Extending the Zero-Safe approach to Coloured, Reconfigurable and Dynamic Nets *

Roberto Bruni, Hernán Melgratti, and Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italia.
{bruni,melgratt,ugo}@di.unipi.it

Abstract. As web applications become more and more complex, primitives for handling interaction patterns among independent components become essential. In fact, distributed applications require new forms of transactions for orchestrating the progress of their negotiations and agreements. Still we lack foundational models that accurately explain the crucial aspects of the problem. In this work we explore how to model transactions in *coloured*, *reconfigurable* and *dynamic nets*, (i.e., high-level/high-order Petri nets that can express mobility and can extend themselves dynamically during their execution). Starting from *zero-safe* nets — a well-studied extension of Place/Transition Petri nets with a transactional mechanism based on a distinction between consistent (observable) and transient (hidden) states — we show how the zero-safe approach can be smoothly applied to a hierarchy of nets of increasing expressiveness.

1 Introduction

To some extent, *place/transition Petri nets* (P/T nets) [27,28] are for Concurrency Theory what finite automata are for the Theory of Computation: their rigorous theories have been consolidated in pioneering work; they are foundational models for many other languages and calculi; they have been enriched in a number of ways (e.g. time, stochastic, data type, high-order and reflection) for taking into account particular features demanded by real case studies and scenarios; they have been applied with success interdisciplinarily and even in industrial applications; they have been a constant reference for comparing emerging paradigms with; they admit intuitive graphical presentations; they are widely used in software engineering and in system specification and verification.

Nowadays, one of the main challenges for researchers with interest in Concurrency is the definition of adequate models for global computing applications, where aspects like distribution, name and code mobility, security, quality of services, and coordination are stretched to the very limit. Several of these aspects have been investigated separately and sometimes combined especially with

* Research supported by the MSR Cambridge Project Napi, by the FET-GC Project IST-2001-32747 Agile, by the MIUR Project COFIN 2001013518 CoMeta, and by the MURST-CNR 1999 Project, *Software Architectures on Cooperative WAN.*

the help of suitable *process calculi* (e.g. π-calculus [26], join calculus [21], spi-calculus [1], ambient calculus [19]), where new primitives can be easily introduced and experimented with. Although such calculi are much more expressive than P/T nets, it is possible to recover their spirit by progressively enriching the basic P/T net model with high-level and high-order features, like exemplified in [15]. The *Petri Box* calculus [4] is a different approach for reconciling both worlds.

In this paper, the aspect we want to focus on is *orchestration*. In fact, as more and more complex global computing applications are developed, then more primitives for handling common interaction patterns between independent components become essential. Academy and Industry are showing renewed interest in the orchestration of distributed applications via programming languages and calculi with primitive transactional mechanisms for managing electronic negotiations and contracts carried on among independent components. Although some solutions have been proposed in the literature (see the section on *related work*), still there is no complete agreement on the foundational models that better expose the crucial points of the problem.

The solution we propose in the paper relies on the so-called *zero-safe approach*, that is shown to span along a hierarchy of concurrent models (of increasing expressiveness), from P/T nets to *dynamic nets*, (i.e., high-level petri nets that can express dynamic network reconfigurability and reflection), stepping through *coloured nets* and *reconfigurable nets*. The hierarchy is indeed the one proposed in [15], where it is also shown that each net flavors correspond to a typeable fragment of join calculus. The straight consequence is that the zero-safe approach can be transferred also to those (sub)calculi at no additional cost.

Zero-safe approach. Zero-safe nets (ZS *nets*) have been introduced to model transactions in concurrent systems [11]. The basic model extends P/T *nets* with a mechanism for expressing serializable concurrent (multiway) transactions. In ZS nets there are two kinds of places (and, consequently, two kinds of tokens), called stable and zero-safe. Roughly, a transaction on a ZS net is a concurrent computation that departs from and arrives to a multiset of stable tokens. Recently, they have been used in [8] to encode short-running transactions of Microsoft Biztalk®, a commercial workflow management system [30]. ZS nets additionally provides a "dynamic" specification of transactions boundaries (as opposed to the "static" one of BizTalk) supporting multiway transactions, which retain several entry and exit points, and admit a number of participants which is statically unknown. Nevertheless, ZS nets are not suitable to express some interesting aspects of negotiations in global computing, such as value passing, dynamic reconfiguration of communication, name mobility, programmable compensations and nesting. Also their expressive power is limited as e.g. reachability is decidable [16].

ZS nets offer a two-level view of the modeled system: (1) the concrete operational view where transient places and the coordination mechanism between activities participating to a transaction are fully exposed; (2) the abstract view, where transactions are seen as atomic activities, and the user is aware of stable places only, while transient places are transparent. In fact, the abstract view is given by an ordinary P/T net, whose places are the stable places of the ZS net and whose transitions are the transactions of the ZS net. Moreover, the correspondence between the two views admits a rigorous mathematical characterization as coreflection between suitable model categories. It is worth remarking that ZS nets with a finite number of transitions can yield abstract P/T nets with infinitely many transitions. From the system designer viewpoint, this means that the combinatorial features of ZS nets can be exploited to keep small the size of the architecture.

The zero-safe approach has been extended to more expressive frameworks such as nets with read and inhibitor arcs [13], which have been shown expressive enough to give a concurrent operational semantics to the language TraLinda (an extension of Linda with transactional primitives).

From the implementation point of view, a distributed interpreter for ZS nets has been proposed in [10] that is based on the ordinary unfolding construction for Petri nets, while both centralized and distributed interpreters have been proposed in [7,8], which are written in (distributed) join calculus. In particular, while the centralized implementation closely corresponds to the spirit of BizTalk's Transaction Manager and can be written in the join fragment corresponding to coloured nets, the distributed implementation exploits a novel commit protocol, called *Distributed 2-Phase Commit* (D2PC) and exploits reflection for dynamic creation of local transaction managers. Given the correspondence in [15], the distributed interpreter can be directly translated in dynamic nets, but neither in reconfigurable nets, nor in coloured nets.

A hierarchy of transactional frameworks. In this paper, we progressively enrich ZS nets by adding: (1) the value passing mechanism of coloured nets; (2) the dynamic interconnection mechanism of reconfigurable nets; (3) the high-order features of dynamic nets.

In most cases, it is shown that that two-level view of the zero-safe approach is fully preserved, in the sense that, e.g. the abstract net of a coloured ZS net is a coloured P/T net, and so on. Moreover, most constructions are consistent with the obvious embedding derived from the hierarchy, in the sense that, e.g. if we regard a coloured ZS net as a reconfigurable ZS net and take the corresponding abstract reconfigurable P/T net, then we get a coloured P/T net. In other words, the diagram in Figure 1 commutes (vertical arrows are the obvious embedding, while horizontal arrows stand for the construction of abstract nets). We used the word "most", because although we conjecture that the construction of abstract nets can be extended to the dynamic case, at the moment the problem is still open and therefore the *tower* in Figure 1 misses the roof.

For each *layer* of the tower we give several examples for illustrating the main features of the corresponding model. The two main case studies which are presented are the *mobile lessees problem* and the *mailing list*. Regarding the mobile lessees problem, first it is shown that an instance of the problem can always be represented as a ZS net, then it is shown that colours allow for modeling all the instances with a unique coloured ZS net. Regarding the mailing list example, first it is shown that reconfigurable arcs are needed for modeling

$$
\begin{tikzcd}
\text{dynamic ZS nets} \arrow[r, dotted, "?"] & \text{dynamic P/T nets} \\
\text{reconfigurable ZS nets} \arrow[r] \arrow[u, hook] & \text{reconfigurable P/T nets} \arrow[u, hook] \\
\text{coloured ZS nets} \arrow[r] \arrow[u, hook] & \text{coloured P/T nets} \arrow[u, hook] \\
\text{ZS nets} \arrow[r] \arrow[u, hook] & \text{P/T nets} \arrow[u, hook]
\end{tikzcd}
$$

Fig. 1. The hierarchy of transactional frameworks.

dynamic message delivery, and then it is shown that the example can be extended with dynamic creation of new mailing lists by exploiting reflection in dynamic ZS nets.

Structure of the paper. In Sections 2 and 3 are background sections, where we recall the basics of P/T nets and ZS nets. In particular, we define the operational semantics of such models and the notion of a *causal net*, the notion of a *process* and the notion of an *abstract net*, which are later extended to account for colours, reconfiguration and high-order. The modeling of (instances of) the mobile lessees problem is instead original.

Section 4 and 5 contains the original proposals for extending the zero-safe approach to coloured and reconfigurable nets. In both cases, the operational and abstract semantics are defined and related by strong correspondence theorems.

Section 6 attempts to extend the zero-safe approach to dynamic nets. The operational semantics of dynamic ZS nets is presented and discussed on the basis of the mailing list example, whereas the abstract semantics is just informally discussed to put in evidence the difficulties in completing the tower in Figure 1.

Some concluding remarks and future work are in Section 7.

Related work. This part collects pointers to recent approaches to formal methods applied to negotiations for distributed systems. It can be skipped without compromising the reading of the rest of the paper.

Recent works have addressed the extension of the coordination language Linda [23] to express transactions. In particular, the serializability of transactions in *JavaSpaces* [31] have been studied in [17] by adding new primitives for handling traditional flat transactions to Linda. An alternative extension with multiway negotiations is proposed in [14], called TraLinda. The semantics of TraLinda relies on a zero-safe extension of contextual nets. Contextual nets have been previously used in [29] to study the serializability on database transactions.

While aforementioned works are closely related to the classical notion of transactions — "all or none" effect of a transaction is observable — web services languages, such as BPEL [6], WSFL[25], XLANG [32], and its graphical representation Biztalk[30], are particular aware of primitives for handling long-lived negotiations. Such languages do not guarantee atomicity of transactions, but provide programmable *compensations* for undoing actions performed by failed negotiations. The compensation mechanism has been introduced in [22] for designing long-running transactions in database applications.

A compensation language, called StAc, has been proposed in [18]. Processes and compensations in StAc are written in terms of atomic activities. Nevertheless, the interaction among activities is reduced to data sharing and is not described at the top-level (they are hidden on the detailed description of the atomic activities).

In the spirit of process description languages, an extension of the π-calculus with nested compensation contexts has been introduced in [5]. Nevertheless, the extension accounts only for compensation and there is no mechanism to restrict the interactions of transactional processes: the communication capabilities of a process do not change when it runs inside a transactional context. A different approach is taken in cJoin [9] — an extension of the Join calculus with nested, compensatable negotiations — where processes in different transactions can interact by joining their original negotiations into a larger one. Finally, [20] introduces the pike calculus based on *conclaves* (i.e., set of dependent processes) as main abstractions for programming fault-tolerant applications. Different notions of transactions can be modelled in pike by combining such abstractions.

2 Petri Nets

In Petri nets, *places* are repositories of *tokens* (i.e. resources, messages) and *transitions* fetch and produce tokens. We consider an infinite set of resource names $\mathcal{P}$. Given $S \subseteq \mathcal{P}$, we denote with $\wp_{\mathrm{f}}(S)$ the set of all finite subsets of S.

Definition 2.1 (Net). *A* net N *is a 4-tuple* $N = (S_N, T_N, \delta_{0N}, \delta_{1N})$ *where* $S_N \subseteq \mathcal{P}$ *is the (nonempty) set of places,* $a, a', \ldots$*,* T_N *is the set of transitions,* $t, t', \ldots$ *(with* $S_N \cap T_N = \emptyset$*), and the functions* $\delta_{0N}, \delta_{1N} : T_N \to \wp_{\mathrm{f}}(S_N)$ *assign respectively, source and target to each transition.*

We will denote $S_N \cup T_N$ by N, and omit subscript N whenever no confusion arises. We abbreviate a transition $t \in T$ such that $\delta_0(t) = s_1$ and $\delta_1(t) = s_2$ as $s_1[\rangle s_2$, where s_1 is usually referred to as the *preset* of t (written ${}^\bullet t$) and s_2 as the *postset* of t (written $t^\bullet$). Similarly for any place a in S, the preset of a (written ${}^\bullet a$) denotes the set of all transitions with target in a (i.e., ${}^\bullet a = \{t | a \in t^\bullet\}$, and the postset of a (written $a^\bullet$) denotes the set of all transitions with source in a (i.e., $a^\bullet = \{t | a \in {}^\bullet t\}$. Moreover, let ${}^\circ N = \{x \in N | {}^\bullet x = \emptyset\}$ and $N^\circ = \{x \in N | x^\bullet = \emptyset\}$ denote the sets of *initial* and *final elements* of N respectively. A place a is said to be *isolated* if ${}^\bullet x \cup x^\bullet = \emptyset$.

