious bisimulation *if* $(p, q) \in R$ *and* $p \xrightarrow{\alpha} p'$, *with* $\mathrm{bn}(\alpha) \# q$, *implies that there is* q' *such that* $q \xrightarrow{\alpha} q'$ *and* $(p', q') \in R$. *The bisimilarity is the largest such relation and is denoted by* $\sim^{NC}$.

We have the following closure result for $\sim^{NC}$.

Theorem 1. $\sim^{NC}$ *is closed under all syntactic operators except input prefix and parallel composition.*

Closure under input prefix not holding is expected. Surprisingly, also the parallel composition is problematic. This is because the semantics is transactional, in the sense that paths can involve more than one synchronization. As in the π-calculus, closure under input prefix is achieved by taking the greatest bisimulation closed under all renamings. Closure under parallel composition is discussed in [31].

3.3 Concurrent `NCPi` (κ`NCPi`)

We now present κ`NCPi`, an extension of `NCPi` with features reflecting real-life routing protocols. The most important one is that the semantics allows observing simultaneous actions taking place in the network, in the form of *multisets* of paths; this follows the intuition that processes should act in a truly distributed manner, without a central coordinator that imposes an interleaving order to their actions. The technical consequence is that bisimilarity becomes a *congruence.* Examples of real-life protocols exploiting such features can be found in [31], where the *Border Gateway Protocol* [41] is modeled, and in section 4.

The syntax of κ`NCPi` processes is the same as `NCPi`, with the following exceptions. Arguments of binders, which we denote by s, can be sites or expressions $l_{(ab)}$, meaning that l_{ab} is bound together with a and b. The intuitive meaning of $c(l_{(ab)}).p$ is an atomic, polyadic version of $c(a).c(b).c(l_{ab}).p$. The output primitive also specifies the *destination site*: $\overline{a}br.p$ can emit the datum r, having destination b, at a and continue as p. The definition of $\mathrm{fn}(p)$ for the new constructs is

$$\mathrm{fn}(\overline{a}br.p) := \{a,b\} \cup \mathrm{n}(r) \cup \mathrm{fn}(p) \qquad \mathrm{fn}(a(l_{(bc)}).p) := \{a\} \cup \mathrm{fn}(p) \smallsetminus (\{b,c\} \cup \mathcal{L}_b \cup \mathcal{L}_c)$$

Well-formed κ`NCPi` processes have to satisfy the requirements of Definition 1 plus the following one: $q = c(l_{(ab)}).p'$ implies $\mathrm{fn}(q) = \{c\} \cup \mathrm{fn}(p') \smallsetminus \{a, b, l_{ab}\}$. Structural congruence is minimal: we only have α-conversion and unfolding; other axioms are moved to observations or implemented through the rules.

Observations for the concurrent semantics, denoted by Λ, are multisets of paths, called *concurrent paths.* For the purpose of describing a more realistic network behavior, we equip paths α with some additional information:

- both input and output paths include a list of links; in the case of input paths, they are the links that can be traversed in order to reach the destination;
- there is a *bound input path* $ab(s); W; \bullet$, representing the reception of a bound name; this is needed because we introduce an explicit scope closure rule;
- paths always specify a destination site, namely b in $\bullet; W; \overline{a}br$, $abr; W; \bullet$ and $ab(s); W; \bullet$.

We remove extrusion paths: extrusions will be represented by concurrent paths, because we will allow many paths to extrude the same name simultaneously. Concurrent paths can be of the following forms:

- the *empty concurrent path* $\mathbf{1}$ indicates that no activity is performed;
- the *singleton concurrent path* α is a concurrent path made of a single path;
- the *union* $\Lambda_1 \,|\, \Lambda_2$ means that the paths in Λ_1 and Λ_2 are being traversed *at the same time*;
- the *extrusion restriction* $(r)\Lambda$ indicates that r is being extruded through one or more paths in Λ.

We impose some axioms on well-formed concurrent paths, telling that they are indeed multisets and that extrusion restrictions can be swapped and grouped at the outermost level.

We now introduce the transition system. Most of the rules are the expected concurrent extensions of Figure 3. The main difference is the synchronization mechanism. This is made of two steps:

(i) paths of parallel processes are collected through the following rule

$$(\textsc{par})\ \frac{p_1 \overset{\Lambda_1}{\Longrightarrow} q_1 \quad p_2 \overset{\Lambda_2}{\Longrightarrow} q_2}{p_1 \,|\, p_2 \xRightarrow{\Lambda_1 \,|\, \Lambda_2} q_1 \,|\, q_2}$$

where bound names in each concurrent path are require to be fresh w.r.t. the other process and its concurrent path;

(ii) other rules pick two compatible paths from the multiset produced by (i) and replace them with their concatenation, without modifying the source process; in other words, these rules synchronize two subprocesses of the source process. For instance, an output path and a service path with a common interaction site can be joined using the following rule, resulting in an extended output path

$$(\textsc{srv-out})\ \frac{p \xRightarrow{(R)\ (\bullet;W;\overline{a}br \,|\, a;W';c \,|\, \Theta)} q}{p \xRightarrow{(R)\ (\bullet;W;W';\overline{c}br \,|\, \Theta)} q}$$

where (R) is a sequence of restrictions and Θ is a concurrent path without extrusion restrictions (they have all been brought at the top level using scope extension).

The behavioral equivalence for κ`NCPi` processes is called *concurrent network conscious bisimilarity*, denoted $\sim_{\kappa}^{NC}$, and is an obvious extension of Definition 3: we require that processes can do the same concurrent paths.

Theorem 2. *$\sim_{\kappa}^{NC}$ is a congruence with respect to all κNCPi operators.*

This result allows us to equip the π-calculus with a compositional semantics: we can characterize π-calculus processes as κ`NCPi` processes via a syntactic restriction where links are forbidden and emission and destination sites in output prefixes coincide. SOS rules derive all possible paths, non-deterministically. In order to control path construction, e.g. according to a specific routing strategy, we can define a *forwarding predicate*

$$\varphi\colon \mathcal{L} \times \mathcal{S} \times \mathit{Proc} \to \{\mathtt{true}, \mathtt{false}\}$$

and then use it as an additional side condition for rules achieving step (ii) described above: $\varphi(l_{ab}, c, p)$ tells when a path of p, with destination c, can be extended with l_{ab}. In this way, for instance, we could exclude non-optimal links according to some metric (cost, latency, distance, and others). See [31] for a forwarding predicate modeling BGP.

3.4 Coalgebraic Semantics of `NCPi`

In [31] we have introduced a presheaf-based coalgebraic semantics for `NCPi`, in the style of [20]. The basic idea is having a model where we distinguish: (a) a domain of resources, (b) a domain of programs and a (c) domain of “maps” between resources and programs. In NCPi, resources of a process are its free sites and links, describing its communication network. Therefore, (a) is a category **G** of suitable graphs, representing networks, equipped with endofunctors that add new vertices and edges, modeling network resources allocation; (b) is **Set**, where some objects are regarded as sets of NCPi processes; (c) is the category of functors **G** → **Set** (*presheaves on* **C**), associating to each network the set of NCPi processes with such network.

The operational semantics, then, is modeled as a coalgebra with states in a presheaf, thus each state is decorated with its networks: this enables the explicit representation of network resources allocation along transitions. Unfortunately, we still have infinitely many states, because allocated resources may grow indefinitely, even if only a finite portion of them is actually accessible, e.g., in recursive processes performing extrusions. However, our presheaf of states is “well-behaved”, so, according to [16], it is always possible to deallocate the unused resources and an equivalent *History Dependent (HD) automaton* [28] can be derived from the NCPi coalgebra. HD-automata are automata with allocation and deallocation along transitions. They admit minimal, possibly finite state, representatives, where all bisimilar states are identified, which can be computed as shown and implemented in [19].

4 Formal Definition and Properties of the PASTRY Distributed Hash Table System

In this section we use κNCPi to model PASTRY overlay networks and DHTs. We will prove the correctness of our model by checking the following property, which says that each message eventually gets to its destination.

Property 1 (Routing convergence). The routing procedure always converges: given a message with target key k and a peer id, either id is responsible for k or it can forward the message to id' numerically closest to k than id.

4.1 Peer Model

The key idea is modeling identifiers as sites, and the routing table and the leaf-set of a peer as two collections of links $\mathcal{L}_{RT}$ and $\mathcal{L}_{LS}$, forming the overlay network. We denote by $a \boxdot b$ a link to b in a's routing table and by $a \boxminus b$ a link to b in a's leaf-set. A peer with identifier a is modeled as the process

$$\begin{aligned}
\mathtt{Peer}(a, \mathcal{L}_{RT}, \mathcal{L}_{LS}) &\stackrel{\text{def}}{=} (\mathcal{O}_{RT})(\mathcal{O}_{LS})\,\mathtt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) \\
&\qquad\qquad\qquad\qquad\quad \mid \mathtt{RT}(\mathcal{L}_{RT}, \mathcal{O}_{RT}) \mid \mathtt{LS}(\mathcal{L}_{LS}, \mathcal{O}_{LS}) \\
\mathtt{Control}(a, \mathcal{O}_{RT}, \mathcal{O}_{LS}) &\stackrel{\text{def}}{=} \mathtt{JoinH}(a) + \mathtt{Route}(\mathcal{O}_{RT}, \mathcal{O}_{LS})
\end{aligned}$$

Processes RT and LS allow other processes to query and modify routing table and leaf-set. These operations are called internally via the names in $\mathcal{O}_{RT}$ and $\mathcal{O}_{LS}$.

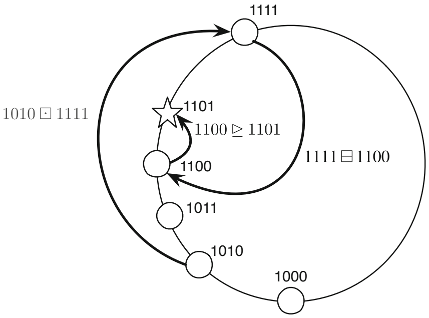


Fig. 4. Routing path from 1010 for the key 1101 in the system of Figure 1

The process `Control` implements the control logic of a peer. `JoinH` executes the distributed protocol for node joins: it updates the peer's own routing data structures and helps populating the joining peer's ones. In [36, Theorem 6.3.1] we show that the reconfiguration of the overlay network due to node joins preserves Property 1. The process `Route` simply activates transportation services over the peer's links. A PASTRY system is modeled as the parallel composition of peer processes. For the system in Figure 1 we have

$$\mathtt{Sys} \overset{\text{def}}{=} \mathtt{Peer}(1000) \,|\, \mathtt{Peer}(1010) \,|\, \mathtt{Peer}(1011) \,|\, \mathtt{Peer}(1100) \,|\, \mathtt{Peer}(1111)$$

4.2 DHT Model

Now we want to model routing behavior for a simple Distributed Hash Table, where observations are routing paths of DHT lookups. In order to do this, we introduce a new type of link: $a \trianglerighteq k$ means that the peer with identifier a is responsible for the key k. We can model a Distributed Hash Table over a PASTRY system made of peers $a_1, \ldots, a_n$ as follows. Suppose the DHT has m key-value pairs $\langle k_i, v_i \rangle$, and let a_{k_i} be the identifier of the peer responsible for k_i, i.e. the closest to k_i among $a_1, \ldots, a_n$.

$$\begin{aligned} \mathtt{DHT} &\overset{\text{def}}{=} \mathtt{Peer}(a_1) \,|\, \ldots \,|\, \mathtt{Peer}(a_n) \,|\, \mathtt{H} \\ \mathtt{H} &\overset{\text{def}}{=} \mathtt{Entry}(k_1, v_1, a_{k_1}) \,|\, \ldots \,|\, \mathtt{Entry}(k_m, v_m, a_{k_m}) \\ \mathtt{Entry}(k, v, a) &\overset{\text{def}}{=} a \trianglerighteq k \,|\, k(b).\overline{a}bv.\mathtt{Entry}(k, v, a) \end{aligned}$$

Here `H` represents the DHT content as the parallel composition of processes that handle the table's entries. The idea is implementing a DHT lookup request for a key k as a message with destination k, carrying the identifier b of the sender. Upon receiving this message, the handler $\mathtt{Entry}(k, v, a)$ for $\langle k, v \rangle$ replies to b with a message containing v.

In [36, Section 6.4] we provide an implementation of the the PASTRY routing strategy via a forwarding predicate, and we show that it yields paths satisfying

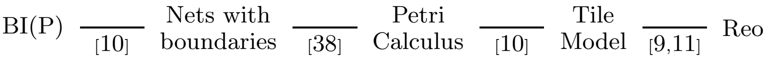


Fig. 5. Relation among the different models of connectors & buffers

Property 1. The consequence is that lookup requests always reach the correct peer ([36, Theorem 6.4.2]). As an example, we show how to compute a routing path in the system of Figure 1. For simplicity, let us consider a DHT with only one key-value pair $(1101, v)$ located at 1100:

$$\mathtt{H} \overset{\text{def}}{=} 1100 \trianglerighteq 1101 \,|\, 1101(a).\overline{1100}\, a\, v.\mathtt{H} \qquad \mathtt{DHT} \overset{\text{def}}{=} \mathtt{Sys} \,|\, \mathtt{H}$$

Consider the following process, representing a user application running at 1010

$$\mathtt{App} \overset{\text{def}}{=} \overline{1010}\ 1101\ 1010.1010(v').\mathtt{App}'(v')\ .$$

This sends a lookup request for the key 1101, receives the result and uses it for some computations. The routing steps for this request are depicted in Figure 4, and correspond to those of Example 1. The corresponding transition is

$$\mathtt{App} \,|\, \mathtt{DHT} \xRightarrow{\bullet;1010\boxdot 1111;1111\boxminus 1100;1100\trianglerighteq 1101;\bullet} 1010(v').\mathtt{App}'(v') \,|\, \mathtt{Sys} \,|\, \overline{1100}\, 1010\, v.\mathtt{H}$$

showing the whole routing path from 1010 to 1100.

5 Networks of Connectors and Components

Component-based design is a modular engineering practice that relies on the separation of concerns between coordination and computation. Component-based systems are built from loosely coupled computational entities, the *components*, whose interfaces comprise the number, kind and peculiarities of communication ports. The term *connector* denote entities that glue the interaction of components [33], by imposing suitable constraints on the allowed communications. The evolution of a network of components and connectors (just *network* for short) is 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 it imposes are the same at each round; *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 for network interaction.

Recent years have witnessed an increasing interest about a rigorous modeling of networks. We survey below, following the chronological order in which they were proposed, some formal approaches to the representation, composition and analysis of networks. Although the approaches we shall consider are quite different in spirit, the mutual correspondence results are summarized in Figure 5. All above approaches deal with systems that have static structures, i.e., systems in which the possible interactions among components are all defined at design time and remain unchanged during runtime. Nevertheless, when shifting to connectors for systems that adapt their behavior to changing environments, the situation is less well-understood. In fact, a general and uniform theory for dynamic connectors is still lacking. Some recent progresses are discussed in Section 8.

The algebra of connectors and the tile model. An algebra consisting of five kinds of basic stateless connectors (plus their duals) is presented in [9]. The connectors can be composed in series or in parallel. The operational, observational and denotational semantics of connectors are first formalized separately and then shown to coincide. Moreover, a complete normal-form axiomatization is defined. The *Petri calculus* in Section 6.2 enriches the algebra in [9] with one-place buffers.

The Tile Model [21,8] offers an operational and abstract semantic framework for concurrent systems [29,18,14] and also for suitable classes of connectors, of which the algebra of stateless connectors is just a particular instance. A *tile* $T : s \xrightarrow[\beta]{\alpha} t$ is a rewrite rule stating that the *initial configuration* s can evolve to the *final configuration* t producing the *effect* β; but the step is allowed only if the 'arguments' of s can evolve by providing the *trigger* α. Triggers and effects are called *observations*. Roughly, the semantics of component-based systems can be expressed via tiles when components and connectors are equipped with sequential composition $s;t$ and with a monoidal tensor product $s \otimes t$. Technically, we require that configurations and observations form two *monoidal categories* [24] with the same objects. Tiles express the reactive behaviour of connectors in terms of a Labelled Transition System (LTS) whose labels are pairs $\langle \text{trigger}, \text{effect} \rangle$. In this context, the usual notion of equivalence is called *tile bisimilarity*, which is a congruence (w.r.t. ; and $\otimes$) when a suitable rule format is met [21].

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 interaction patterns, including several 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 nodes. The semantics of Reo has been formalized in many ways, tile model included [2]. See [22] for a recent survey.

The BIP component framework. BIP [4] is a component framework for constructing systems by superposing three layers of modeling: 1) Behaviour, the lower level, representing the sequential computation of individual components; 2) Interaction, the middle layer, defining the handshaking mechanisms between these components; and 3) Priority, the top level, assigning a partial order of privileges to the admissible synchronizations. The lower layer consists of a set of atomic components with ports, modeled as automata whose arcs are labelled by sets of ports. The second layer consists of connectors that define suitable relations between ports. We name BI(P) the fragment of BIP without priorities (see

Section 7). In absence of priorities, the interaction layer admits the algebraic presentation given in [5]. 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 (cf. Chapter I.3 [17]).

Nets with boundaries and the wire calculus. Nets with boundaries [38] takes inspiration from the open nets of [3]. The idea is to extend Petri nets with interfaces that can be used by transitions to synchronize their firings with the environment. Nets with boundaries can be composed in series and in parallel and come equipped with a labelled transition system operational semantics. The correspondence between BI(P) and nets with boundaries is outlined in Section 7.

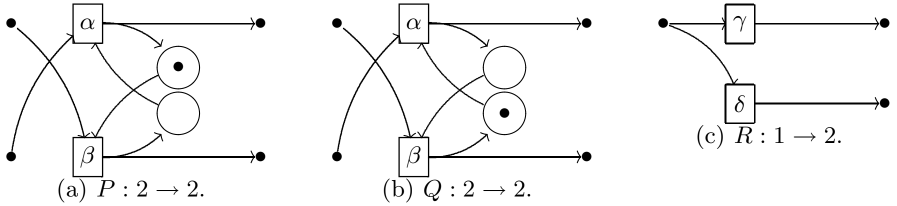
The wire calculus [37] shares strong similarities with the tile model, in the sense that it has sequential and parallel compositions and exploits trigger-effect pairs labels as observations. However, it is presented as a process algebra instead of via monoidal categories and it exploits a slightly different kind of vertical composition. Each process comes with an input/output arity typing, written $P : (n, m)$ for a process P with n input ports and m output ports. 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. In [38,11] a dialect of the wire calculus, called *Petri calculus*, has been used to give an exact characterization of a special class of (stateful) connectors that can be expressed as nets with boundaries. This result is outlined next.

6 Connector Algebras for Petri Nets

In this section we follow the contribution in [38,13]. Roughly, nets with boundaries are first introduced, that come equipped with sequential and parallel composition and with a labelled transition system semantics. Then, the *Petri calculus* is presented, that roughly models circuit diagrams with one-place buffers and interfaces. The first result shows that a Petri calculus process can be defined for each net such that the translation preserves and reflects operational semantics (and thus bisimilarity). The second result provides the converse translation, from Petri calculus to nets. The work in [38] has been recently improved in [11,13] by considering different firing policies for nets and exploiting the tile model to deal with Place/Transition Petri nets with boundaries.

6.1 Petri Nets with Boundaries

Petri nets [34] consist of *places* (i.e. resources types), which are repositories of *tokens* (i.e., resource instances), and *transitions* that remove and produce tokens.



(a) $P : 2 \to 2$.

(b) $Q : 2 \to 2$.

(c) $R : 1 \to 2$.

Fig. 6. Three nets with boundaries

Definition 4 (Net). *A* net *N is a 4-tuple $N = (S, T, {}^\circ -, -^\circ)$ where S is the (nonempty) set of places, T is the set of transitions, (with $S \cap T = \varnothing$), and the functions ${}^\circ -, -^\circ : T \to 2^S$ assign finite sets of places, called respectively source and target, to each transition.*

Transitions t, u are *independent* when ${}^\circ t \cap {}^\circ u = t^\circ \cap u^\circ = \varnothing$. A set U of transitions is 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 $N = (S, T, {}^\circ -, -^\circ)$ be a net, $X, Y \subseteq S$ and $t \in T$. Write: $(N, X) \to_{\{t\}} (N, Y) \overset{\mathsf{def}}{=} {}^\circ t \subseteq X \wedge t^\circ \subseteq Y \wedge X \backslash {}^\circ t = Y \backslash t^\circ$.*

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

$$(N, X) \to_U (N, Y) \overset{\mathsf{def}}{=} {}^\circ U \subseteq X \wedge U^\circ \subseteq Y \wedge X \backslash {}^\circ U = Y \backslash U^\circ.$$

Note that, for any $X \subseteq S$, $(N, X) \to_\varnothing (N, X)$. A pair (N, X) (or just X when N is obvious from the context) is called a *marking*. Sometimes nets comes equipped with an *initial marking* X_0, representing the initial state of the system.

In the following, given $n \in \mathbb{N}$, we let $\underline{n} \overset{\mathsf{def}}{=} \{0, 1, \ldots, n-1\}$.

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 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 2. Figure 6 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 post-set relations. The left (resp., right) interface is depicted by points situated on the left (resp., on the right).

Note that for any $k \in \mathbb{N}$, there is a bijection $\ulcorner \_ \urcorner : 2^{\underline{k}} \to \{0, 1\}^k$ with $\ulcorner U \urcorner_i = 1$ if $i \in U$ and $\ulcorner U \urcorner_i = 0$ otherwise. This fact is exploited to define the semantics of a net with boundaries $N : m \to n$ as a double-labelled transition system, where transition labels are pairs of strings in $\{0, 1\}^m \times \{0, 1\}^n$, representing the tokens requested on the left/right boundary by the firing.

Definition 7 (Semantics). *Let $N : n \to n$ be a net and $X, Y \subseteq S$. We write $(N, X) \xrightarrow[\beta]{\alpha} (N, Y)$ iff there exists an independent $U \subseteq T$ such that $(N, X) \to_U (N, Y)$, with $\alpha = \ulcorner {}^\bullet U \urcorner$ and $\beta = \ulcorner U^\bullet \urcorner$.*

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 the disjoint union of the initial markings of N and M.