Remark. We consider only nets whose transitions have a non-empty preset, i.e. such that ${}^\circ N \subseteq S$.

$$\frac{m\ [\rangle\ m' \in T \quad m'' \in \mathcal{M}_S}{m \oplus m'' \rightarrow_T m' \oplus m''}\ (\text{FIRING}) \qquad \frac{m_1 \rightarrow_T m_1' \quad m_2 \rightarrow_T m_2'}{m_1 \oplus m_2 \rightarrow_T m_1' \oplus m_2'}\ (\text{STEP})$$

Fig. 2. Operational semantics of P/T nets.

Note that in a net, the target and the source of a transition is a set of states, and thus transitions can consume and produce at most one token in each state. More generally in P/T nets, a transition can fetch and produce several tokens in a particular place, i.e., the pre and postset of a transition are multisets.

Definition 2.2 (Multiset). *Given a set S, a* multiset *over S is a function $m : S \rightarrow \mathbb{N}$. Let $dom(m) = \{a \in S \mid m(a) > 0\}$. The set of all finite multisets (i.e., with finite domain) over S is written $\mathcal{M}_S$. The empty multiset (i.e., with $dom(m) = \emptyset$) is written $\emptyset$. The multiset union $\oplus$ is defined as $(m_1 \oplus m_2)(a) = m_1(a) + m_2(a)$.*

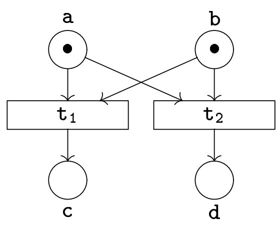
Note that $\oplus$ is associative and commutative, and $\emptyset$ is the identity for $\oplus$. Hence, $\mathcal{M}_s$ is the free commutative monoid $S^\oplus$ over S. We write a for a singleton multiset m such that $dom(a) = \{a\}$ and $m(a) = 1$.

Definition 2.3 (p/t net). *A* marked place / transition Petri net *(P/T net) is a tuple $N = (S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N})$ where $S_N \subseteq \mathcal{P}$ is a set of places, T_N is a set of transitions, the functions $\delta_{0N}, \delta_{1N} : T_N \rightarrow \mathcal{M}_{S_N}$ assign respectively, source and target to each transition, and $m_{0N} \in \mathcal{M}_{S_N}$ is the initial marking.*

The notions of pre and postset, initial and final elements, and isolated places are straightforwardly extended to consider multisets instead of sets.

The operational semantics of P/T nets is given by the inference rules in Figure 2. Given a net N, the proof $m \rightarrow_T m'$ means that a marking m evolves to m' under a *step*, i.e., the concurrent firing of several transitions. Rule FIRING describes the evolution of the state of a net (represented by the marking $m \oplus m''$) by applying a transition $m[\rangle m'$, which consumes the tokens m corresponding to its preset and produces the tokens m' corresponding to its postset. The multiset m'' represents idle resources, i.e. the tokens that persist during the evolution. Rule STEP stands for the parallel composition of computations, meaning that several transitions can be applied in parallel as far as there are enough tokens to fire all of them. We omit the subscript T whenever it is clear from the context. The sequential composition of computations is indicated $\rightarrow^*$, i.e. $m \rightarrow^* m'$ denotes the evolution of m to m' under a (possibly empty) sequence of steps.

Example 2.1 (A simple P/T *net).* Let N be a P/T net s.t. $S = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, $T = \{\mathtt{t}_1, \mathtt{t}_2\}$, $\delta_0(\mathtt{t}_1) = \delta_0(\mathtt{t}_2) = \{\mathtt{a}, \mathtt{b}\}$, $\delta_1(\mathtt{t}_1) = \{\mathtt{c}\}$, $\delta_1(\mathtt{t}_2) = \{\mathtt{d}\}$, $m_0 = \{\mathtt{a}, \mathtt{b}\}$. Figure 3(a) shows the graphical representation of N. As usual, places are represented with circles, transitions with boxes, tokens with dots, and the pre



(a) P/T net N.

$$\dfrac{\dfrac{\mathtt{t}_1 = \mathtt{a} \oplus \mathtt{b}[\rangle \mathtt{c} \in T}{\mathtt{a} \oplus \mathtt{b} \to_T \mathtt{c}}\,\text{FIRING} \qquad \dfrac{\mathtt{t}_2 = \mathtt{a} \oplus \mathtt{b}[\rangle \mathtt{d} \in T}{\mathtt{a} \oplus \mathtt{b} \to_T \mathtt{d}}\,\text{FIRING}}{\mathtt{a} \oplus \mathtt{a} \oplus \mathtt{b} \oplus \mathtt{b} \to_T \mathtt{c} \oplus \mathtt{d}}\,\text{STEP}$$

(b) A computation in N for $\mathtt{a} \oplus \mathtt{a} \oplus \mathtt{b} \oplus \mathtt{b}$

Fig. 3. A simple P/T net.

and postset functions are represented with arcs. Figure 3(b) shows a possible computation in N for the initial marking $\{\mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{b}\}$, which corresponds to the concurrent firing of $\mathtt{t}_1$ and $\mathtt{t}_2$.

3 Zero-Safe nets

In this section we recall the basics of the zero-safe approach by following the presentation given in [11]. Zero-safe nets are an extension of Petri nets suitable to express transactions. Differently from P/T nets, the places of zero-safe nets are partitioned into ordinary and transactional ones (called *stable* and *zero*, respectively). Accordingly to the ordinary terminology, in a '0-safe' net all places cannot contain any token in all reachable markings. *Zero-safe net* — note the word 'zero' instead of the digit '0' — is used to denote that the net contains zero places that cannot contain any token in any observable marking. The role of zero places is to coordinate the atomic execution of complex collections of transitions.

Definition 3.1 (zs net). *A* Zero-Safe net *(*ZS *net) is a 6-tuple $B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B}, Z_B)$ where $N_B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B})$ is the underlying* P/T *net and the set $Z_B \subseteq S_B$ is the set of* zero places*. The places in $S_B \backslash Z_B$ (denoted by L_B) are called* stable places*. A stable marking m is a multiset of stable places (i.e., $m \in \mathcal{M}_{L_B}$), and the initial marking m_{0B} must be stable.*

Note that markings $m \in \mathcal{M}_{S_B}$ can be seen as pairs (s, z) with $m = s \oplus z$, where $s \in \mathcal{M}_{L_B}$ is a stable marking and $z \in \mathcal{M}_{Z_B}$ is the multisets of zero resources, because $\mathcal{M}_{S_B} \simeq \mathcal{M}_{L_B} \times \mathcal{M}_{Z_B}$. Transitions are written $m[\rangle m'$, with m and m' multisets of stable and zero places. A transaction goes from a stable marking to another stable marking. The key point is that stable tokens produced during a transaction are made available only at commit time, when no zero tokens are left. As usual, we omit subscripts when referring to components of a ZS net if they are clear from the context.

$$\text{(FIRING)}\ \frac{s \oplus z [\rangle s' \oplus z' \in T \quad s'' \in \mathcal{M}_{L_B} \quad z'' \in \mathcal{M}_{Z_B}}{(s \oplus s'', z \oplus z'') \rightarrow_T (s' \oplus s'', z' \oplus z'')} \qquad \text{(STEP)}\ \frac{(s_1, z_1) \rightarrow_T (s_1', z_1') \quad (s_2, z_2) \rightarrow_T (s_2', z_2')}{(s_1 \oplus s_2, z_1 \oplus z_2) \rightarrow_T (s_1' \oplus s_2', z_1' \oplus z_2')}$$

$$\text{(CONCATENATION)}\ \frac{(s_1, z) \rightarrow_T (s_1', z'') \quad (s_2, z'') \rightarrow_T (s_2', z')}{(s_1 \oplus s_2, z) \rightarrow_T (s_1' \oplus s_2', z')} \qquad \text{(CLOSE)}\ \frac{(s, \emptyset) \rightarrow_T (s', \emptyset)}{(s, \emptyset) \Rightarrow_T (s', \emptyset)}$$

Fig. 4. Operational semantics of ZS nets.

The operational semantics of ZS nets is defined by the two relations $\Rightarrow_T$ and $\rightarrow_T$ in Figure 4. Rules FIRING and STEP are the ordinary ones for Petri nets, for the execution of one/many transition(s). However, sequences of steps differ from the ordinary transitive closure of $\rightarrow_T$: The rule CONCATENATION composes zero tokens in series but stable tokens in parallel, hence stable tokens produced by the first step cannot be consumed by the second step. Transactions are step sequences from stable markings to stable markings, when CLOSE can be applied. The moves $(s, \emptyset) \Rightarrow_T (s', \emptyset)$ define all the atomic activities of the net, and hence they can be performed in parallel and sequentially as the transitions of an ordinary net. It is worth noting that a step $(s, \emptyset) \Rightarrow_T (s', \emptyset)$ can be itself the parallel composition of several concurrent transactions (by rule STEP).

One of the main advantages of the zero safe approach is that it prevents combinatorial explosion at the specification level. In fact, atomic activities can be defined in terms of several subactivities, which keeps the description of the system small, tractable and modular.

Example 3.1 (The free choice problem). Suppose the net introduced in Example 2.1 (see Figure 3(a)) to code the assignment of two resources a and b either to the activity c or d. By firing t_1 the resources are assigned to c, and by t_2 to d. The nondeterministic choice encoded by the net corresponds to a centralized coordination mechanism that guarantees that both resources are assigned atomically to the same activity. Nevertheless, if one wants to model the system using a *free choice* net, where all decisions are made locally (i.e., by looking just one place) the situation is different. Consider the free choice net shown in Figure 5. It models the system with two independent decisions: one for the assignment of a, the other for the assignment of b.

Nevertheless, the free choice net admits computations not allowed in the abstract system in Figure 3(a). In fact, the free choice net has deadlocks: consider the firing of $\texttt{assign}_{\texttt{a,c}}$ and $\texttt{assign}_{\texttt{b,d}}$. In this case, the net cannot evolve to either b or c, which is a computation not possible in the original net.

ZS nets can be used to overcome this problem, by defining intermediate places as zero places. The assignment problem can be modelled as the ZS net in Figure 6, where smaller circles stand for zero places. The ZS net avoids deadlocks because

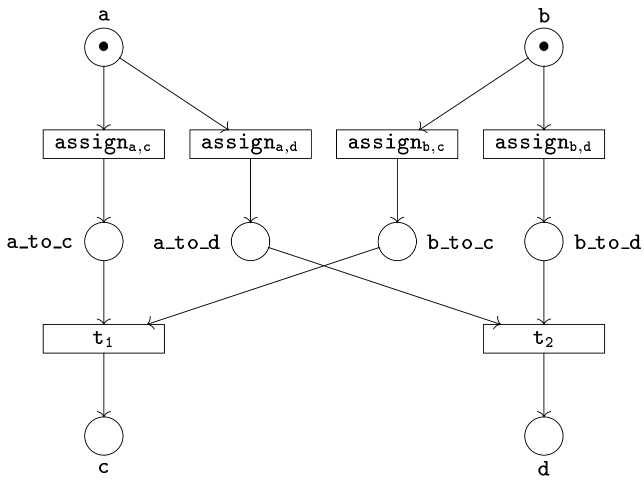


Fig. 5. Free choice net for the assignment problem.

computations ending in markings containing zero tokens are recoverable and not observable.

Example 3.2 (Mobile lessees). The general problem that we want to model consist of a set of apartments that can be rented immediately, a group of people looking for an apartment, and people that want to change their apartments, i.e., they are willing to move to another apartment if somebody else can rent their actual apartments. Consider an instance of the problem with three apartments A, B, C and four people P, Q, R, S. The initial state can be represented as in Figure 7(a), where the apartment A is available for rent, P and S are searching for an apartment, Q wants to leave B, and R wants to leave C. Figure 7(b) shows the preferences of people on apartments.