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 synchronization is a pair (U, V) with $U \subseteq T_N$ and $V \subseteq T_M$ independent sets of transitions such that: (1) $U \cup V \neq \varnothing$ and (2) $U^\bullet = {}^\bullet V$.

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

The sequential composition $N; M : m \to k$ is defined as the net with boundaries $(S_N \uplus S_M, T_{N;M}, {}^\circ-_{N;M}, -^\circ_{N;M}, {}^\bullet-_{N;M}, -^\bullet_{N;M})$, where:

- $T_{N;M} \stackrel{\mathsf{def}}{=} \{(U, V) | U \subseteq T_N, V \subseteq T_M, (U, V)$ a minimal synchronisation$\}$,
- ${}^\circ(U, V)_{N;M} = {}^\circ(U)_N \uplus {}^\circ(V)_M$ 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. Of course, only minimal synchronizations are selected. The initial marking of $N; M$ is the disjoint union of X_{0N} and X_{0M}.

Example 3. Consider the nets $P : 2 \to 2$ and $R : 1 \to 2$ in Figure 6. Then, the composed net $P; (R \otimes R) : 2 \to 4$ is shown in Figure 7.

6.2 Petri Calculus

In this section we introduce an algebra of connectors that roughly enriches the algebra of stateless connectors from [9] with one-place buffers along [2]. We call it *Petri calculus* after [38]. The algebra of stateless connectors in [9] can be regarded as a fragment of the Petri calculus where all 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.

Terms of the Petri Calculus are defined by the grammar:

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

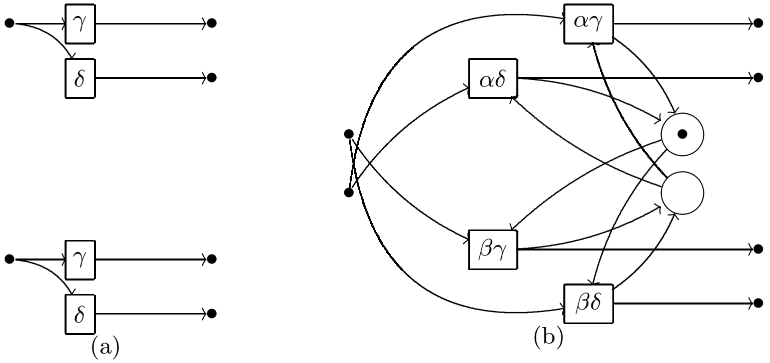


Fig. 7. Nets $R \otimes R$ (a) and $P; (R \otimes R)$ (b)

$\bigcirc : (1,1)$ $\quad$ $\odot : (1,1)$ $\quad$ $\mathsf{I} : (1,1)$ $\quad$ $\mathsf{X} : (2,2)$ $\quad$ $\nabla : (1,2)$ $\quad$ $\Delta : (2,1)$

$\bot : (1,0)$ $\quad$ $\top : (0,1)$ $\quad$ $\wedge : (1,2)$ $\quad$ $\vee : (2,1)$ $\quad$ $\downarrow : (1,0)$ $\quad$ $\uparrow : (0,1)$

$$\frac{R_1 : (k,l) \quad R_2 : (m,n)}{R_1 \otimes R_2 : (k+m, l+n)} \qquad \frac{R_1 : (k,n) \quad R_2 : (n,l)}{R_1 \otimes R_2 : (k,l)}$$

Fig. 8. Sort inference rules

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

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

Fig. 9. Operational semantics for the Petri Calculus

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 X, the duplicator ∇ and its dual Δ, the mutex $\wedge$ and its dual $\vee$, the hiding $\bot$ and its dual $\top$, the inaction $\downarrow$ and its dual $\uparrow$. Any term R has a unique associated *sort* (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 (see Fig. 8).

The operational semantics is defined by the rules in Fig. 9, 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, the induced bisimilarity is a congruence.

Example 4. For example, let $R_1 \stackrel{\mathsf{def}}{=} (\nabla \otimes \nabla); (\odot \otimes \mathsf{X} \otimes \bigcirc); (\Delta \otimes \Delta); \mathsf{X}$ and $R_2 \stackrel{\mathsf{def}}{=} (\nabla \otimes \nabla); (\bigcirc \otimes \mathsf{X} \otimes \odot); (\Delta \otimes \Delta); \mathsf{X}$. It is immediate to check that both $R_1 : (2,2)$

and $R_2 : (2,2)$, in fact: $\nabla \otimes \nabla : (2,4)$, $\odot \otimes \mathsf{X} \otimes \bigcirc : (4,4)$, $\bigcirc \otimes \mathsf{X} \otimes \odot : (4,4)$, $\Delta \otimes \Delta$: $(4,2)$, and $\mathsf{X} : (2,2)$. The only moves for R_1 are $R_1 \xrightarrow[00]{00} R_1$ and $R_1 \xrightarrow[10]{01} R_2$ while for R_2 are $R_2 \xrightarrow[00]{00} R_2$ and $R_2 \xrightarrow[01]{10} R_1$. It is immediate to note that R_1 and R_2 are terms analogous to the nets in Fig. 6 and that R_1 is bisimilar to $\mathsf{X}; R_2; \mathsf{X}$.

A close correspondence between nets with boundaries and Petri calculus terms is established in [13], 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 $R_{N,X} : (m,n)$ that preserves and reflects the semantics of N. Conversely, for any term $R : (m,n)$ of the Petri calculus there exists a bisimilar net $N_R : m \to n$. We refer the interested reader to [13].

7 From BI(P) to Petri Nets and Vice Versa

This section surveys the correspondence between BI(P) systems and nets (and with the Petri calculus, by transitivity) as studied in [10]. First, a composition operation for BI(P) systems is introduced that enables the structured definition of larger systems. 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. Then, this compositional version of BI(P) systems is used to define a compositional mapping of BI(P) systems to bisimilar nets with boundaries (see Section 7.2). Finally, it is shown that any net with boundaries with vacuous left interface can be encoded as a BI(P) system (see Section 7.3).

7.1 BI(P): BIP Without Priorities

This section reports on the formal definition of BIP as presented in [6]. Since we disregard priorities, we call BI(P) the framework presented here.

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 an interaction $a \subseteq P$ over the set of ports $P_i \subseteq P$, i.e., $a \downarrow_{P_i} = a \cap P_i$. Projection extends to sets of interactions by $\gamma \downarrow_P = \{a \downarrow_P \mid a \in \gamma \ \wedge \ a \downarrow_P \neq \varnothing\}$.

Definition 8 (Component). *A* component $B = (Q, P, \to)$ *is a triple where* Q *is a set of states,* P *is a set of ports, and* $\to \subseteq Q \times 2^P \times Q$ *is the set of transitions.*

As usual, we write $q \xrightarrow{a} q'$ to denote the transition $(q, a, q') \in \to$. We let q_a_q' be the *name* of the transition $q \xrightarrow{a} q'$. Given a transition $t = q\_a\_q'$, we let ${}^\circ t$, t° and $\lambda(t)$ denote respectively its source q, its target q' and its label a. An interaction a is enabled in q, denoted $q \xrightarrow{a}$, iff there exists q' s.t. $q \xrightarrow{a} q'$. By abusing notation, we will also write $q \xrightarrow{\varnothing} q$ for any q.

Definition 9 (BI(P) system). *A* BI(P) system $BS = \gamma(B_1, \ldots, B_n)$ *is the composition of a finite set* $\{B_i\}_{i=1}^n$ *of transitions systems* $B_i = (Q_i, P_i, \to_i)$

such that their sets of ports are pairwise disjoint, i.e., $P_i \cap P_j = \varnothing$ *for* $i \neq j$ *parameterized 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* BS*, written* $\iota(BS)$*.*

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

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

Definition 10 (Coherent interaction extension). *A set of interactions* γ' *is a coherent extension of* γ *over the set of ports* P*, written* $\gamma \rhd_P \gamma'$*, iff* $\gamma' \downarrow_P \subseteq \gamma$*.*

The idea underlying coherent extension is that the extended set of interactions γ' does not allow more interactions (in P) than those specified by γ.

Definition 11 (cBI(P) system). *A composite BI(P) system* C*, cBI(P) for short, is either a BI(P) system* $\gamma(B_1, \ldots, B_n)$ *or a composition* $\gamma(C_1, \ldots, C_n)$ *where* $\{C_i = \gamma_i(C_{i,1}, \ldots, C_{i,n_i})\}_{i=1}^n$ *is a family of cBI(P) systems such that their sets of underlying ports are pairwise disjoint, i.e.,* $\iota(C_i) \cap \iota(C_j) = \varnothing$ *for* $i \neq j$*, and* γ *a set of interactions over* $\uplus_{i=1}^n \iota(C_i)$ *s.t.* $\gamma_i \rhd_{\iota(C_i)} \gamma$*.*

The semantics of cBI(P) systems is defined analogously to that of BI(P) systems by viewing each subsystem as a component. Next result states that any BI(P) system can be seen as a cBI(P) system of exactly two components. We will use this property when defining the compositional encoding of BI(P) systems.

Lemma 1. *Let* $BS = \gamma(B_1, \ldots, B_n)$ *be a BI(P) system. Then, for any* $i < n$*,* BS *is bisimilar to the cBI(P) system* $C = \gamma(\gamma \downarrow_{P_{1..i}} (B_1, \ldots, B_i), \gamma \downarrow_{P \setminus P_{1..i}} (B_{i+1}, \ldots, B_n))$ *where* $P = \iota(BS)$ *and* $P_{1..i} = \uplus_{j \leq i} \iota(B_j)$*.*

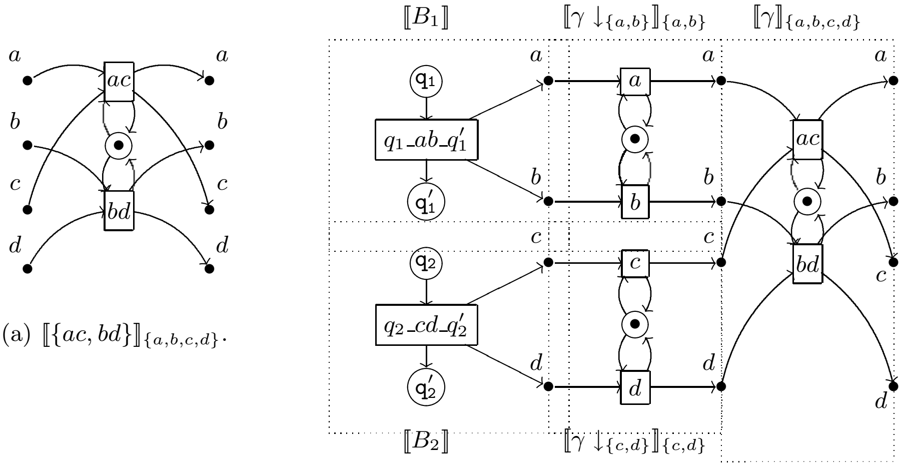
7.2 Structural Mapping from BI(P) to Nets with Boundaries

Given a finite set S with $k = |S|$, we use w_S to denote an injective function $w_S : S \rightarrow \underline{k}$ that orders elements of S. By abuse of notation, we write also w_S to denote its extension $w_S : 2^S \rightarrow 2^{\underline{k}}$.

Definition 12. *Let* $B = (Q, P, \rightarrow)$ *be a transition system. The corresponding net with boundaries* $[\![B]\!] : 0 \rightarrow |P|$ *is* $[\![B]\!] = (Q, T, {}^\circ -, -^\circ, {}^\bullet -, -^\bullet)$ *where:*

- $T = \{q\_a\_q' \mid q \xrightarrow{a} q'\}$*.*
- ${}^\circ(q\_a\_q') = \{q\}$ *and* $(q\_a\_q')^\circ = \{q'\}$*.*
- ${}^\bullet(q\_a\_q') = \varnothing$ *and* $(q\_a\_q')^\bullet = w_P(a)$*.*

Lemma 2. *Let* $B = (Q, P, \rightarrow)$ *be a transition system. Then,* $q \xrightarrow{a} q'$ *if and only if* $([\![B]\!], \{q\}) \xrightarrow[\ulcorner w_P(a) \urcorner]{} ([\![B]\!], \{q'\})$*.*


a
b
c
d
ac
bd
$[\![B_1]\!]$
$[\![\gamma\downarrow_{\{a,b\}}]\!]_{\{a,b\}}$
$[\![\gamma]\!]_{\{a,b,c,d\}}$
q1
$q_1\_ab\_q_1'$
q'1
q2
$q_2\_cd\_q_2'$
q'2
$[\![B_2]\!]$
$[\![\gamma\downarrow_{\{c,d\}}]\!]_{\{c,d\}}$

(a) $[\![\{ac, bd\}]\!]_{\{a,b,c,d\}}$.

(b) $[\![\gamma(B_1, B_2)]\!]$ with $\gamma = \{ac, bd\}$.

Fig. 10. Compositional encoding

Next definition introduces the encoding of a set of synchronizations glueing components as a marked net with boundaries.

Definition 13. *Let γ be a set of synchronizations over P. The corresponding marked net with boundaries $[\![\gamma]\!]_P : |P| \to |P|$ is $(\{P_\gamma\}, \gamma, {}^\circ-, -^\circ, {}^\bullet-, -^\bullet, \{P_\gamma\})$ with: ${}^\circ a = a^\circ = \{P_\gamma\}$ and ${}^\bullet a = a^\bullet = w_P(a)$.*

Note that the place P_γ guarantees that all interactions are mutually exclusive.

Example 5. Consider the set of interactions $\gamma = \{ac, bd\}$ and assume $w_{\{a,b,c,d\}}$ coincides with alphabetical order. The corresponding net is shown in Fig. 10(a).

Lemma 3. $[\![\gamma]\!]_P \xrightarrow[\ulcorner w_P(a)\urcorner]{\ulcorner w_P(a)\urcorner} [\![\gamma]\!]_P$ *iff $a \in \gamma$.*

Next definition introduces the compositional encoding of BI(P) systems.

Definition 14. *Let $C = \gamma(C_1, \ldots, C_n)$ be a cBI(P) system. The net with boundaries $[\![C]\!] : 0 \to |P|$ with $P = \iota(C)$ is recursively defined as*

$$\begin{aligned}
[\![\gamma(C_1)]\!] &= [\![C_1]\!]; [\![\gamma]\!]_P \\
[\![\gamma(C_1, \ldots, C_n)]\!] &= (\gamma \downarrow_{\iota(C_1)} [\![C_1]\!] \otimes [\![\gamma \downarrow_{P\setminus\iota(C_1)} (C_2, \ldots, C_n)]\!]); [\![\gamma]\!]_P
\end{aligned}$$

Example 6. Consider the BI(P) system $\{ac, bd\}(B_1, B_2)$ where B_1 has just one transition $q_1 \xrightarrow{ab} q_1'$ and B_2 has only $q_2 \xrightarrow{cd} q_2'$. The encoded net is in Figure 10(b). Note the necessity of considering all transitions in the encoding of $\{ac, bd\}$ to be mutual exclusive. Otherwise, the encoded form will also allow behaviors like $([\![B]\!], \{q_1, q_2\}) \xrightarrow[\ulcorner abcd\urcorner]{} ([\![B]\!], \{q_1', q_2'\})$, where $abcd \notin \{ac, bd\}$.

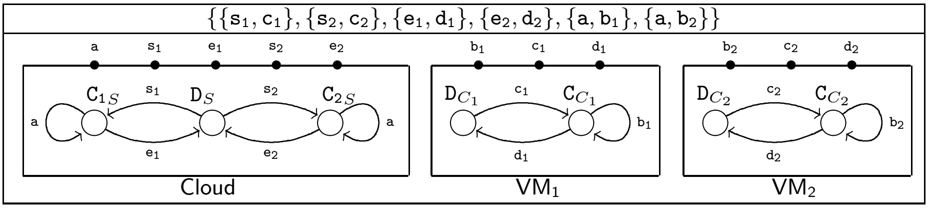


Fig. 11. A simple BI(P) system

Theorem 3. *Let* $C = \gamma(C_1, \ldots, C_n)$ *be a cBI(P) system. Then,* $(q_1, \ldots, q_n) \xrightarrow{a}_\gamma (q'_1, \ldots, q'_n)$ *iff* $([\![C]\!], \{q_1, \ldots, q_n\}) \xrightarrow[\ulcorner w_P(a) \urcorner]{} ([\![C]\!], \{q'_1, \ldots, q'_n\})$ *with* $P = \iota(C)$.

7.3 Encoding Nets with Boundaries into BI(P)

This section shows that any net with boundaries without left interface can be seen as a BI(P) system consisting on just one component. The correspondence is stated by showing that there exists a straightforward encoding that maps states and transitions of the net to states and transitions of the unique component.

Definition 15. *Let* $N : 0 \to n$ *with* $N = (S, T, {}^\circ-, -^\circ, {}^\bullet-, -^\bullet)$ *be a net with boundaries. Then, the corresponding BI(P) system* $BS_N = \gamma(B)$ *is defined as follows:* $\gamma = 2^{\underline{n}}$ *and* $B = (2^S, 2^{\underline{n}}, \to)$ *with* $\to = \{X \xrightarrow{a} Y \mid (N, X) \xrightarrow[\ulcorner a \urcorner]{} (N, Y)\}$.

Note that $N : 0 \to n$ corresponds to a component that has $2^{\underline{n}}$ ports, i.e., one port for any possible combination of the ports on the interface.

8 Reconfigurable and Dynamic BIP

In order to contribute to the development of a general theory for dynamic connectors, in this section we present two other extensions of the BI(P) framework with different degrees of "dynamism" that allow enhanced conciseness, modularity and expressiveness. A *reconfigurable* BI(P) system allows for the dynamic modification of interactions among components. A *dynamic* BI(P) system supports the runtime creation / elimination of ports and components.

Example 7. Consider the BI(P) system shown in Fig. 11, which contains a cloud manager component Cloud that interacts with two virtual machines $\mathtt{VM_1}$ and $\mathtt{VM_2}$. The Cloud starts a connection with $\mathtt{VM_i}$ via the interaction $\mathtt{s_i c_i}$. After the session is started, the manager and the clients can interact on $\mathtt{ab_i}$. The session ends when $\mathtt{e_i d_i}$ is performed. Note that the manager has dedicated ports $(\mathtt{s_i}, \mathtt{e_i})$ for handling the connections of different machines. Next, we introduce two enhancements that allows for a more compact description of this system.

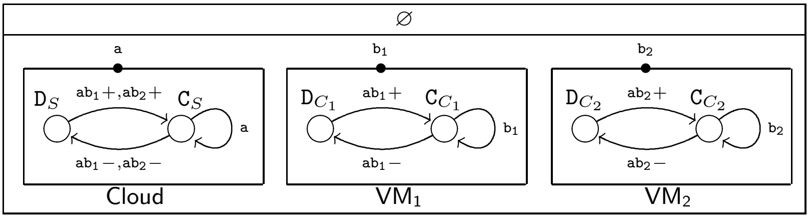


Fig. 12. A simple reconfigurable BI(P) system

8.1 Reconfigurable BI(P)

Our first extension is concerned with the possibility of enabling and disabling specific interactions dynamically. An interaction a can be enabled / disabled when all components involved in the interaction a agree to do so. After a is enabled, it can be used as an ordinary interaction until it gets disabled.

Our first result proves that any reconfigurable BI(P) system is equivalent to an ordinary BI(P) system where a "controller" component is introduced for each interaction that can be added or removed at run-time. Thus, reconfigurable BI(P) only provides a more compact representation of ordinary systems, while ordinary BI(P) representations may require an exponential blow up in the number of controllers (interactions are subsets of ports). The crux of the proof is the fact that the set of controller components can be defined statically.

Transitions in a reconfigurable BI(P) component have a decoration ρ that can be either (i) ϵ for ordinary interactions, (ii) $+$ to add a new interaction, and (iii) $-$ to remove an interaction.

Definition 16 (Reconfigurable component). *Let $\mathcal{P}$ be a set of ports. A* reconfigurable component *$R = (Q, P, \twoheadrightarrow)$ is a transition system where Q is a set of states, $P \subset \mathcal{P}$ is a finite set of ports, and $\twoheadrightarrow \subseteq Q \times 2^{\mathcal{P}} \times \{+, -, \epsilon\} \times Q$ is the set of labelled transitions such that $(q, a, \epsilon, q') \in \twoheadrightarrow$ implies $a \in 2^P$ and $(q, a, \rho, q') \in \twoheadrightarrow$ with $\rho \in \{+, -\}$ implies $a \cap P \neq \varnothing$.*

We write $q \xrightarrow{a\rho}\!\!\!\!\twoheadrightarrow q'$ for $(q, a, \rho, q') \in \twoheadrightarrow$. We say that a is enabled in q, denoted $q \overset{a}{\twoheadrightarrow}$, iff there exists q' s.t. $q \xrightarrow{a\epsilon}\!\!\!\!\twoheadrightarrow q'$. We assume that for all q, q' it holds $q \xrightarrow{\varnothing\epsilon}\!\!\!\!\twoheadrightarrow q'$ iff $q = q'$. Given a set of ports P, we write $a\#P$ if $a \cap P = \varnothing$.