The formulation of the problem as a ZS net is shown in Figure 8. Note that there is a place for any apartment available for immediate rent in the initial state (`A_free`), a place for any person looking for an apartment (`S_wants` and `P_wants`), and a place for any person willing to change apartment (`Q_changes_B` and `R_changes_C`). There is also a place for any possible rent (i.e., accordingly to the preference matrix in 7(b)). For instance, the transition `S_takes_A` states that person S can rent the apartment A whenever A is free and S is searching for an apartment, and a token in `S_moves_A` means that the person S has rented the apartment A. The more interesting transitions are `Q_leaves_B` and `R_leaves_C`, each of them starts a transaction. In fact, they describe the activity of changing an apartment as the orchestration of two different activities, one in which a person finds a new apartment, and other in which the apartment is rented. For instance, when `Q_leaves_B` is fired a token is produced in `Q_search`, and another in `B_avail`. Note this transaction can finish only when both tokens are consumed,

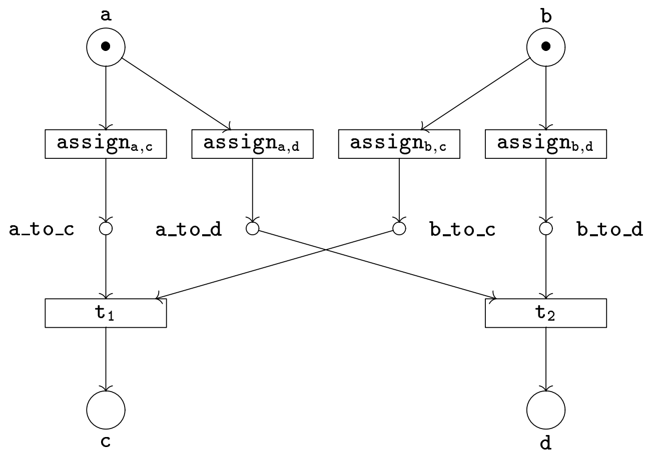


Fig. 6. Free choice net for the assignment problem.

	P	Q	R	S	$Free$
A					•
B		•			
C			•		
$Wants$	•			•	

(a) Initial State.

	P	Q	R	S
A		•		•
B			•	
C	•			

(b) Preferences.

Fig. 7. An instance of the mobile lessees.

meaning that both Q has rented a new apartment and B has been rented. The initial marking denotes the initial state of the problem.

In Figure 9 we show a proof for a transaction in which Q leaves B and takes A, R leaves C and takes B; and P takes B, while S remains without apartment. For space reason, we abbreviate the name of places (i.e. $\mathtt{A_f}$ for `A_free`, $\mathtt{QB_c}$ for `Q_changes_B`, and similarly for the rest). Moreover, we write stable places with capital letters, while zero places are written with lower case. The computation corresponds to the parallel begin of two transactions (where `Q_changes_B` and `R_changes_C` are decomposed into two subactivities) followed by the parallel execution of `Q_takes_A`, `P_takes_C` and `R_takes_B`

3.1 Abstract Semantics

As stated by the operational semantics of ZS nets (Figure 4), the observable states of a system are those represented by stable markings, while the meaningful

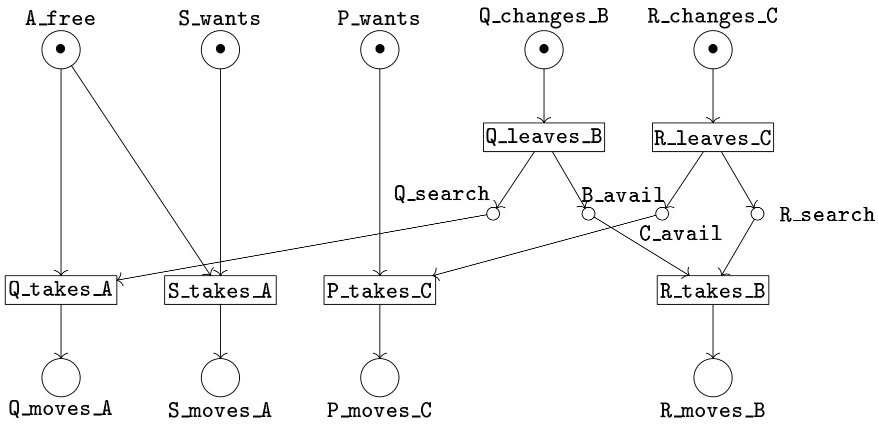


Fig. 8. ZS net of the mobile lessees example.

A:

$$\dfrac{\dfrac{\dfrac{\mathtt{A_f} \oplus \mathtt{q_s} \; [\rangle \; \mathtt{QA_m} \in T}{(\mathtt{A_f}, \mathtt{q_s}) \rightarrow_T (\mathtt{QA_m}, \emptyset)}\,(\mathrm{F}) \quad \dfrac{\mathtt{b_a} \oplus \mathtt{r_s} \; [\rangle \; \mathtt{RB_m} \in T}{(\emptyset, \mathtt{b_a} \oplus \mathtt{r_s}) \rightarrow_T (\mathtt{RB_m}, \emptyset)}\,(\mathrm{F})}{(\mathtt{A_f}, \mathtt{q_s} \oplus \mathtt{b_a} \oplus \mathtt{r_s}) \rightarrow_T (\mathtt{QA_m} \oplus \mathtt{RB_m}, \emptyset)}\,(\mathrm{S}) \quad \dfrac{\mathtt{P_w} \oplus \mathtt{c_a} \; [\rangle \; \mathtt{PC_m} \in T}{(\mathtt{S_w} \oplus \mathtt{P_w}, \mathtt{c_a}) \rightarrow_T (\mathtt{S_w} \oplus \mathtt{PC_m}, \emptyset)}\,(\mathrm{F})}{(\mathtt{S_w} \oplus \mathtt{P_w} \oplus \mathtt{A_f}, \mathtt{q_s} \oplus \mathtt{r_s} \oplus \mathtt{b_a} \oplus \mathtt{c_a}) \rightarrow_T (\mathtt{S_w} \oplus \mathtt{QA_m} \oplus \mathtt{RB_m} \oplus \mathtt{PC_m}, \emptyset) \quad (A)}\,(\mathrm{S})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathtt{QB_c} \; [\rangle \; \mathtt{q_s} \oplus \mathtt{b_a} \in T}{(\mathtt{QB_c}, \emptyset) \rightarrow_T (\emptyset, \mathtt{q_s} \oplus \mathtt{b_a})}\,(\mathrm{F}) \quad \dfrac{\mathtt{RC_c} \; [\rangle \; \mathtt{r_s} \oplus \mathtt{c_a} \in T}{(\mathtt{RC_c}, \emptyset) \rightarrow_T (\emptyset, \mathtt{r_s} \oplus \mathtt{c_a})}\,(\mathrm{F})}{(\mathtt{QB_c} \oplus \mathtt{RC_c}, \emptyset) \rightarrow_T (\emptyset, \mathtt{q_s} \oplus \mathtt{r_s} \oplus \mathtt{b_a} \oplus \mathtt{c_a})}\,(\mathrm{S}) \qquad A}{(\mathtt{S_w} \oplus \mathtt{P_w} \oplus \mathtt{A_f} \oplus \mathtt{QB_c} \oplus \mathtt{RC_c}, \emptyset) \rightarrow_T (\mathtt{S_w} \oplus \mathtt{QA_m} \oplus \mathtt{RB_m} \oplus \mathtt{PC_m}, \emptyset)}\,(\mathrm{CONCAT})}{(\mathtt{S_w} \oplus \mathtt{P_w} \oplus \mathtt{A_f} \oplus \mathtt{QB_c} \oplus \mathtt{RC_c}, \emptyset) \Rightarrow_T (\mathtt{S_w} \oplus \mathtt{QA_m} \oplus \mathtt{RB_m} \oplus \mathtt{PC_m}, \emptyset)}\,(\mathrm{CLOSE})$$

Fig. 9. A proof for the execution of a transaction in the mobile lessees ZS net.

computations (i.e., the atomic activities) are the *stable steps* of the net, i.e. the steps consuming and producing stable markings (relation $\Rightarrow$). Since stable steps can be composed in sequence and parallel, a stable step can be thought of as the execution of several *basic* transactions, i.e., stable steps that cannot be decomposed into other stable steps. Consequently, all the correct behaviours of the system can be derived from the set of basic transactions of the net. The abstract semantics of ZS net is intended to capture the behaviour of a ZS net in terms of its basic transactions.

In this context, a transaction denotes an activity of the system that might be composed by many, possibly concurrent, coordinated subactivities. Since the concurrent semantics of an operational model is usually defined by considering as equivalent all the computations where the same concurrent events are executed in different orders, it follows that we should quotient out those transactions which are equivalent from a concurrent point of view, in such a way that the actual order of execution of concurrent transitions in the ZS net is invisible in the abstract net.

In order to identify the equivalent executions from a concurrent point of view there are two main approaches: the *collective token philosophy* (CTph) and the *individual token philosophy* (ITph). The net semantics under the CTph does not distinguish among different instances of the idealized resources (i.e., tokens). This is a valid interpretation of the behaviour of a system only when any such instance is *operationally* equivalent to all the others. Nevertheless, tokens may have different origins and histories, thus the *causality* information carried on by different tokens is disregarded when identifying equivalent computations w.r.t. CTph, which turns to be the main drawback of this approach. Alternatively, the ITph takes into account the causal dependencies arising in concurrent executions.

The abstract semantics of zero-safe nets has been largely studied under both philosophies. In particular, in both cases the abstract net is characterized by an adjunction on suitable categories [11]. We summarise here the basics of the abstract semantics under the ITph, which is the most significant one. In particular, the distinction between tokens with different origins and history relies on the notion of a deterministic process. A deterministic process denotes a particular computation in a net, and therefore it explicitly carries the causal information between firings. To define processes we need two other concepts: net morphisms and causal nets.

Definition 3.2 (Net morphism). *Let N, N' be P/T nets. A pair $f = (f_S : S_N \to S_{N'}, f_T : T_N \to T_{N'})$ is a* net morphism *from N to N' (written $f : N \to N'$) if $f_S(\delta_{iN}(t)) = \delta_{iN'}(f_T(t))$.*

We usually omit subscripts when they are clear from the context. With abuse of notation we apply functions (i.e., f_S) over (multi)sets, meaning the multiset obtained by applying the function element-wise: $f_S(\{m_0, \ldots, m_n\}) = f_S(m_0) \oplus \ldots \oplus f_S(m_n)$.

Definition 3.3 (Causal Net and Process). *A net $K = (S_K, T_K, \delta_{0K}, \delta_{1K})$ is a* causal net *(also called* deterministic occurrence net*) if it is* acyclic *and $\forall t_0 \neq t_1 \in T_K, \delta_{iN}(t_0) \cap \delta_{iN}(t_1) = \emptyset$, for $i = 0, 1$.*

A (Goltz-Reisig) process *for a P/T net N is a net morphism P from a causal net K to N.*

Two processes P and P' of N are *isomorphic* and thus equivalent if there exists a net isomorphism $\psi : K_P \to K_{P'}$ such that $\psi; P' = P$.

Given a process $P : K \to N$, the set of *origins* and *destinations* of P are defined as $O(P) = {}^{\circ}K$ and $D(P) = K^{\circ} \cap S_K$, respectively. We write $pre(P)$ and $post(P)$ for the multisets denoting the initial and final markings of the process, i.e. $pre(C) = P(O(P))$ and $post(C) = P(D(P))$. Moreover, as isomorphisms respect initial and final markings, we say that $O(\xi) = pre(P)$, $D(\xi) = post(P)$, for $\xi = [\![P]\!]_{\approx}$. Finally, the set of *evolution places* of a process P is the set $E_P = \{P(a) | a \in K, |{}^{\bullet}a| = |a^{\bullet}| = 1\}$.

Definition 3.4 (Connected transaction). *Given a* ZS *net B, let P be a process of the underlying* P/T *net N_B. The equivalence class $\xi = [\![P]\!]_{\approx}$ is a* connected transaction *of B if:*

- *$pre(P)$ and $post(P)$ are stable markings, i.e., the process starts by consuming stable tokens and produces only stable tokens;*
- *$E_P \subseteq Z_B$, i.e. stable tokens produced during the transaction cannot be consumed during in the same transaction;*
- *P is connected, i.e. the set of transitions T_K is non-empty, and for all $t_0, t_1 \in T_K$ there exists an undirected path connecting t_0 and t_1; and*
- *P is full, i.e., it does not contain idle (i.e., isolated) places (i.e., $\forall a \in S_K, |{}^\bullet a| + |a^\bullet| \geq 1$).*

We denote by Ξ_B (ranged by ς) the set of connected transaction of B.

A connected transaction can be executed when the state of the net contains enough stable tokens to enable all the transitions independently. At the end of its execution no token may be left on zero places (nor may be found on them at the beginning of the step). This means that all the zero tokens produced by a transaction are also consumed by the same transaction. Moreover, in a connected transaction no intermediate marking is stable.

Definition 3.5 (Causal abstract net). *Let $B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B}, Z_B)$. The net $I_B = (S_B \backslash Z_B, \Xi_B, \delta_{0I}, \delta_{1I}, m_{0B})$, with $\delta_{0I}(\varsigma) = pre(\varsigma)$ and $\delta_{0I}(\varsigma) = post(\varsigma)$, is the* causal abstract net *of B (we recall that $pre(\varsigma)$ and $post(\varsigma)$ denote the multisets $P_\varsigma(O(\varsigma))$ and $P_\varsigma(D(\varsigma))$, respectively, and that Ξ_B is the set of all the connected transactions of B).*

Example 3.3 (Abstract Net for the Mobile lessees problem). Figure 10 shows the abstract net corresponding to the ZS net in Figure 8. In the abstract net there are only two transitions, each of them representing an abstract transaction of the ZS net: `S_takes_A`, corresponding to the homonymous transition in the ZS net; `Q_takes_A` & `P_takes_C` & `R_takes_B`, for the atomic negotiation in which Q leaves B and takes A, R leaves C and takes B; and P takes B. These two transitions are enough to model the abstract behaviour of the system. In fact, any other combination is not possible because it would imply that some exchanged apartment (i.e., B or C) remains available for rent or a person willing to change apartment (Q or R) remains without apartment, which is an inconsistent state (with pending negotiations).

Note that one of the main advantages of the approach is that it allows to fully specify the behaviour of a system without analyzing all possible global combinations. Consider an instance of the lessees problem with a larger number of apartments and people, and a more complicated set of preferences. It could be tedious to figure out which are all the possible combinations that correspond to consistent transformations in the system. Moreover, it is possible to describe an infinite abstract net with a finite ZS net, as the multicasting system presented in [11] or the generalized version of the mobile lessees problem analyzed in Section 4.3.

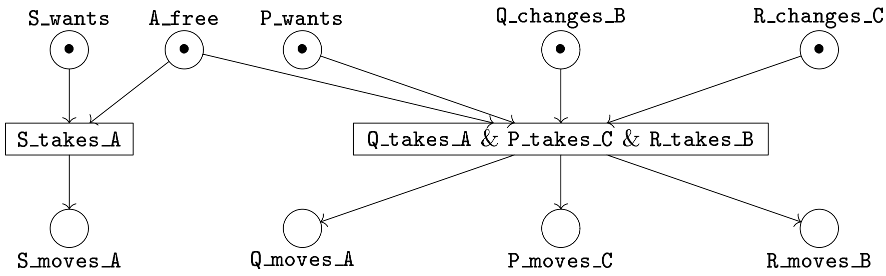


Fig. 10. Abstract net of the mobile lessees example.

The correspondence between the concrete and the abstract view is stated by the following theorem [12].

Theorem 3.1. *Let B be a* ZS *net and I_B its abstract net. Then $m \rightarrow_{T_{I_B}} m'$ iff $m \Rightarrow_{T_B} m'$.*

4 Adding colours to zs nets

4.1 Coloured p/t nets

In coloured P/T nets [24] (known also as *high-level* nets), tokens carry on information, which is given by their *colours*. Actually, colours are values/data associated with a particular instance of a resource. Hence, the state and transitions of a net exploit also the information present in tokens, i.e. their colours.

We consider an infinite set of constant colour names $\mathcal{B}$, ranged over by $x, x_1, \ldots$ and an infinite set of colour variables $\mathcal{V}$, ranged over by $v, w, \ldots$. We denote the set of constants and variables with $\mathcal{C} = \mathcal{B} \cup \mathcal{V}$. Moreover, we require constant and variable colours to be disjoint ($\mathcal{B} \cap \mathcal{V} = \emptyset$) and different from place names, i.e. $\mathcal{C} \cap \mathcal{P} = \emptyset$. Let S be a set, S^* stands for the set of all finite (possible empty) sequences on S, i.e. $S^* = \{(s_1, \ldots, s_n) | n \geq 0 \wedge s_i \in S\}$. The empty sequence is denoted by $\bullet$, and the underlying set of a sequence $(s_1, \ldots, s_n)$ by:

$$\overline{(s_1, \ldots, s_n)} = \bigcup_i \{s_1\}$$

Definition 4.1 (Coloured net). *A* coloured net *N is a 5-tuple $N = (S_N, C_N, T_N, \delta_{0N}, \delta_{1N})$ where $S_N \subseteq \mathcal{P}$ is the (nonempty) set of places, $C_N \subseteq \mathcal{C}$ is the set of colours, T_N is the set of transitions (with $S_N \cap T_N = \emptyset$), and the functions $\delta_{0N}, \delta_{1N} : T_N \rightarrow \wp_{\mathrm{f}}(S_N \times C_N^*)$ assign respectively, source and target to each transition. To assure that a transition fetches and produces at most one token in a place we require that $\forall t \in T_N, if (s, c_1), (s, c_2) \in \delta_{iN}(t)$ then $c_1 = c_2$, for $i = 0, 1$.*

The pre and postset of a transition are defined similarly to Section 2, but taking into account that they are coloured sets instead of sets. Analogously, for any place a in S_N, the preset of a (written ${}^\bullet a$) denotes the set of all transitions with target in a (i.e., ${}^\bullet a = \{t | (a, c) \in t^\bullet\}$, and the postset of a (written $a^\bullet$) denotes the set of all transitions with source in a (i.e., $a^\bullet = \{t | (a, c) \in {}^\bullet t\}$. The definitions for the sets of *initial* and *final elements*, and *isolated* place are identical to those given in Section 2.

Note that in coloured nets a transition $m_1 [\rangle m_2$ denotes a pattern that should be matched/instantiated with appropriated colours in order to be applied. In particular, constant colours appearing in m_1 act as values that should be matched in order to fire the transition, while variables should be instantiated with appropriate colours. Variables are binders of colours occurring in m_2. For instance, the transition $t = (a_1, v), (a_2, v), (a_3, x_1)\ [\rangle (a_1, v), (a_4, x_2)$ denotes a pattern stating that whenever a_1 and a_2 contain tokens with the same colour (but they can be of any constant colour because of the variable v) and a_3 contains a token with constant colour x_1, the transition can be fired. When t is fired, the tokens matching the preset are consumed, and a new token is put in a_1, whose colour corresponds to the consumed tokens in a_1 and a_2, and a token with colour x_2 is produced in a_4. Consequently, the firing of t over $m = (a_1, x_3), (a_2, x_3), (a_3, x_1)$ will produce $m' = (a_1, x_3), (a_4, x_2)$. From a functional point of view, colour variables occurring in the preset of a transition act as its parameters, which are called *received colours.*

Definition 4.2 (Received colours of a transition). *The* colour of a set $s \subseteq \mathcal{S} \times \mathcal{C}^*$ *is defined as* $col(s) = \cup_{(a,c) \in s} \overline{c}$*, the* set of constants *is* $col_{\mathcal{B}}(s) = col(s) \cap \mathcal{B}$*, and the* set of variables $col_{\mathcal{V}}(s) = col(s) \cap \mathcal{V}$*. Given a transition* $t = m[\rangle m'$*, the set of* received colours *(also* received names*) of* t *is given by* $rn(t) = col_{\mathcal{V}}(m)$*.*

Remark. As variables are used to describe parameters in a transition, we will consider only coloured nets in which each transition $t = m[\rangle m'$ satisfies $col_{\mathcal{V}}(m') \subseteq rn(t)$. This restriction states that all variables occurring in the postset of a transition are bound to some variable in the preset.

Clearly, previous definitions can be straightforwardly extended to consider coloured multisets instead of sets.

Definition 4.3 (Coloured Multiset). *Given two sets* S *and* C*, a* coloured multiset *over* S *and* C *is a function* $m : S \to C \to \mathbb{N}$*. Let* $dom(m) = \{(s, c) \in S \times C \mid m(s)(c) > 0\}$*. The set of all finite multisets over* S *and* C^* *is written* $\mathcal{M}_{S,C}$*. The multiset union is defined as* $(m_1 \oplus m_2)(s)(c) = m_1(s)(c) + m_2(s)(c)$*. We write* $s(c)$ *for a multiset* m *such that* $dom(m) = \{(s, c)\}$ *and* $m(s)(c) = 1$*. Additionally,* $(s, c) \in m$ *is a shorthand for* $(s, c) \in dom(m)$*, while* $s \in m$ *means* $(s, c) \in m$ *for some* c*.*

Definition 4.4 (c-p/t net). *A* coloured marked place / transition net *(*C-P/T *net) is a 6-tuple* $N = (S_N, C_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N})$ *where* $S_N \subseteq \mathcal{P}$ *is the set of places,* $C_N \in \mathcal{C}$ *is the set of colours,* T_N *is a set of transitions, the functions* $\delta_{0N}, \delta_{1N} : T \to \mathcal{M}_{S_N, C_N}$ *assign respectively, source and target to each transition,* *and* $m_{0N} \in \mathcal{M}_{S_N,C_N}$ *is the initial marking. Moreover,* $\forall t \in T_N, col_{\mathcal{V}}(t^\bullet) \subseteq rn(t)$ *(i.e., variables in the postset are bound to received names), and* $col(m_{0N}) \subseteq \mathcal{B}$ *(i.e., tokens in the initial marking are coloured with constants).*

As aforementioned, the firing of a transition t in a coloured net requires to instantiate t with appropriate colours, i.e., those corresponding to tokens present in places. Consequently, the instantiation of a transition corresponds to a substitution on colour variables.

Definition 4.5 (Substitution on colours). *Let* $\sigma : \mathcal{V} \to \mathcal{B} \cup \mathcal{V}$ *be a partial function. The* substitution $v\sigma$ on a colour variable v *is* c *if* $\sigma(v) = c$*, otherwise it is* v*, i.e., it is the identity when* σ *it is not defined. Instead, the* substitution $x\sigma$ on a constant colour x *produces* x*, i.e., it has no effect. The* substitution on a colour sequence *is the simultaneous substitution on the names appearing in the sequence, i.e.,* $(c_1, \ldots, c_n)\sigma = (c_1\sigma, \ldots, c_n\sigma)$*. The* colour substitution on a multiset $m \in \mathcal{M}_{S,C}$ *is given by* $(m \star \sigma)(s)(c) = \sum_{d\sigma=c} m(s)(d)$.

The operational semantics of coloured nets is given by replacing the rule FIRING in Figure 2 by the following version:

$$\text{(COLOURED-FIRING)}\quad \frac{t = m\ [\rangle\ m' \in T \quad m'' \in \mathcal{M}_{S,C}}{m \star \sigma \oplus m'' \to_T m' \star \sigma \oplus m''}\quad dom(\sigma) = rn(t) \text{ and } \sigma(v) \in \mathcal{B} \text{ for } v \in dom(\sigma)$$

Remark: α-equivalence on defined names. Note that the variables chosen to denote colours in the preset of a transition are meaningless. Actually, they act as binders whose scope is just *that* transition, and consequently the can be changed without modifying the meaning of a transition. Therefore we define the following relation over transitions, called α-conversion on received colours.