Definition 17 (Reconfigurable BI(P) system). *A reconfigurable* BI(P) *system $RS = \gamma(R_1, \ldots, R_n)$ is the composition of a finite set $\{R_i\}_{i=1}^n$ of reconfigurable components $R_i = (Q_i, P_i, \twoheadrightarrow_i)$ 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 $\gamma \subset 2^{\mathcal{P}}$. We call $P = \biguplus_{i=1}^n P_i$ the underlying set of ports of RS, written $\iota(RS)$.*

Example 8. The scenario in Example 7 can be modeled as the reconfigurable BI(P) system in Fig. 12, where for simplicity we represented multiple transitions

$$\frac{a \in \gamma \quad \forall i \in 1..n : q_i \xtwoheadrightarrow{a\downarrow_{P_i}\epsilon} q_i'}{\gamma(q_1,\ldots,q_n) \xtwoheadrightarrow{a} \gamma(q_1',\ldots,q_n')}\,[\textsc{int}]$$

$$\frac{a \in 2^P \smallsetminus \gamma \quad \neg(a\#P_i) \implies q_i \xtwoheadrightarrow{a+} q_i' \quad (a\#P_i) \implies q_i' = q_i \quad \gamma' = \gamma \cup \{a\}}{\gamma(q_1,\ldots,q_n) \xtwoheadrightarrow{a} \gamma'(q_1',\ldots,q_n')}\,[\textsc{add}]$$

$$\frac{a \in \gamma \quad \neg(a\#P_i) \implies q_i \xtwoheadrightarrow{a-} q_i' \quad (a\#P_i) \implies q_i' = q_i \quad \gamma' = \gamma \smallsetminus \{a\}}{\gamma(q_1,\ldots,q_n) \xtwoheadrightarrow{a} \gamma'(q_1',\ldots,q_n')}\,[\textsc{del}]$$

Fig. 13. Operational semantics of reconfigurable BI(P) systems

with the same source and target (but different labels) with a single arc with multiple labels (e.g., $\mathtt{ab_1}+, \mathtt{ab_2}+$). Now, the transitions $\mathtt{ab_i}+$ and $\mathtt{ab_i}-$ respectively allow for the dynamic enabling / disabling of the interaction $\mathtt{ab_i}$.

The semantics of a reconfigurable BI(P) system $RS = \gamma(R_1,\ldots,R_n)$ with $\iota(RS) = P$ and $\gamma \subseteq 2^P$ is given by the transition system $(Q, \twoheadrightarrow)$ where

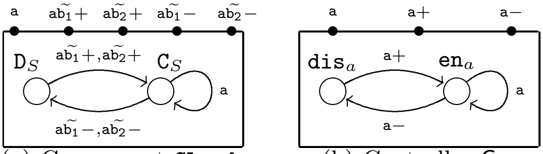
- $Q = 2^P \times \Pi_i Q_i$ (we write $\gamma(q_1,\ldots,q_n)$ for $\langle\gamma,\langle q_1,\ldots,q_n\rangle\rangle \in Q$), and
- $\twoheadrightarrow \subseteq Q \times 2^P \times Q$ is the least set of transitions given by the rules in Fig. 13.

Each state of the transition system keeps, not only the states of all components but also, the set γ of all enabled interactions. Rule [INT] stands for ordinary interactions. Rule [ADD] accounts for the addition of a new global interaction a to the set of enabled interactions γ. Rule [DEL] specifies the removal of an enabled interaction and is analogous to [ADD].

Example 9. Consider the reconfigurable BI(P) system in Example 8. The initial state in which no connection has been established is given by the term $\varnothing(\mathtt{D}_S, \mathtt{D}_{C_1}, \mathtt{D}_{C_2})$. In this state, the system can start a session between the `Cloud` and either $\mathtt{VM_1}$ or $\mathtt{VM_2}$. Assuming that a session with $\mathtt{VM_1}$ is established, then the system can move as follows (where $s = \{\mathtt{ab_1}\}(\mathtt{C}_S, \mathtt{C}_{C_1}, \mathtt{D}_{C_2})$):

$$\varnothing(\mathtt{D}_S, \mathtt{D}_{C_1}, \mathtt{D}_{C_2}) \xtwoheadrightarrow{\mathtt{ab_1}+} s \xtwoheadrightarrow{\mathtt{ab_1}} \ldots \xtwoheadrightarrow{\mathtt{ab_1}} s \xtwoheadrightarrow{\mathtt{ab_1}-} \varnothing(\mathtt{D}_S, \mathtt{D}_{C_1}, \mathtt{D}_{C_2})$$

Let $\mathcal{R}(R) = \{a \mid (q,a,\rho,q') \in\twoheadrightarrow \text{ and } \rho \neq \epsilon\}$ be the set of reconfigurable interactions of a reconfigurable component $R = (Q,P,\twoheadrightarrow)$. For any $a \in \mathcal{R}(R)$ we add two additional ports $\widetilde{a}^R+$ and $\widetilde{a}^R-$ in the encoded component, where $\widetilde{a}^R = (a \cap P) \cup \{\tilde{p} \mid p \in a \smallsetminus P\}$ (decorations $\tilde{\cdot}$ are needed to guarantee uniqueness of ports in different components). We let $\widetilde{\mathcal{R}(R)} = \{\widetilde{a}^R \mid a \in \mathcal{R}(R)\}$. For example, $\widetilde{\mathcal{R}(\mathtt{Cloud})} = \{\widetilde{\mathtt{ab_1}}, \widetilde{\mathtt{ab_2}}\}$. The function $\mathcal{R}(\cdot)$ is extended to reconfigurable BI(P) systems $RS = \gamma(R_1,\ldots,R_n)$ by letting $\mathcal{R}(RS) = \bigcup_{1\leq i\leq n} \mathcal{R}(R_i)$.



(a) Component `Cloud` (b) Controller C_a

Fig. 14. Encoding reconfigurable BI(P) in BI(P)

Definition 18. *Let* $R = (Q, P, \twoheadrightarrow)$ *be a reconfigurable component. The corresponding BI(P) component* $[\![R]\!]$ *is* $(Q, P \cup (\widetilde{\mathcal{R}(R)} \times \{+,-\}), \rightarrow)$ *with* $(q, a, q') \in \rightarrow$ *iff* $(q, a, \epsilon, q') \in \twoheadrightarrow$ *or* $(q, a', \rho, q') \in \twoheadrightarrow$, $\rho \neq \epsilon$ *and* $a = (\widetilde{a'}^{R}, \rho)$.

Figure 14(a) shows the BI(P) component corresponding to the reconfigurable component `Cloud` depicted in Fig. 12.

Next, we associate any reconfigurable interaction a with a new BI(P) component $\mathsf{C}_a = (Q_{\mathsf{C}_a}, P_{\mathsf{C}_a}, \rightarrow)$, called *controller*: it models the dynamic enabling / disabling of a (see Fig. 14(b)).

Definition 19. *Let* $RS = \gamma(R_1, \ldots, R_n)$ *be a reconfigurable BI(P) system with* $\mathcal{R}(RS) = \{a_0, \ldots, a_j\}$*. The corresponding BI(P) system is defined by*

$$[\![\gamma(R_1, \ldots, R_n)]\!] \;= [\![\gamma]\!]([\![R_1]\!], \ldots, [\![R_n]\!], \mathsf{C}_{a_0}, \ldots, \mathsf{C}_{a_j})$$

where $[\![\gamma]\!] = (\gamma \smallsetminus \mathcal{R}(RS)) \cup (\bigcup_{a \in \mathcal{R}(RS), \rho \in \{\epsilon,+,-\}} \{[\![a]\!]_\rho\})$ *with*

$$[\![a]\!]_\rho = \begin{cases} \{a\rho\} \cup \{\widetilde{a}^{R_i}\rho \mid 1 \leq i \leq n \text{ and } a \in \mathcal{R}(R_i)\} & \text{if } \rho \in \{+,-\} \\ \{a\} \cup \{p \mid 1 \leq i \leq n \text{ and } p \in a \downarrow_{P_i}\} & \text{if } \rho = \epsilon \end{cases}$$

Finally, any state $\gamma(q_1, \ldots, q_n)$ *of* RS *is associated with a state* $[\![\gamma(q_1, \ldots, q_n)]\!] = (q_1, \ldots, q_n, s_0, \ldots, s_j)$ *of* $[\![RS]\!]$ *where* $s_i = \mathtt{en}_{a_i}$ *if* $a_i \in \gamma$*, and* $s_i = \mathtt{dis}_{a_i}$ *if* $a_i \notin \gamma$.

Example 10. The reconfigurable system introduced in Example 8 is encoded as the BI(P) system shown in Fig. 15.

Theorem 4. *Given* $RS = \gamma(R_1, \ldots, R_n)$*, we have* $\gamma(q_1, \ldots, q_n) \overset{a}{\twoheadrightarrow} \gamma'(q_1', \ldots, q_n')$ *iff* $\exists b \in \{a, \gamma_a, \gamma_{a+}, \gamma_{a-}\}$ *s.t.* $[\![\gamma(q_1, \ldots, q_n)]\!] \xrightarrow{b}_{[\![\gamma]\!]} [\![\gamma'(q_1', \ldots, q_n')]\!]$.

8.2 Dynamic BI(P)

In this section we further extend BI(P) by allowing the dynamic replication of components. In the case of dynamic BI(P) we can define systems that are possibly infinite state and more expressive than ordinary BI(P) systems. We take as an inspiring example the notion of correlation sets in web services [40,23]. In these cases, when a service call is made, then an instance of the session is initialized with suitable correlation data (e.g., specific message fields) gathered