Two transitions $t_1 = m_1[\rangle m_1'$ and $t_2 = m_2[\rangle m_2'$ are α-convertible if there exists an injective substitution $\sigma : \mathcal{V} \to \mathcal{V}$, where $dom(\sigma) \subseteq rn(t_1)$, such that $m_1 \star \sigma = m_2$ and $m_1' \star \sigma = m_2'$. The α-conversion is an equivalence relation, which is denoted by $\equiv_\alpha$. We usually talk about transitions up-to α-equivalence.

4.2 Coloured zs nets

The ZS version of a coloured net is obtained also by distinguishing stable places from zero ones.

Definition 4.6 (c-zs net). *A* coloured ZS net *(*C-ZS *net for short) is a 7-tuple* $B = (S_B, C_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B}, Z_{0B})$ *where* $N_B = (S_B, C_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B})$ *is the underlying* C-P/T *net and the set* $Z_B \subseteq S_B$ *is the set of* zero places*. The places in* $S_B \backslash Z_B$ *(denoted by* L_B*) are called* stable places*. A stable marking* m *is a coloured multiset of stable places (i.e.,* $m \in \mathcal{M}_{L_B,C_B}$*), and the initial marking* m_{0B} *must be stable and satisfy* $col(m_{0B}) \subseteq \mathcal{B}$.

The operational semantics of C-ZS nets is a straightforward extension of rules given in Figure 4, where the firing rule is replaced by the following version:

$$\text{(COLOURED-FIRING)}\quad \frac{t = s \oplus z \; [\rangle \; s' \oplus z' \in T \quad s'' \in \mathcal{M}_{L,C} \quad z'' \in \mathcal{M}_{Z,C}}{(s \star \sigma \oplus s'', z \star \sigma \oplus z'') \rightarrow_T (s' \star \sigma \oplus s'', z' \star \sigma \oplus z'')} \quad \begin{array}{l} dom(\sigma) = rn(t), \text{ and} \\ \sigma(v) \in \mathcal{B} \text{ for } v \in dom(\sigma) \end{array}$$

We still write a marking $m = s \oplus z$ as a pair (s, z) to denote that $s \in \mathcal{M}_{L,C}$ and $z \in \mathcal{M}_{Z,C}$.

Example 4.1 (C-ZS net for the mobile lessees problem). A more general representation for the mobile lessees problem introduced in the Example 3.2 can be given in terms of C-ZS nets. Consider the net in Figure 11, where label on arcs corresponds to the colours of the pre and postset of a transition. Tokens present in the place `free` represent apartments that are available for being rented immediately. The actual identity of the apartment is given by the colour of the token. Similarly people looking for an apartment are represented as coloured tokens in `wants`, and those willing to change apartment as tokens (w', v') in `changes`, meaning that the person w' changes the apartment v'. The transition `freeing` initiates a transaction by making available for rent an offered apartment. Analogously, `searching` initiate a transaction in which a person is looking for an apartment. Transition `changing` starts a transaction by rendering available the offered apartment and putting a token in the place of persons looking for an apartment. Finally, the transition `taking` states that a person w'' searching for an apartment can take the available apartment v'' if she likes it (i.e., a token with colour (w'', v'') is in the set of preferences). A token with colour (w'', v'') produced in the place `moves` means that the person w'' has moved to the apartment v''. It is worth noting that tokens are actually produced on place `moves` when no token is left in the zero places (i.e., `avail` or `search`).

While in the ZS net different instances of the problem (i.e., different set of apartments, people or preferences) correspond to different structures of the net (i.e., states, transitions and flow function), in the coloured version the structure is the same for every instance of the problem, the only thing that changes is the initial marking. In fact, all the information about a particular instance of the problem is represented by colours.

Contextual nets. The self-loops introduced to model the preference sets in the Example 4.1 can be better modelled as *read arcs.* Nets with read arcs allow for modelling "read without consume", where many readers can access concurrently the same resource. Consider transition `taking` in Figure 11. Actually, there is no need to consume the token in `pref`. To fire `taking` it is enough to check the presence of a token with suitable colours on `pref`. The extension of the ZS model to contextual nets have been studied in [13].

ZS *nets can be encoded as* C-P/T *nets.* ZS nets have been encoded in [7] into a fragment of the join calculus corresponding to the coloured nets. In such encoding

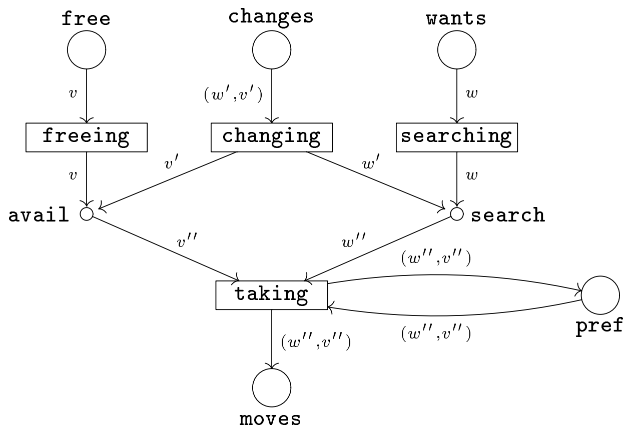


Fig. 11. Coloured ZS net of the mobile lessees example.

(which is called flat) the transactional mechanism of ZS nets is implemented through a centralized coordinator, which is aware of the zero tokens present in the net. Roughly, stable tokens produced during a transaction are kept frozen by the coordinator, which will release them when no zero token is left in the net.

4.3 Abstract net under the ITph approach

In order to define the abstract net associated to a C-ZS net we revise the notion of causal net and processes in order to take into account colours.

Definition 4.7 (Coloured Causal Net). *A coloured net $K = (S_K, C_K, T_K, \delta_{0K}, \delta_{1K})$ is a* causal net *if it is* acyclic *and transitions do not share places in their pre and postsets, i.e. if $a \in \delta_{iN}(t_0)$ and $a \in \delta_{iN}(t_1)$ then $t_0 = t_1$.*

When viewing coloured causal nets as descriptions of runs, it should be clear that differently from causal nets, in the coloured version a causal net can be blocked because the bindings between the different colours are not consistent. Consider a simple causal net with the following transitions $t_0 = (a_0, \bullet)[\rangle(b_0, x_0)$, $t_1 = (a_1, \bullet)[\rangle(b_1, x_1)$, and $t_2 = (b_0, v), (b_1, v)[\rangle(a, \bullet)$. Starting with the marking $a_0(\bullet) \oplus a_1(\bullet)$, it cannot execute completely because t_0 produces a token with the constant colour x_0 on b_0 and t_1 a token with colour x_1 on b_1, but t_2 requires tokens in b_0 and b_1 with the same colour.

In general, causal nets can execute completely when the colours used to label transitions are sequences of variables of the same length.

Definition 4.8 (Plain nets). *A causal net K is a* plain net *if $\exists k \in \mathbb{N}$ s.t. $\forall t \in T, (s,c) \in {}^{\bullet}t \cup t^{\bullet} : \overline{c} \subseteq \mathcal{V} \wedge |c| = k$, with $|c|$ denoting the length of c.*

As for P/T net, we define a notion of morphism between C-P/T nets.

Definition 4.9 (Coloured net morphism). *Let N, N' be* C-P/T *nets. A tuple $f = (f_S : S_N \to S_{N'}, f_T : T_N \to T_{N'}, \sigma = \{\sigma_t\}_{t \in T_N})$ is said a* coloured net morphism *from N to N' (written $f : N \to_\sigma N'$) if $f_S({}^\bullet t)[\rangle f_S(t^\bullet) = {}^\bullet f_T(t) \star \sigma_t[\rangle f_T(t)^\bullet \star \sigma_t$.*

Note that a morphism explains also the correspondence between colours used by transitions. Each σ_t relates each colour appearing in t with a colour in $f_T(t)$. Moreover, transitions in N are required to be a particular case of those in N'.

Definition 4.10 (Process of a coloured net). A (Goltz-Reisig) process *for a* C-P/T *net N is coloured net morphism $P : K \to_\sigma N$, from a coloured causal net K to N, s.t. $\forall \sigma_t \in \sigma$, σ_t is injective and $\sigma_t : \mathcal{V} \to \mathcal{V}$.*

Two coloured processes P and P' are *isomorphic* and thus equivalent if there exists a net isomorphism $\psi : K_P \to_\sigma K_{P'}$ such that $\psi; P' = P$.

A process P associates a coloured causal net K to a C-P/T net N. As K is itself a coloured net, its transitions can be fired for any suitable substitution of colours. Therefore, a process describes several runs that start from initial markings with different colours. (Our approach is similar to that presented in [3]). Nevertheless, this does not mean that a process can be instantiated for any possible combination of colours. In fact, a process stands for executions where just one token is produced and/or consumed from a particular place, and consequently the colours appearing in the preset and postset of a place must coincide. Consequently, a process implicitly defines a relation among colours in admissible markings. A compatible execution of a process is an instantiation of colours that satisfies such constraints.

For simplicity, we defined the notion of compatible execution for the equivalence class of P, i.e. $[\![P]\!]_\approx$.

Definition 4.11 (Compatible execution of $[\![P]\!]_\approx$). *Let $\varsigma \in [\![P]\!]_\approx$ such that $\forall t_1, t_2 \in T_\varsigma, rn(t_1) \cap rn(t_2) = \emptyset$, i.e., transitions do not share variables. A substitution σ is said a* compatible execution *of ς if $\forall a \in S_\varsigma, {}^\bullet a \star \sigma = a^\bullet \star \sigma$. If such σ exists, we say that $[\![P]\!]_\approx$ is compatible. A process P is* compatible *if there exists a compatible execution for $[\![P]\!]_\approx$.*

A compatible execution captures the notion of unification that takes place when computing in a coloured net.

Example 4.2. Two simple processes for the C-ZS net in Figure 11 are presented in Figure 12. The first one represents a person looking for an apartment that takes a free apartment, while the second shows the process in which two people exchange their apartments.

The first process can be used as representative of its equivalence class because its transitions do not share variables. The substitution $\sigma = \{v/v', w/w'\}$ is a compatible execution for the first process. Note that w/w' in σ captures the idea that the token consumed from the state `wants` refers to the same person of the token used from the preference set (similarly, v/v' relates the different

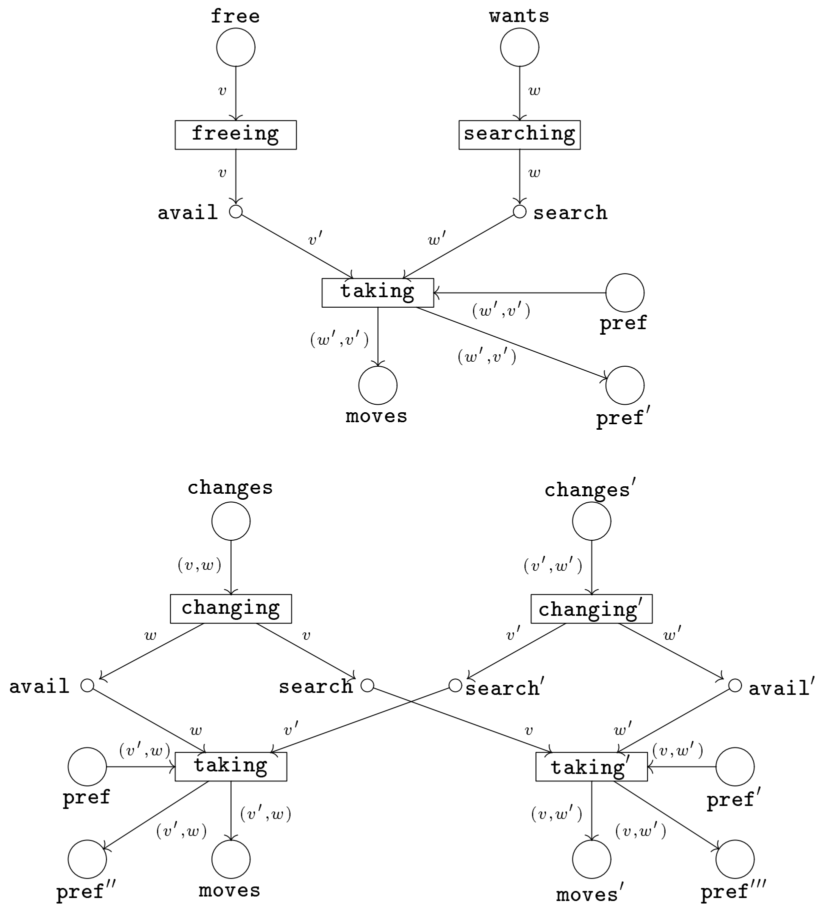


Fig. 12. Two coloured processes for the mobile lessees example.

tokens referring to the same apartment). Observe also that the substitution $\sigma' = \{v/x, v'/x, w/x, w'/x\}$ is a compatible execution for the same process, which requires all names to be equals to the constant x. Clearly, σ' imposes more restrictive constraints than σ. Moreover, σ' is a particular case of σ.

On the other hand, the second process cannot be taken as representative of its equivalence class because some transitions, such as $\texttt{changing}'$ and $\texttt{taking}$, share variables. Nevertheless, a representative can be obtained by applying α-conversion on transitions.

Proposition 4.1. *Any process P of a plain net is compatible.*

As we are interested on capturing the most general definition for equivalent executions, we will associate particular cases to instantiations of more general ones. Consequently, we are interested on the less restrictive constraints on colours implied by a process, which is called the *most general compatible execution.*

Definition 4.12 (mgce). *A compatible execution σ is said the* most general compatible execution *(shorten as* mgce*) of ς, written σ_ς, if for every other compatible execution σ' there exist a substitution γ s.t. $\forall t \in T_\varsigma, (t \star \sigma_\varsigma) \star \gamma = t \star \sigma'$.*

We write by Ξ_B (ranged by ς) the set of connected transaction of B.

The definition for connected transactions is identical to Definition 3.4, but requiring processes to be compatible. Consequently, the definition of the abstract net is immediate, the only difference is that abstract transitions are defined in terms of the processes and their mgce.

Definition 4.13 (Causal abstract coloured net). *Let $B = (S_B, C_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B}, Z_B)$. The net $I_B = (S_B \backslash Z_B, C_B, \Xi_B, \delta_{0I}, \delta_{1I}, m_{0B})$, with $\delta_{0I}(\varsigma) = pre(\varsigma) \star \sigma_\varsigma$ and $\delta_{0I}(\varsigma) = post(\varsigma) \star \sigma_\varsigma$, is the* causal abstract net *of B (we recall that σ_ς is the mgce for ς, that $pre(\varsigma)$ and $post(\varsigma)$ denote the multisets $P_\varsigma(O(\varsigma))$ and $P_\varsigma(D(\varsigma))$, respectively, and that Ξ_B is the set of all the compatible connected transactions of B).*

Example 4.3 (Abstract coloured net for the generalized mobile lessees problem). Figure 13 shows a partial view of the abstract net corresponding to the mobile lessees example. Transition `wants&free` corresponds to the atomic step in which a person who is searching for an apartment rents an available apartment. Transition `n changes` corresponds to the case in which n people interchange their apartments. Note there are infinite transitions of this kind, one for any $n \geq 2$. Similarly, the transition `n changes&wants&free` stands for the atomic step in which n people change their apartments, but one of them takes a free apartment and one person looking for an apartment participates in the exchange. Observe that this infinite abstract net is modelled with a finite concrete ZS net.

Finally, the correspondence between the two different views provided by the concrete ZS net and the abstract net is guaranteed by the following result.

Theorem 4.1. *Let B be a C-ZS net and I_B its abstract net. Then $m \rightarrow_{T_{I_B}} m'$ iff $m \Rightarrow_{T_B} m'$.*

Proof (sketch). $\Rightarrow$) By induction on the structure of the proof $m \rightarrow_{T_{I_B}} m'$. (i) When the reduction is obtained by applying rule FIRING, then there is a transition $m_1[\rangle m_1'$ in I_B s.t. $m_1 \star \sigma \oplus m'' = m$ and $m_1' \star \sigma \oplus m'' = m'$, i.e., m_1 is consumed, m_1' is produced, and m'' denotes idle resources. Consequently, by the construction of the abstract net there is a connected transaction (a compatible process) ς with a mgce σ_ς s.t. $pre(\varsigma) \star \sigma_\varsigma = m_1$ and $post(\varsigma) \star \sigma_\varsigma = m_1'$. We can build a proof for $m_1 \star \sigma_\varsigma \star \sigma \Rightarrow_B m_1' \star \sigma_\varsigma \star \sigma$ using ς in the following way: at each step use rules COLOURED-FIRING and STEP for firing all enable transitions, then combine steps with rule CONCATENATION. The concatenation rule can always be applied because the evolution places of a concatenable transaction are zero-places. The family of substitutions σ_t used by the morphism explains how variables appearing

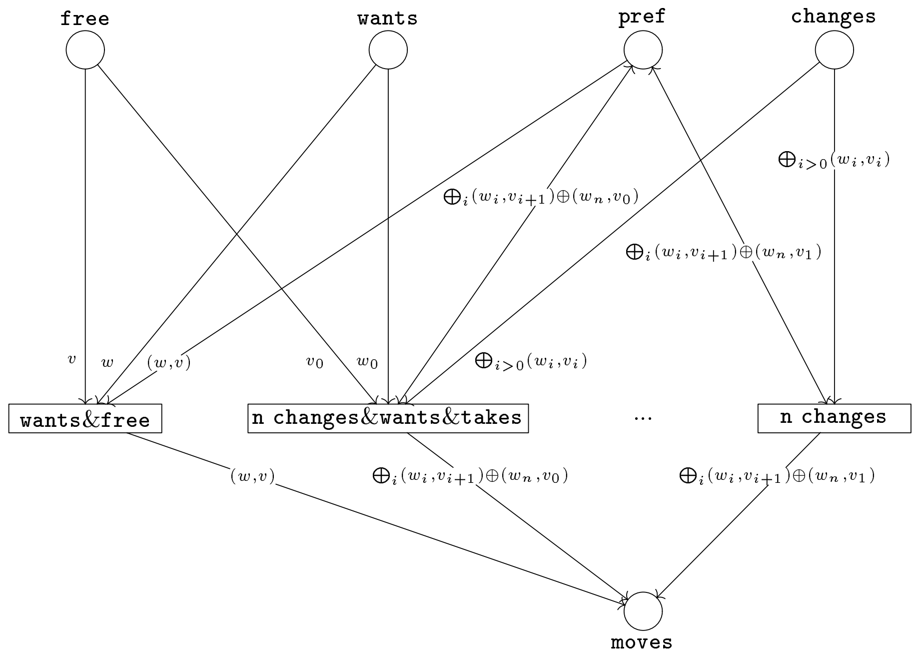


Fig. 13. Partial abstract net of the C-ZS net of the mobile lessees example.

in the transitions (of the net B) are used in the process. Consequently, any substitution used in the proof to fire a transition t is defined as $\sigma_t \star \sigma_\varsigma \star \sigma$ restricted to $rn(t)$. Note that proof obtained by adding idle resources in the application of FIRING is also a valid rule. Consequently, the idle resources m'' can always be added to the computation described by the process. (ii) When the reduction is obtained by applying rule STEP the proof is immediate by using inductive hypothesis on premises and by noting that steps in the abstract net corresponds also to steps in the ZS net.

$\Leftarrow$) Note that it is possible to define a process P describing the computation $m \Rightarrow_{T_B} m'$, s.t. $pre(P) = m$, $post(P) = m'$. Moreover, the causal net used by P contains a place for each produced token in the proof, and a transition for any application of FIRING. Note that by rule CONCATENATION all evolution places corresponds to zero places. If two independent computations are combined with rule STEP, then the process has independent subnets, each of them is a process from a stable marking to a stable marking. So, they can be considered independently. Obviously, each independent process is compatible, because it is a possible computation of the net (the compatible execution can be built from the substitutions used during the proof). Therefore the abstract net contains a transition representing this process. This is guaranteed because transitions in the abstract net are defined in terms of the mgce (i.e., any compatible instantiation of the processes can be obtained as an instantiation the mgce). Consequently, there is a firing corresponding to any independent subnet in the process. The entire computation in the abstract net can be obtained by using rule STEP to combine concurrent firings. Isolated places in the causal net are idle resources and can be added to any firing in the proof.

ZS *nets as* C-ZS *nets.* P/T (and ZS) nets can be seen as a particular case of C-P/T (resp., C-ZS) nets where tokens are coloured with the empty sequence $\bullet$.

Definition 4.14 (Coloured version of a p/t net). *Let $N = (S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N})$ be a* P/T *net. The coloured net $C_N = (S_N, \emptyset, T_N, \delta_{0C_N}, \delta_{1C_N}, m_{0C_N})$, with $\delta_{iC_N}(t)(a, \bullet) = \delta_{iN}(t)(a)$ for $i = 1, 2$ and $m_{0C_N}(a, \bullet) = m_{0N}(a)$ is the* coloured version *of N. Given a* ZS *net B, the* C-ZS *net C_B is coloured version B if its underlying* C-P/T *net N_{C_B} is the coloured version of the underlying* P/T *net N_B of B, and $Z_B = Z_{C_B}$.*

It should be noted that the construction of the abstract nets under these two different views is consistent. That is, the coloured abstract net for a ZS net coincides with the abstract (non-coloured) net.

Theorem 4.2. *Let B a* ZS *net, I_B its abstract* P/T *net, C_B and C_{I_B} their coloured versions, and I_{C_B} the abstract net of C_B (i.e., the colored version of B). Then $C_{I_B} \approx I_{C_B}$.*

Proof (sketch). The proof follows from the fact that the coloured version C_P of a process P of a net N is a process of the coloured net C_N. Moreover, if $P \approx P'$ then $C_P \approx C'_P$. On the other hand, the coloured version of P is always a plain net, because all colours are sequences of length 0. Therefore any coloured process C_P is compatible, and consequently $C_{I_B} \approx I_{C_B}$.

5 Reconfigurable zs nets

5.1 Reconfigurable nets

The idea behind reconfigurable nets (R-P/T nets) is that basic colours are names of places in the net, and consequently the postset of a transition is not static, but depends on the colours of the consumed tokens. For instance, a transition $t = a(v)[\rangle v(a)$ denotes a pattern that consumes a token from a and generates a token in the place corresponding to the colour v of the consumed message. In fact, if t is applied to $m_1 = a(b)$ it produces $m_2 = b(a)$. Instead, when applied to $m'_1 = a(b')$, it generates $m'_2 = b'(a)$.

Consequently, the definitions of nets and of place/transitions nets can be extended in order to allow received names to appear as places in the postsets of transitions. We consider an infinite set of variable names $\mathcal{V}$, ranged over by $v, w, \ldots$. We require also variable names be different from place names, i.e., $\mathcal{V} \cap \mathcal{P} = \emptyset$. Moreover, the constant colours are names of places, hence $\mathcal{C} = \mathcal{P}$.

Definition 5.1 (r-p/t net). *A* Reconfigurable marked place / transition net *is a 5-tuple $N = (S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N})$ where $S_N \subseteq \mathcal{P}$ is a set of places, T_N is a set of transitions, the functions $\delta_{0N} : T_N \to \mathcal{M}_{S_N, S_N \cup \mathcal{V}}$ and $\delta_{1N} : T_N \to \mathcal{M}_{S_N \cup \mathcal{V}, S_N \cup \mathcal{V}}$ assign respectively, source and target to each transition, and $m_{0N} \in \mathcal{M}_{S_N, S_N}$ is the initial marking. Moreover, for every t in T_N we require $\delta_{1N}(t) \subseteq \mathcal{M}_{S_N \cup rn(t), S_N \cup rn(t)}$, i.e., variables occurring in the postset of a transition are received names.*

Note that we allow variables to occur in the preset of a transition just in colour positions, while they can also occur in place positions in the postsets. Variables are used analogously to variables in the coloured model, i.e., they are the parameters of a transition that should be instantiated in order to fire the transition. As usual, we consider transitions up-to α-conversion.