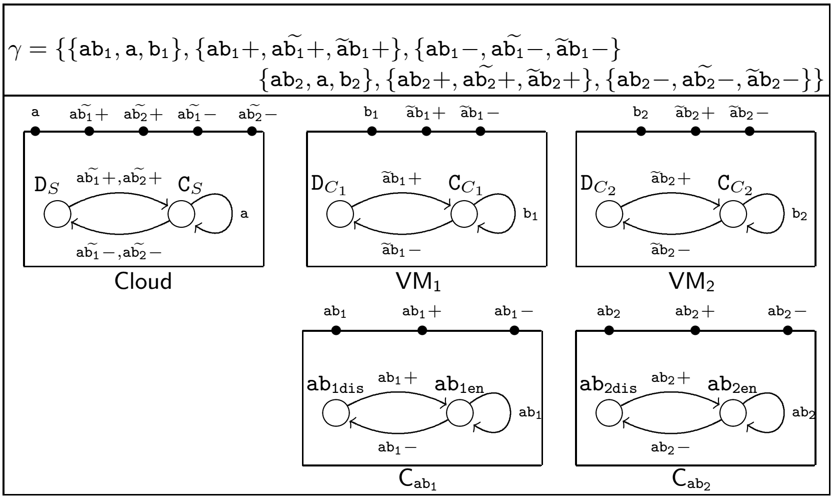

$\gamma = \{\{\mathtt{ab}_1, \mathtt{a}, \mathtt{b}_1\}, \{\mathtt{ab}_1+, \mathtt{a}\widetilde{\mathtt{b}_1}+, \tilde{\mathtt{a}}\mathtt{b}_1+\}, \{\mathtt{ab}_1-, \mathtt{a}\widetilde{\mathtt{b}_1}-, \tilde{\mathtt{a}}\mathtt{b}_1-\}$
$\{\mathtt{ab}_2, \mathtt{a}, \mathtt{b}_2\}, \{\mathtt{ab}_2+, \mathtt{a}\widetilde{\mathtt{b}_2}+, \tilde{\mathtt{a}}\mathtt{b}_2+\}, \{\mathtt{ab}_2-, \mathtt{a}\widetilde{\mathtt{b}_2}-, \tilde{\mathtt{a}}\mathtt{b}_2-\}\}$
a
$\mathtt{a}\widetilde{\mathtt{b}_1}+$
$\mathtt{a}\widetilde{\mathtt{b}_2}+$
$\mathtt{a}\widetilde{\mathtt{b}_1}-$
$\mathtt{a}\widetilde{\mathtt{b}_2}-$
$\mathtt{D}_S$
$\mathtt{a}\widetilde{\mathtt{b}_1}+,\mathtt{a}\widetilde{\mathtt{b}_2}+$
$\mathtt{C}_S$
a
$\mathtt{a}\widetilde{\mathtt{b}_1}-,\mathtt{a}\widetilde{\mathtt{b}_2}-$
Cloud
$\mathtt{b}_1$
$\tilde{\mathtt{a}}\mathtt{b}_1+$
$\tilde{\mathtt{a}}\mathtt{b}_1-$
$\mathtt{D}_{C_1}$
$\tilde{\mathtt{a}}\mathtt{b}_1+$
$\mathtt{C}_{C_1}$
$\mathtt{b}_1$
$\tilde{\mathtt{a}}\mathtt{b}_1-$
VM$_1$
$\mathtt{b}_2$
$\tilde{\mathtt{a}}\mathtt{b}_2+$
$\tilde{\mathtt{a}}\mathtt{b}_2-$
$\mathtt{D}_{C_2}$
$\tilde{\mathtt{a}}\mathtt{b}_2+$
$\mathtt{C}_{C_2}$
$\mathtt{b}_2$
$\tilde{\mathtt{a}}\mathtt{b}_2-$
VM$_2$
$\mathtt{ab}_1$
$\mathtt{ab}_1+$
$\mathtt{ab}_1-$
$\mathtt{ab}_{1\mathtt{dis}}$
$\mathtt{ab}_1+$
$\mathtt{ab}_{1\mathtt{en}}$
$\mathtt{ab}_1$
$\mathtt{ab}_1-$
$\mathsf{C}_{\mathtt{ab}_1}$
$\mathtt{ab}_2$
$\mathtt{ab}_2+$
$\mathtt{ab}_2-$
$\mathtt{ab}_{2\mathtt{dis}}$
$\mathtt{ab}_2+$
$\mathtt{ab}_{2\mathtt{en}}$
$\mathtt{ab}_2$
$\mathtt{ab}_2-$
$\mathsf{C}_{\mathtt{ab}_2}$

Fig. 15. A simple reconfigurable BI(P) system encoded in ordinary BI(P)

for the partner. To this aim we exploit *colored* tokens, where the colors are freshly created session identifiers. This way, we do not need to replicate the ports and structure of components, instead we keep all the coloured tokens within the same instance of the component, then it can happen that two or more coloured tokens mark the same state at the same time. An interaction is possible only when all the involved components carry correlated colors, i.e., identifiers for the same session. In fact, while session identifiers are created locally to each component (e.g., $\mathtt{s}_1$ in a first component and $\mathtt{s}_2$ in a second component), a new interaction is also created that correlates them (e.g., $\mathtt{s}_1\mathtt{s}_2$). Possibly many sessions are opened with the same partners involved. In subsequent interactions, correlation tokens are then exploited to identify the session that interaction is part of. When the session ends, the correlation tokens are discarded. At the beginning, when the system is initialized, we assume that all components carry correlated tokens, i.e., that they are all part of the same session. Notably, reachability is decidable for dynamic BI(P). This is achieved by tracing a correspondence between dynamic BI(P) systems and Place/Transition (P/T) Petri nets.

We assume an infinite set of port names $\mathcal{P}$ ranged over by $\mathtt{a}, \mathtt{b}, \ldots$, an infinite set of port variable names $\mathcal{X}$ ranged over by $x, y, \ldots$, and an infinite set of state names $\mathcal{Q}$ ranged over by $\mathtt{p}, \mathtt{q}, \ldots$, such that $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{X}$ pairwise disjoint. As in general an interaction is related to a specific session, we sometimes decorate ports and interactions with specific correlation tokens as their subscripts. For example, for $a = \mathtt{ab}$ we write $a_\mathtt{c}$ for $\mathtt{a}_\mathtt{c}\mathtt{b}_\mathtt{c}$.

Definition 20 (Dynamic component). *A* dynamic component *is a tuple* $D = (Q, P, \rightarrow)$ *where* $Q \subset \mathcal{Q}$ *is a set of places,* $P \subset \mathcal{P}$ *is a set of ports, and* $\rightarrow$ *is a finite set of transitions, each having one of the following shapes:*

$$\frac{\mathtt{q}(x) \xrightarrow{\alpha} \mathtt{q}'(x) \qquad \mathtt{a} \in \delta(P) \qquad \alpha \in \{a_x, x\}}{\langle P, \mathtt{q}(\mathtt{a}) \oplus f\rangle \xrightarrow{\alpha[x/\mathtt{a}]} \langle P, \mathtt{q}'(\mathtt{a}) \oplus f\rangle}\,[\textsc{Cint}]$$

$$\frac{\mathtt{q}(x) \xrightarrow{a_x y+} \mathtt{q}'(x) \oplus \mathtt{q}''(y) \qquad \mathtt{a} \in \delta(P) \qquad \mathtt{b} \notin P}{\langle P, \mathtt{q}(\mathtt{a}) \oplus f\rangle \xrightarrow{a_{\mathtt{a}}\mathtt{b}+} \langle P \cup \{\mathtt{b}\} \cup P_{\mathtt{b}}, \mathtt{q}'(\mathtt{a}) \oplus \mathtt{q}''(\mathtt{b}) \oplus f\rangle}\,[\textsc{Copen}]$$

$$\frac{\mathtt{q}(x) \xrightarrow{x-} \varnothing \qquad \mathtt{a} \in \delta(P)}{\langle P, q(\mathtt{a}) \oplus f\rangle \xrightarrow{\mathtt{a}-} \langle P \smallsetminus (\{\mathtt{a}\} \cup P_{\mathtt{a}}), f\rangle}\,[\textsc{Cclose}]$$

Fig. 16. Operational semantics of dynamic components

- $\mathtt{q}(x) \xrightarrow{a_x} \mathtt{q}'(x)$, *i.e., (a coloured version of) a BI(P) transition;*
- $\mathtt{q}(x) \xrightarrow{a_x y+} \mathtt{q}'(x) \oplus \mathtt{q}''(y)$, *i.e., a port creation;*
- $\mathtt{q}(x) \xrightarrow{x-} \varnothing$, *i.e., a port removal;*
- $\mathtt{q}(x) \xrightarrow{x} \mathtt{q}'(x)$, *i.e., an interaction over a dynamically created port.*

Ports that appear in labels of the form a_x are parametric to the correlation token and are called *static* ports; the other ports are called *dynamic*. We assume static ports cannot be used as correlation tokens. In the following we denote by P_x the set of static ports of P, by $P_{\mathtt{a}}$ the set of static ports in P parametrized by the token $\mathtt{a}$ and by $\delta(P)$ the set of dynamic ports. For example, if $P = \{\mathtt{a}, \mathtt{b}\}$ with $\mathtt{a}$ static and $\mathtt{b}$ dynamic, then $P_{\mathtt{c}} = \{\mathtt{a}_{\mathtt{c}}\}$. Note that if all transitions have the form $\mathtt{q}(x) \xrightarrow{a_x} \mathtt{q}'(x)$ then D is essentially an ordinary BI(P) component.

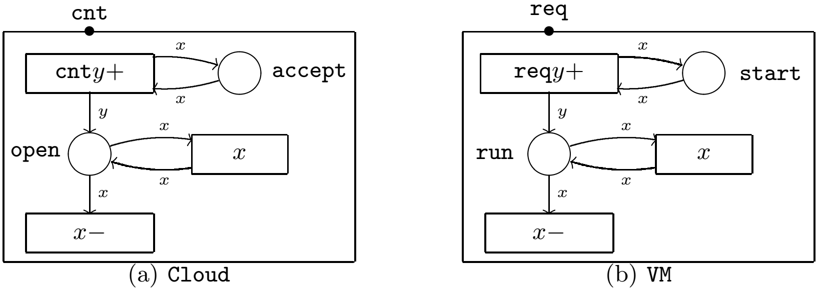
The current state of a dynamic component $D = (Q, P, \rightarrow)$ takes the form $\langle P, f\rangle$ with $P \subset \mathcal{P}$ defining the current ports of the component (that includes opened sessions) and $f : Q \rightarrow 2^P$ such that $f(\mathtt{q}_1) \cap f(\mathtt{q}_2) = \varnothing$ for $\mathtt{q}_1 \neq \mathtt{q}_2$. The function f represents the current internal state of the component replicas. For example, if $f(\mathtt{q}) = \{\mathtt{a}, \mathtt{b}\}$ then there are two replicas of the component, one involved in session $\mathtt{a}$ and one in $\mathtt{b}$ both with current state $\mathtt{q}$. The condition $f(\mathtt{q}_1) \cap f(\mathtt{q}_2) = \varnothing$ for $\mathtt{q}_1 \neq \mathtt{q}_2$ guarantees that each replica is associated with a different session and that to each session corresponds exactly one state.

As a matter of notation we denote $f \oplus \mathtt{p}(\mathtt{a})$ the function defined as

$$(f \oplus \mathtt{p}(\mathtt{a}))(\mathtt{q}) = \begin{cases} f(\mathtt{q}) & \text{if } \mathtt{q} \neq \mathtt{p} \\ f(\mathtt{q}) \cup \{\mathtt{a}\} & \text{if } \mathtt{q} = \mathtt{p} \end{cases}$$

Remark 1. Initially there is only one session opened for each component. To shorten the notation but without loss of generality, we shall assume that such initial session identifier is void, i.e. $f(\mathtt{p}) = \{\bullet\}$ and omit the corresponding port $\bullet$ from the drawing of components (in the initial and subsequent states).

The operational semantics of components is given by the three rules in Fig. 16. The first rule ([Cint]) deals with both: i) the case of an ordinary interaction $a_{\mathtt{a}}$ (here coloured by the token $\mathtt{a}$); and ii) the case of a dynamic interaction over the session associated with $\mathtt{a}$. The rule ([Copen]) is the most complex one, as



(a) Cloud (b) VM

Fig. 17. Two dynamic components

it deals with component spawning and port creation. Here the freshly created session identifier is $\mathtt{b}$, which is then used as a fresh dynamic port, together with suitable instances $P_{\mathtt{b}}$ of the static ports of the component. The spawned instance of the component has initial state $\mathtt{q}''(\mathtt{b})$. Ports in $P_{\mathtt{b}}$ will allow the spawned instance of the component to interact on static ports with some other spawned components that are part of the same session. Moreover, the spawned instance of the component will be able to interact on the port $\mathtt{b}$ by synchronizing with all the other spawned components that are part of the same session. Note that although the token $\mathtt{b}$ has been created within the session $\mathtt{a}$, such information is not maintained in the state, i.e., sessions $\mathtt{a}$ and $\mathtt{b}$ will run independently. Finally, the rule ([Cclose]) deals with session closure, where the token $\mathtt{a}$ and all the ports $\{\mathtt{a}\} \cup P_{\mathtt{a}}$ associated with the closed session $\mathtt{a}$ are discarded.