The main difference between coloured and reconfigurable nets is that when a transition t is fired in a R-P/T net, the variables in the postset of t should be substituted also when they appear in place position. The following definition introduces the substitution of names occurring both in colour and place position.

Definition 5.2 (Substitution). *Let $\sigma : \mathcal{V} \to \mathcal{V} \cup \mathcal{P}$ be a partial function. The* substitution *σ on a multiset $m \in \mathcal{M}_{\mathcal{V}\cup\mathcal{P}, \mathcal{V}\cup\mathcal{P}}$ is given by $(m\sigma)(s)(c) = \sum_{r\sigma = s \wedge d\sigma = c} m(s)(d)$.*

The operational semantics for reconfigurable nets can be defined by replacing the rule (FIRING) in Figure 4 by the following (RECONF-FIRING) rule:

$$\text{(RECONF-FIRING)}\quad \frac{t = m\ [\rangle\ m' \in T \quad m'' \in \mathcal{M}_{S,S}}{m \star \sigma \oplus m'' \to_T m'\sigma \oplus m''} \quad \begin{array}{l} dom(\sigma) = rn(t), \text{ and} \\ \sigma(v) \in S \text{ for } v \in dom(\sigma) \end{array}$$

Comparing rule (COLOURED-FIRING) of C-P/T nets and (RECONF-FIRING) of R-P/T nets, it should be clear that in both cases a transition t can be fired on m only when m contains an instance of the preset obtained by renaming only colours (i.e., $m \star \sigma$). That is, a transition in both C-P/T nets and R-P/T nets consumes messages from a fixed set of places. Differently, in C-P/T nets, tokens generated by firing t corresponds to an instance of the postset of t obtained by substituting only colours accordingly to σ, whereas in R-P/T nets the renaming also affects names appearing in place position. For this reason, in C-P/T nets a transition produces messages in a fixed set of places (although their colour can be different for each firing). Instead, in R-P/T nets two different firings of the same transition can produce messages in different places, i.e. the postset of a transition changes dynamically depending on the colours of the consumed messages.

5.2 Reconfigurable ZS nets

The first consideration is that in R-P/T net there is no difference between places and colours. Taking into account that places in ZS nets can be either stable or zero, also colours are zero and stable. Consequently, the distinction between stable and zero markings must also consider colours present inside places. Actually, a stable marking should contain only stable names, therefore we write s to indicate $s \in \mathcal{M}_{L,L}$ and we write z for denoting a zero marking. At a first glance, it could appear that any other marking denotes a zero marking. This is not the case for a non-empty marking $m \in \mathcal{M}_{L,Z}$. Markings of this kind contain stable places with tokens coloured with zero names, which is somehow contrary to the ZS approach. Note that in ZS nets, tokens in stable places produced during a transaction are released only at commit, when all zero tokens have been consumed. Consequently, we will restrict zero markings to $z \in \mathcal{M}_{Z,S}$. We denote the set of well-defined markings as $\mathcal{W}_{L,Z} = \mathcal{M}_{L,L} \cup \mathcal{M}_{Z,L\cup Z}$. Additionally, we consider the set of variable names $\mathcal{V}$ as partitioned into sets: $\mathcal{V}_L$, the set of stable variables $V, W, \ldots$, and $\mathcal{V}_Z$ the set of zero variables $v, w, \ldots$.

Definition 5.3 (r-zs net). *A* Reconfigurable ZS net *is a 6-tuple $B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_0B, Z_B)$ where $N_B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_0B)$ is the underlying* R-P/T *net and the set $Z_B \subseteq S_B$ is the set of* zero places. *The places in $S_B \backslash Z_B$ (denoted by L_B) are called stable places. A stable marking m is a coloured multiset of* stable places *(i.e., $m \in \mathcal{M}_{L_B,L_B}$), and the initial marking m_0B must be stable. Moreover, we impose the pre and postset functions to be defined over well-defined markings, i.e., $\forall t \in T_B, \delta_{iN}(t) \in \mathcal{W}_{L_B\cup\mathcal{V}_L, Z_N\cup\mathcal{V}_Z}$, for $i = 1, 2$.*

We require transitions to be fired with appropriate names, that is zero variables are substituted by zero places and stable variables by stable places.

Definition 5.4 (Type preserving substitution). *A substitution σ is* type preserving *if $\forall V \in \mathcal{V}_L, \sigma(V) \in (L_B \cup \mathcal{V}_L)$ and $\forall v \in \mathcal{V}_Z, \sigma(v) \in (Z_B \cup \mathcal{V}_Z)$.*

In what follows we assume all substitutions to be type preserving. Then, firing rule for R-ZS nets can be written as follows.

(RECONF-FIRING)
$$\frac{t = s \oplus z \; [\rangle \; s' \oplus z' \in T \quad s'' \in \mathcal{M}_{L,L} \quad z'' \in \mathcal{M}_{Z,S}}{(s \star \sigma \oplus s'', z \star \sigma \oplus z'') \rightarrow_T (s'\sigma \oplus s'', z'\sigma \oplus z'')} \quad \begin{array}{l} dom(\sigma) = rn(t), \text{ and} \\ \sigma(v) \in S \text{ for } v \in dom(\sigma) \end{array}$$

Example 5.1 (Mailing list). Consider a data structure that allows to send atomically a message to a list of subscribers (in the sense that it is either sent to all or to none). Figure 14 shows a ZS net corresponding to such structure. **Nil** is a stable constant colour, all other colours used for labelling arcs are stable variables.

The stable place `newSubs` contains the tokens corresponding to the agents that want to be subscribed to the list. Their colours are the places in which they expect to receive a new message. Place `top` contains the element on top of the list (the latest subscriber). We assume that an empty list is denoted with a token coloured with the constant colour **Nil**. A list is encoded with several tokens in place `subscList`, where each token carries on the information corresponding to one subscriber and the next subscriber in the list, hence their colours are pairs.

By firing the transition `add` a new subscriber N is added on top of the list. The token corresponding to the previous subscriber on top of the list (whose colour is T) is replaced with a new token of colour N, i.e., the new subscriber becomes the top of the list. Also, a new token is produced in `subscList` whose colour is (N, T), meaning that the subscriber that follows N in the list is the previous element on top of the list T.

Transition `tell` allows to send a message M to every subscriber in the list. When `tell` is fired a new transaction is initiated, because a new token is generated in place `sending`, which is a zero place. Note that the top of the list is consumed, and a new token with the same colour is produced in `top`, but it will be released only when the transaction finishes. Therefore, transitions `add` and `tell` will not be enabled until the current transaction finishes.

The zero token present in `sending` contains the information of the subscriber to notify (i.e., the first colour of the pair), and the message to send (i.e., the second colour). Transition `notify` is a reconfigurable transition. In fact, it consumes from `sending` the token (T, M), and sends M to the subscriber T (nevertheless this token will be available actually when the transaction finishes). Additionally, `notify` takes from `subscList` the subscriber F that follows T in the list, and update the state of the transaction by putting in the zero place `sending` a token to notify the next subscriber with M.

The transaction finishes when the end of the list is reached. That is, when the token in `sending` is addressed to the receiver **Nil**. At this point, the transition `end` can be fired to consume the zero token present in the net, which will release all the stable tokens produced during the transaction. At this moment all subscribers atomically receive message M, and the top of the list is available for executing new activities.

5.3 Abstract net under the ITph approach

The definition of the abstract semantics of R-ZS net also relies on the identification of the basic atomic computations of the net, and for the ITph approach on the notion of Goltz-Reisig processes. The interesting point here is that during a computation on a reconfigurable net some transitions are instantiated in a particular way. Consider the reconfigurable net shown in 15(a), consisting of two transitions $t_1 = a(u, v)[\rangle u(v)$ and $t_2 = c(w)[\rangle w(\bullet)$, where a, b and c are places, and u, v and w variables. Figure 15(b), shows a possible execution in the net where the transition $\mathtt{t}'_1$ is an particular case of t_1, where the received name u has been used as colour c. Consequently, our notion of processes of a reconfigurable net is based on this idea of instantiation.

Definition 5.5 (Instance of a transition). *Let $t = m[\rangle m'$ be a transition. A transition i is an* instance *of t for a substitution σ if $dom(\sigma) \subseteq rn(t)$ and $i = m \star \sigma[\rangle m'\sigma$.*

Definition 5.6 (Reconfigurable net morphism). *Let N, N' be* R-P/T *nets. A tuple $f = (f_S : S_N \to S_{N'}, f_T : T_N \to T_{N'}, \rho = \{\rho_t\}_{t \in T_N}, \sigma = \{\sigma_t\}_{t \in T_N})$ is a* reconfigurable net morphism *from N to N' (written $f : N \to_{\sigma,\rho} N'$) if $\forall t \in T_N$:*

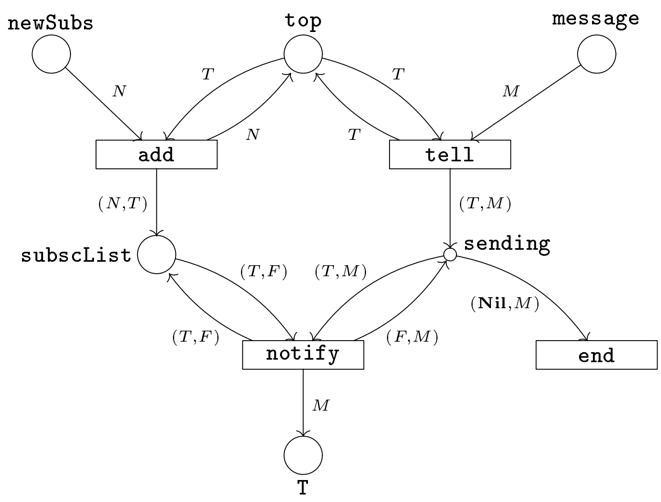


Fig. 14. A R-ZS net for the mailing list example.

– $\rho_t : \mathcal{V} \to \mathcal{P}$ *(i.e., substitutes variables by constants);*
– $f_S({}^\bullet t \star \rho_t)[\rangle f_S(t^\bullet \star \rho_t)$ *is an instance of* ${}^\bullet f_T(t)[\rangle f_T(t)^\bullet$ *for* σ_t*.*

Substitution ρ_t *are referred to as* proper substitutions *or* instantiations*.*

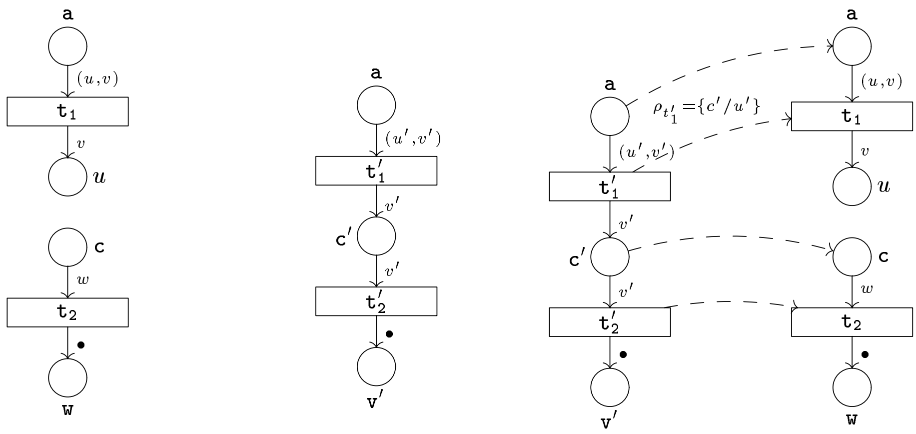
For the particular case of coloured transitions (i.e., transitions without reconfigurable capabilities), the condition required on the mapping is analogous to that on coloured net morphisms (Definition 4.9). In fact, no proper instantiations are needed. Moreover, if used they correspond to instantiations of colours.

Figure 15(c) shows a morphism between reconfigurable nets. Note that the received name u' of $\mathtt{t}_1'$ has been instantiated as c' (i.e., the proper substitution is $\rho_{t_1} = \{c'/u'\}$), because t_2 consumes messages from c. Nevertheless, the whole net is still a reconfigurable net. In fact, the place v' in which the final transition will produce the token depends on the colour of the token consumed from a'.