Example 11. Consider a cloud manager that interacts with a possibly unbounded number of clients. This behaviour can be modeled as the component depicted in Fig. 17(a), where arcs are decorated with the colors of the involved tokens. (The analogous client is in Fig. 17(b).) The component in Fig. 17(a) has one static port cnt, two places accept and open with the following three transitions:

- $t_0 = \mathtt{accept}(x) \xrightarrow{\mathtt{cnt}_x\ y+} \mathtt{accept}(x) \oplus \mathtt{open}(y)$: the action cnt opens a new session (whose id is stored in place open).
- $t_1 = \mathtt{open}(x) \xrightarrow{x} \mathtt{open}(x)$: for any open session, an action on the corresponding dynamic port can be repeatedly performed.
- $t_2 = \mathtt{open}(x) \xrightarrow{x-} \varnothing$: An already opened session x is closed by synchronizing all participants to that session on the interaction $x-$.

Definition 21 (Dynamic BI(P) system). *A* dynamic BI(P) system $DS = \gamma(D_1, \ldots, D_n)$ *is the composition of a finite set* $\{D_i\}_{i=1}^n$ *of dynamic BI(P) components* $D_i = (Q_i, P_i, \rightarrow_i)$ *such that their sets of ports are pairwise disjoint, parametrized by a set* $\gamma \subset 2^P$ *of interactions over the set of ports* $P = \biguplus_{i=1}^n P_i$.

Without loss of generality, we assume that for any $a \in \gamma$ it is either the case that a contains static ports only and we call it *static* or it contains no static port

$$\frac{a \in \gamma \qquad \forall i. s_i \xrightarrow{a\downarrow_{P_i}} s_i'}{\gamma(s_1,\ldots,s_n) \xrightarrow{a} \gamma(s_1',\ldots,s_n')}\,[\textsc{Sint}]$$

$$\frac{\begin{array}{c} a \in \gamma \quad i \in I(a) \implies s_i \xrightarrow{a\downarrow_{P_i}\mathtt{b_i}+} s_i' \quad \mathtt{b_i} \text{ fresh} \quad \sigma = [{}^{ids_i(a)}/_{\mathtt{b}_i}]_{i\in I(a)} \\ i \in \overline{I(a)} \implies s_i' = s_i \qquad\qquad b = \{\mathtt{b_i}\}_{i\in I(a)} \end{array}}{\gamma(s_1,\ldots,s_n) \xrightarrow{a} (\gamma \cup \{b\} \cup \gamma_\sigma)(s_1',\ldots,s_n')}\,[\textsc{Sopen}]$$

$$\frac{a \in \gamma \qquad i \in I(a) \implies s_i \xrightarrow{a\downarrow_{P_i}-} s_i' \qquad i \in \overline{I(a)} \implies s_i' = s_i}{\gamma(s_1,\ldots,s_n) \xrightarrow{a} (\gamma \ominus a)(s_1',\ldots,s_n')}\,[\textsc{Sclose}]$$

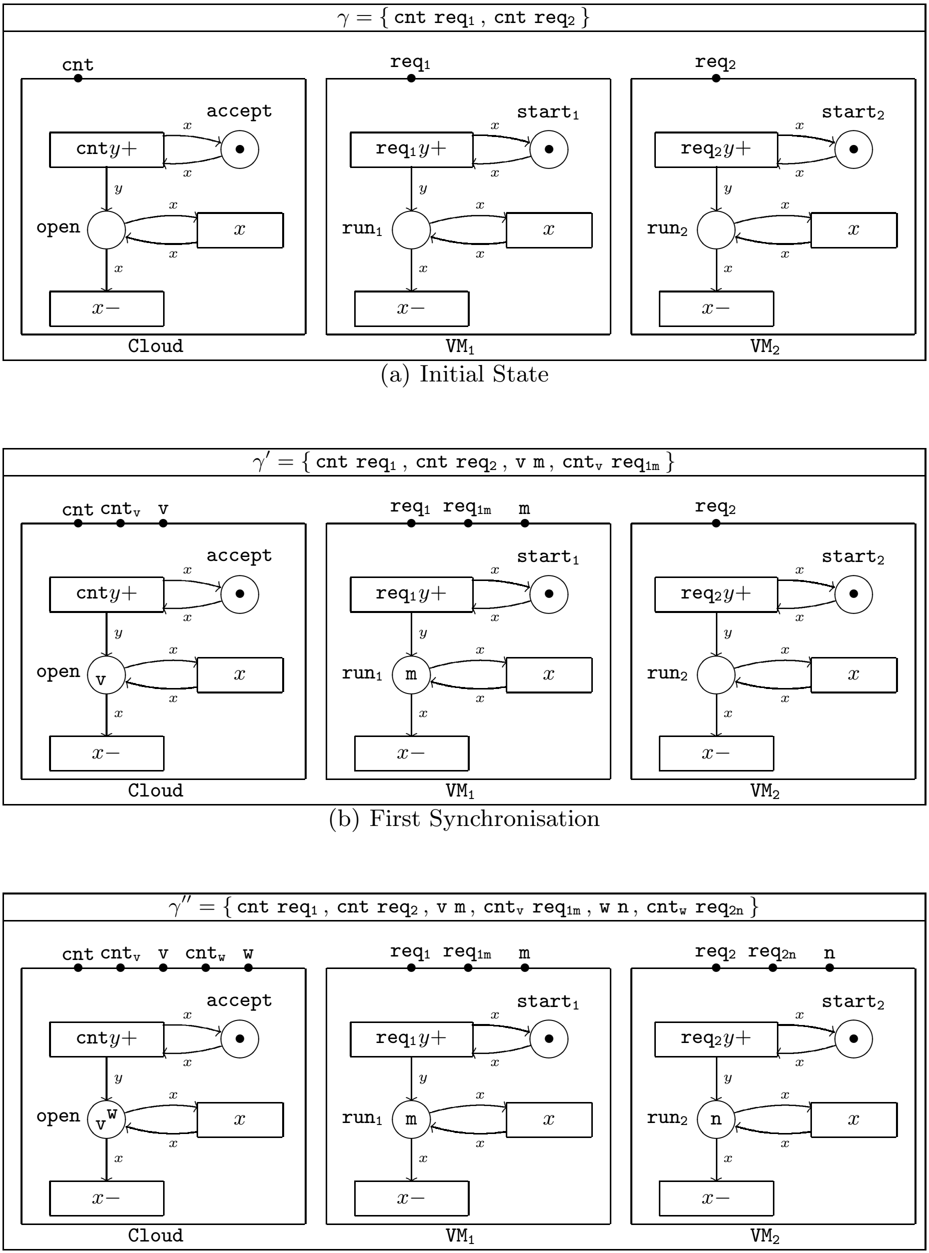
Fig. 18. Operational semantics of dynamic BI(P) systems

at all and we call it *dynamic*. Moreover, if $a \downarrow_{P_i}$ is made of static ports, then $a \downarrow_{P_i} = a'_{\mathtt{a}_i}$ for some a' and $\mathtt{a}_i \in P_i$, i.e., all static ports in $a \downarrow_{P_i}$ are parametrized by the same session identifier $\mathtt{a}_i$. In such case, we let $ids_i(a)$ denote $\mathtt{a}_i$ We write $I(a)$ to denote the set of indices $\{i \mid \neg(a\#P_i)\}$ of the components involved in a and $\overline{I(a)}$ to denote its complement $[1,n] \smallsetminus I(a) = \{i \mid a\#P_i\}$. If a is static, we denote by $ids(a)$ the set $\{ids_i(a) \mid i \in I(a)\}$, otherwise we let $ids(a) = \varnothing$.

Given a substitution $\sigma = [{}^{\mathtt{a}_i}/_{\mathtt{b}_i}]_{i\in I}$ and a static interaction $a \in \gamma$ such that $ids(a) \subseteq \{\mathtt{a}_i\}_{i\in I}$ we write a_σ for the interaction obtained by replacing in a each parameter $\mathtt{a}_i$ by the corresponding parameter $\mathtt{b}_i$. Moreover, we write γ_σ for the set of renamed static interactions $\{a_\sigma \mid a \in \gamma \wedge ids(a) \subseteq \{\mathtt{a}_i\}_{i\in I}\}$. Finally, given a dynamic interaction a we let $\gamma \ominus a = \{a' \in \gamma \mid a' \cap a = \varnothing \wedge ids(a') \cap a = \varnothing\}$ be the set of interactions in γ where the ports in a do not appear.

Let s_i range over $2^{P_i} \times P_i^{Q_i}$ representing a generic state of the component D_i. The semantics of a dynamic BI(P) system $\gamma(D_1,\ldots,D_n)$ is defined by the three rules in Fig. 18. Rule [Sint] deals with the usual synchronization. Rule [Sopen] represents the opening of a session: local fresh session identifiers $\mathtt{b_i}$ are created (that are used in s_i') and the set of interactions is enriched with the new session synchronization $b = \{\mathtt{b_i}\}_{i\in I(a)}$ and a renamed instance γ_σ of the static interactions in γ (for the new session). Finally, rule [Sclose] deals with the synchronized closing of a session: note that the set of interactions is updated by removing the interactions concerned with the closed session.

Example 12. Consider the dynamic BI(P) components introduced in Example 11. We illustrate one possible run of the server with two clients in Fig. 19. Roughly, it corresponds to the series of transitions in Fig. 20, where $\gamma, \gamma', \gamma''$ are the ones indicated in Fig. 19. Note that suitable replicas $\mathtt{cnt_v}$, $\mathtt{cnt_w}$, $\mathtt{req_{1m}}$, $\mathtt{req_{2n}}$ of the static ports $\mathtt{cnt}$, $\mathtt{req}_1$, $\mathtt{req}_2$ have been created locally to each component, and that the set of interactions has been enriched with suitable replicas $\mathtt{cnt_v}\,\mathtt{req_{1m}}$ and $\mathtt{cnt_w}\,\mathtt{req_{2n}}$ of the static interactions $\mathtt{cnt}\,\mathtt{req_1}$ and $\mathtt{cnt}\,\mathtt{req_2}$ together with freshly created dynamic interactions $\mathtt{v\,m}$ and $\mathtt{w\,n}$. Let s denote the last state reached. Then, the server can interact with the clients by performing the inter-


γ = { cnt req₁ , cnt req₂ }
cnt
req₁
req₂
accept
start₁
start₂
cnt y+
req₁ y+
req₂ y+
open
run₁
run₂
x
x−
y
Cloud
VM₁
VM₂
γ′ = { cnt req₁ , cnt req₂ , v m , cnt_v req₁ₘ }
cnt cnt_v v
req₁ req₁ₘ m
v
m
γ″ = { cnt req₁ , cnt req₂ , v m , cnt_v req₁ₘ , w n , cnt_w req₂ₙ }
cnt cnt_v v cnt_w w
req₂ req₂ₙ n
v w
n

(a) Initial State

(b) First Synchronisation

(c) Second Synchronisation

Fig. 19. A run of the server with two clients

$$\gamma \begin{pmatrix} \langle\{\texttt{cnt}\}, \texttt{accept}(\bullet)\rangle\,, \\ \langle\{\texttt{req}_1\}, \texttt{start}_{\texttt{1}}(\bullet)\rangle\,, \\ \langle\{\texttt{req}_2\}, \texttt{start}_{\texttt{2}}(\bullet)\rangle \end{pmatrix} \xrightarrow{\texttt{cnt}\ \texttt{req}_1} \gamma' \begin{pmatrix} \langle\{\texttt{cnt}, \texttt{cnt}_{\texttt{v}}, \texttt{v}\}, \texttt{accept}(\bullet) \oplus \texttt{open}(\texttt{v})\rangle\,, \\ \langle\{\texttt{req}_1, \texttt{req}_{\texttt{1m}}, \texttt{m}\}, \texttt{start}_{\texttt{1}}(\bullet) \oplus \texttt{run}_{\texttt{1}}(\texttt{m})\rangle\,, \\ \langle\{\texttt{req}_2\}, \texttt{start}_{\texttt{2}}(\bullet)\rangle \end{pmatrix}$$

$$\xrightarrow{\texttt{cnt}\ \texttt{req}_2} \gamma'' \begin{pmatrix} \langle\{\texttt{cnt}, \texttt{cnt}_{\texttt{v}}, \texttt{v}, \texttt{cnt}_{\texttt{w}}, \texttt{w}\}, \texttt{accept}(\bullet) \oplus \texttt{open}(\texttt{v}) \oplus \texttt{open}(\texttt{w})\rangle\,, \\ \langle\{\texttt{req}_1, \texttt{req}_{\texttt{1m}}, \texttt{m}\}, \texttt{start}_{\texttt{1}}(\bullet) \oplus \texttt{run}_{\texttt{1}}(\texttt{m})\rangle\,, \\ \langle\{\texttt{req}_2, \texttt{req}_{\texttt{2n}}, \texttt{n}\}, \texttt{start}_{\texttt{2}}(\bullet) \oplus \texttt{run}_{\texttt{2}}(\texttt{n})\rangle \end{pmatrix}$$

Fig. 20. Transitions representing a run of the server with two clients

actions $\texttt{v m}$ and $\texttt{w n}$ as many times as needed, with the system remaining in the same state s: $s \xrightarrow{\texttt{v m}} s \xrightarrow{\texttt{w n}} s \cdots$ Finally, we illustrate the case when the session between the server and the second client is closed:

$$s \xrightarrow{\texttt{w n}} \gamma' \begin{pmatrix} \langle\{\texttt{cnt}, \texttt{cnt}_{\texttt{v}}, \texttt{v}\}, \texttt{accept}(\bullet) \oplus \texttt{open}(\texttt{v})\rangle\,, \\ \langle\{\texttt{req}_1, \texttt{req}_{\texttt{1m}}, \texttt{m}\}, \texttt{start}_{\texttt{1}}(\bullet) \oplus \texttt{run}_{\texttt{1}}(\texttt{m})\rangle\,, \\ \langle\{\texttt{req}_2\}, \texttt{start}_{\texttt{2}}(\bullet)\rangle \end{pmatrix}$$

The above transition is obtained by combining a closing transitions of the server (label $\texttt{w}-$) with a closing transition of the second client (label $\texttt{n}-$).

Unlike reconfigurable BI(P) systems, dynamic BI(P) systems are strictly more expressive than ordinary BI(P) systems. This can be immediately seen by noting that BI(P) systems are finite state, while this is not the case for dynamic BI(P) systems (see, e.g., Example 12).

In [12] we have defined a correspondence between dynamic BI(P) systems and Place/Transition Petri nets. This is interesting because: i) it shows that properties like reachability remains decidable and ii) it draws a nice analogy with the correspondence between ordinary BI(P) systems and Petri nets in [10].

Roughly, given a dynamic BI(P) system $DS = \gamma(D_1, ..., D_n)$ we define a P/T Petri net $N(DS)$ whose places are tuples of states from components $D_1, ..., D_n$ and whose transitions represent the possible interactions. Note that $N(DS)$ is determined statically and although it may contain more places and transitions than those strictly necessary, it is finite.

Another dynamic extension of BIP is Dy-BIP [7]. With respect to Dy-BIP, we think dynamic BI(P) has some advantages. While Dy-BIP imposes ad hoc restrictions (e.g., transitions of atomic components are labelled with only one single local port instead of a set of local ports) and extensions (e.g. transitions of atomic components are decorated with non-local architecture constraints that may involve port names of other components, thus compromising the modularity of the specification and moreover history variables are introduced to store the identity of interacting components), this is not necessary for dynamic BI(P). Furthermore, the number of component instances cannot change in Dy-BIP, contrary to dynamic BI(P). Finally, the definition of Dy-BIP systems can be error-prone or lead to incomplete specifications unless the design methodology outlined in [7] is adopted.

9 Concluding Remarks

One of the main limitations of the state-of-the-art theories of component-based system is the lack of a reference paradigm for describing and analyzing their highly dynamic interactions. In this paper we have overviewed some recent proposals for addressing this limitation that emerged within the ASCENS project, i.e., the Network-Conscious pi-calculus and possible BI(P) enhancements.

References

1. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. in Comp. Sci. 14(3), 329–366 (2004)
2. Arbab, F., Bruni, R., Clarke, D., Lanese, I., Montanari, U.: Tiles for reo. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 37–55. Springer, Heidelberg (2009)
3. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional semantics for open Petri nets based on deterministic processes. Mathematical Structures in Computer Science 15(1), 1–35 (2005)
4. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: SEFM'06, pp. 3–12. IEEE Computer Society Press, Los Alamitos (2006)
5. Bliudze, S., Sifakis, J.: The algebra of connectors - structuring interaction in BIP. IEEE Trans. Computers 57(10), 1315–1330 (2008)
6. Bliudze, S., Sifakis, J.: Causal semantics for the algebra of connectors. Formal Methods in System Design 36(2), 167–194 (2010)
7. Bozga, M., Jaber, M., Maris, N., Sifakis, J.: Modeling dynamic architectures using Dy-BIP. In: Gschwind, T., De Paoli, F., Gruhn, V., Book, M. (eds.) SC 2012. LNCS, vol. 7306, pp. 1–16. Springer, Heidelberg (2012)
8. Bruni, R.: Tile Logic for Synchronized Rewriting of Concurrent Systems. Ph.D. thesis, Computer Science Department, University of Pisa (1999)
9. Bruni, R., Lanese, I., Montanari, U.: A basic algebra of stateless connectors. Theor. Comput. Sci. 366(1-2), 98–120 (2006)
10. Bruni, R., Melgratti, H., Montanari, U.: Connector algebras, Petri nets, and BIP. In: Clarke, E., Virbitskaite, I., Voronkov, A. (eds.) PSI 2011. LNCS, vol. 7162, pp. 19–38. Springer, Heidelberg (2012)
11. Bruni, R., Melgratti, H., Montanari, U.: A connector algebra for P/T nets interactions. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011 – Concurrency Theory. LNCS, vol. 6901, pp. 312–326. Springer, Heidelberg (2011)
12. Bruni, R., Melgratti, H.C., Montanari, U.: Behaviour, interaction and dynamics. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 382–401. Springer, Heidelberg (2014)
13. Bruni, R., Melgratti, H.C., Montanari, U., Sobocinski, P.: Connector algebras for C/E and P/T nets' interactions. Logical Methods in Computer Science 9(3) (2013)
14. Bruni, R., Montanari, U.: Dynamic connectors for concurrency. Theor. Comput. Sci. 281(1-2), 131–176 (2002)
15. Campbell, A.T., Katzela, I., Miki, K., Vicente, J.B.: Open signaling for ATM, internet and mobile networks (OPENSIG'98). Computer Communication Review 29(1), 97–108 (1999)

16. Ciancia, V., Kurz, A., Montanari, U.: Families of symmetries as efficient models of resource binding. ENTCS 264(2), 63–81 (2010)
17. Combaz, J., Bensalem, S., Tiezzi, F., Margheri, A., Pugliese, R., Kofron, J.: Correctness of Service Components and Service Component Ensembles. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 107–159. Springer, Heidelberg (2015)
18. Ferrari, G.L., Montanari, U.: Tile formats for located and mobile systems. Inf. Comput. 156(1-2), 173–235 (2000)
19. Ferrari, G.L., Montanari, U., Tuosto, E.: Coalgebraic minimization of HD-automata for the pi-calculus using polymorphic types. Theor. Comput. Sci. 331(2-3), 325–365 (2005)
20. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: LICS 2001, pp. 93–104. IEEE Computer Society Press, Los Alamitos (2001)
21. Gadducci, F., Montanari, U.: The tile model. In: Proof, Language, and Interaction, pp. 133–166. MIT Press, Cambridge (2000)
22. Jongmans, S.-S.T.Q., Arbab, F.: Overview of thirty semantic formalisms for Reo. Scientific Annals of Computer Science 22(1), 201–251 (2012), doi:10.7561/SACS.2012.1.201
23. Lapadula, A., Pugliese, R., Tiezzi, F.: A formal account of WS-BPEL. In: Lea, D., Zavattaro, G. (eds.) COORDINATION 2008. LNCS, vol. 5052, pp. 199–215. Springer, Heidelberg (2008)
24. MacLane, S.: Categories for the Working Mathematician. Springer, Heidelberg (1971)
25. Mayer, P., Velasco, J., Klarl, A., Hennicker, R., Puviani, M., Tiezzi, F., Pugliese, R., Keznikl, J., Bureš, T.: The Autonomic Cloud. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 495–512. Springer, Heidelberg (2015)
26. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G.M., Peterson, L.L., Rexford, J., Shenker, S., Turner, J.S.: Openflow: enabling innovation in campus networks. Comput. Commun. Rev. 38(2), 69–74 (2008)
27. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I–II. Inf. Comput. 100(1), 1–77 (1992)
28. Montanari, U., Pistore, M.: Structured coalgebras and minimal hd-automata for the π-calculus. Theor. Comput. Sci. 340(3), 539–576 (2005)
29. Montanari, U., Rossi, F.: Graph rewriting, constraint solving and tiles for coordinating distributed systems. Applied Categorical Structures 7(4), 333–370 (1999)
30. Montanari, U., Sammartino, M.: Network conscious π-calculus: A concurrent semantics. ENTCS 286, 291–306 (2012)
31. Montanari, U., Sammartino, M.: A network-conscious π-calculus and its coalgebraic semantics. Theor. Comput. Sci. 546(0), 188–224 (2014), doi:10.1016/j.tcs.2014.03.009
32. Openflow foundation website, `http://www.openflow.org/`
33. Perry, D.E., Wolf, E.L.: Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes 17, 40–52 (1992)
34. Petri, C.: Kommunikation mit Automaten. Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn (1962)
35. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware, pp. 329–350 (2001)
36. Sammartino, M.: A Network-Aware Process Calculus for Global Computing and its Categorical Framework. Ph.D. thesis, University of Pisa (2013)

37. Sobocinski, P.: A non-interleaving process calculus for multi-party synchronisation. In: ICE'09. EPTCS, vol. 12, pp. 87–98 (2009)
38. Sobociński, P.: Representations of Petri net interactions. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 554–568. Springer, Heidelberg (2010)
39. Tennenhouse, D.L., Wetherall, D.J.: Towards an active network architecture. Comput. Commun. Rev. 26, 5–18 (1996)
40. Viroli, M.: A core calculus for correlation in orchestration languages. J. Log. Algebr. Program. 70(1), 74–95 (2007)
41. Rekhter, Y.: A border gateway protocol 4 (bgp-4). (March 1995), `http://www.ietf.org/rfc/rfc1771.txt`