Definition 5.7 (Process of a reconfigurable net). *A* (Goltz-Reisig) process *for a* R-P/T *net* N *is a reconfigurable net morphism* $P : K \to_{\sigma,\rho} N$*, from a reconfigurable causal net* K *to* N*, s.t. every* ρ_{t_k} *is minimal (i.e., for every other* ρ'_{t_k} *that satisfies the morphism conditions* $\rho'_{t_k} \subseteq \rho_{t_k}$ *holds) and every* σ_{t_k} *is injective and* $\sigma_{t_k} : \mathcal{V} \to \mathcal{V}$*.*

As done for coloured nets, we also define a notion of compatible execution of a process to capture the relation between the different colours appearing in the causal net.

Definition 5.8 (Compatible execution of $[\![P]\!]_\approx$). *Let* $\varsigma \in [\![P]\!]_\approx$ *s.t.* $\forall t_1, t_2 \in T_\varsigma, rn(t_1) \cap rn(t_2) = \emptyset$*, i.e., transitions do not share variables. A substitution* σ *is said a* compatible execution *of* ς *if:*



(a) A R-P/T net (b) An execution. (c) A morphism.

Fig. 15. Reconfigurable net morphism.

– *if* $\forall a \in S_\varsigma, {}^\bullet a \star \sigma = a^\bullet \star \sigma$.
– σ *is consistent with any proper instantiation* ρ_t *in* ς, *i.e.,* $\forall t \in T_\varsigma$, $\rho_t \subseteq \sigma$.

If such σ *exists, we say that* $[\![P]\!]_\approx$ *is compatible. A process* P *is* compatible *if there exists a compatible execution for* $[\![P]\!]_\approx$.

The first condition is similar to that for compatible executions of coloured processes. The second one assures that in a compatible execution a name is not instantiated in different ways.

Example 5.2. Consider a reconfigurable net consisting of the following transitions: $t_1 = a(v)[\rangle b(v) \oplus v(\bullet)$, $t_2 = b(w)[\rangle w(\bullet)$, $t_3 = d(\bullet)[\rangle f(\bullet)$ and $t_4 = e(\bullet)[\rangle g(\bullet)$. Figure 16 shows two process of the net (both can be taken as representative of their equivalence class because their transitions do not share variable names). For simplicity, we call transitions in the causal net t' if they are mapped to t, while corresponding places have the same name. Places with name such as $\mathtt{w} = \mathtt{e}$ denote the proper instantiations used by the morphism. Consider the first process, the place $\mathtt{v} = \mathtt{d}$ can be mapped only into d, because $\mathtt{t}'_3$ in the net corresponds to t_3 in the original net. Consequently, $\mathtt{t}'_1$ is an instance of t_1 for the proper instantiation $\{d/v\}$.

Note that the first process does not admit a compatible execution σ. By the first condition of a compatible execution, σ should include substitutions $\{v/w\}$, $\{w/v\}$ or $\{u/v, u/w\}$. By the second, as $\mathtt{t}'_1$ is a proper instantiation for $\{d/v\}$ then $\{d/v\}$ should be in σ. Similarly, by considering $\mathtt{t}'_2$, $\{e/w\}$ should be in σ. Hence, all conditions together are inconsistent because a substitution is a function and cannot assign two different substitutions for the same variable.

The second process admits $\sigma = \{d/v, d/w\}$ as a compatible execution.

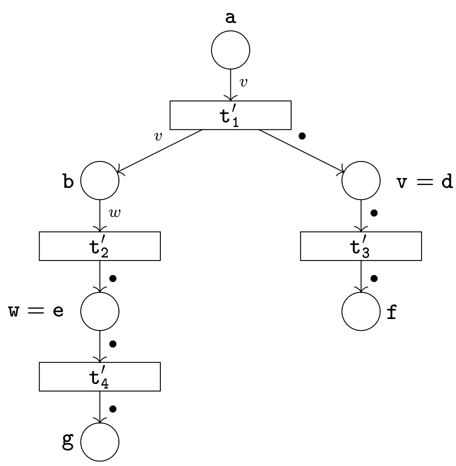


Fig. 16. Two reconfigurable processes.

The definitions for the most general compatible execution (mgce), compatible process, connected transactions, and causal abstract net are analogous to those presented in Section 4.3.

Example 5.3 (Abstract net for the mailing list example). In Figure 17 we (partially) show the abstract net corresponding to the R-ZS net in Figure 14. Transition `add` is identical to transition `add` in Figure 14. The transition `tell n` sends atomically a message M to n subscribers in the list whose top is N_1 and finishes in **Nil**. There is one such transition for any $n \geq 2$. The transition `drop` handles the case in which the list is empty. In such situations the message sent is simply lost (consumed).

Also for the reconfigurable case, the following theorem assures the correspondence between the abstract and the concrete view.

Theorem 5.1. *Let B be a* R-ZS *net and I_B its abstract net. Then $m \rightarrow_{T_{I_B}} m'$ iff $m \Rightarrow_{T_B} m'$.*

Proof. The proof follows as in Theorem 4.1 (considering also proper instantiations).

C-ZS nets as R-ZS nets. C-ZS nets are a particular case of R-ZS net, where no transition uses received names as places in their postset. Thus, given a C-ZS net B, it is possible to construct its abstract C-P/T net C_B and its abstract R-P/T net R_B. The following results assure that both constructions are isomorphic.

Proposition 5.1. *Let N be a coloured net. If P is a compatible coloured process, then P is a compatible reconfigurable process.*

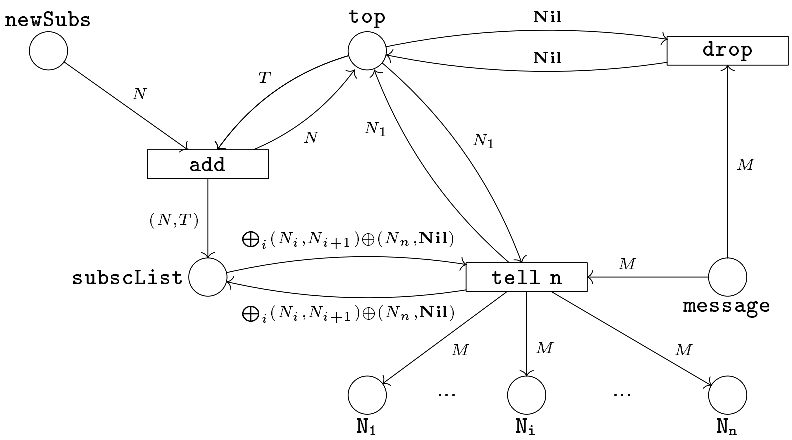


Fig. 17. A partial view of the abstract net for the mailing list example.

Theorem 5.2. *Let B be a* C-ZS *net, C_B its abstract* C-P/T *net, and R_B its abstract* R-P/T *net R_B. Then $R_B \approx C_B$.*

Proof (sketch). The proof follows from Proposition 5.1. First noting that equivalence classes are the same under both views. Finally, as proper instantiations are not used for coloured transition, a compatible execution under the coloured view is also compatible under the reconfigurable view.

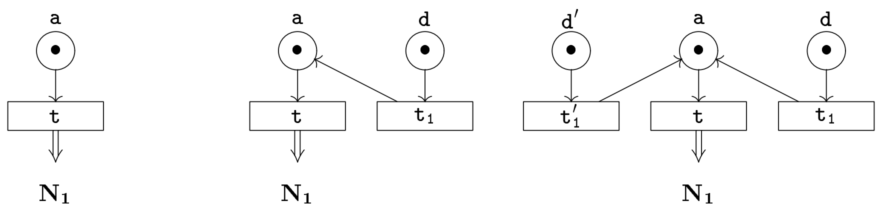
6 Towards Dynamic zs nets

6.1 Dynamic nets

While in reconfigurable nets the sets of states and transitions remains unchanged during computations, *dynamic nets* can create new components while executing: new places and transitions may be added to the net when a transition is fired. The main idea is that the firing of a transition may allocate a new subnet, which is parametric on the actual values of the received names. Nevertheless, it is not possible to modify existing transitions: they always consume tokens from a fixed multiset of places and the postset is always the same expression (multiset of places or nets) parametric on the received values. Moreover, it is not possible to attach new transitions with preset in a place after the net has been instantiated (i.e., there is not input capability). The definition given here of dynamic nets follows the presentation given in [2].

Definition 6.1 (DN). *The set* DN *is the least set satisfying the following equation:*

$$\begin{aligned}\mathcal{N} = \{(S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N}) \mid \\ S_N \subseteq \mathcal{P} \;\wedge\; \delta_{0N} : T_N \to \mathcal{M}_{S_N,\mathcal{C}} \;\wedge\; \delta_{1N} : T_N \to \mathcal{N} \;\wedge\; m_{0N} \in \mathcal{M}_{\mathcal{P},\mathcal{C}}\}\end{aligned}$$



(a) Initial state. (b) After firing t. (c) After two firings of t.

Fig. 18. A simple dynamic net.

If $(S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N}) \in$ DN: S_N is the set of places, T_N is the set of the transitions, δ_{0N} and δ_{1N} are the functions assigning the pre and postset to every transition, and m_{0N} is the initial marking. Note that while in previous nets the initial marking is required to be a multiset over the places of the net, here we allow a net to fixed a marking over states that are not defined by it. In fact, the initial marking m_{0N} is a multiset over $\mathcal{P}$ (i.e., $m_{0N} \in \mathcal{M}_{\mathcal{P},\mathcal{C}}$) and not over the places of the net S_N ($\mathcal{M}_{S_N,\mathcal{C}}$). A trivial example is the way in which a coloured transition $a(v)[\rangle b(\bullet)$ is written: $a(v)[\rangle(\emptyset, \emptyset, \emptyset, \emptyset, b(\bullet))$, where b clearly does not belong to the new subnet. In what follows, we write coloured and reconfigurable transitions as in the previous sections, and use verbose notation just for transitions that allocate new components. Also the postset of transitions defined in a new subnet can produce tokens in places not defined by it. Nevertheless, well-defined subnets cannot use places that do not belong to the net they are in.

Names defined in S_N act as binders on N. Therefore, nets are considered up-to α-conversion on S_N. Specially, the creation in N of a new subnet N_1 means the creation of a α-equivalent net N_1' s.t. all names in $S_{N_1'}$ are guaranteed to be different from any other place in N (i.e., they are fresh).

Example 6.1 (A simple dynamic net). Consider the net N represented in Figure 18(a). The double-lined arrow indicates the dynamic transition $t = a(\bullet)[\rangle N_1$, which creates an instance of the subnet N_1 when fired. We allow the initial marking of N_1 and the postset of transitions in T_{N_1} to generate tokens in a. Therefore, the following is a valid definition for N_1: $S_{N_1} = \{d\}, T_{N_1} = \{t_1\}, m_1 = a(\bullet) \oplus d(\bullet)$ and $t = d(\bullet)[\rangle a(\bullet)$. A firing of t will produced the net shown in 18(b). A new place d and a transition t (whose pre and postset are $d(\bullet)$ and $a(\bullet)$, resp.) have been added to the net. Also two tokens have been produced: one in a and the other in d, accordingly to the initial marking of N_1. In this marking t is enabled and can be fired again. The intended meaning of the new activation of t is to create a new subnet: a new place and a new transition whose names are different from others in the net (Figure 18(c)).

Definition 6.2 (Defined and Free names). *The set of* defined names in a marking m *is* $dn(m) = \{a | a \in m\}$, *i.e. names appearing in place position. Given*

$N = (S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N}) \in$ DN, *the* set of defined *(dn) and* free *(fn)* names *of transitions, sets of transitions, and nets are defined as follow:*

$dn(m_1[\rangle N_1) = dn(m_1)$
$dn(N) = dn(T_N) = \bigcup_{t \in T_N} dn(t)$
$fn(m_1[\rangle N_1) = dn(m_1) \cup col_{\mathcal{B}}(m_1) \cup (fn(N_1) \setminus rn(m_1))$
$fn(T_N) = \bigcup_{t \in T_N} fn(t) \setminus dn(T_N)$
$fn(N) = fn(T_N) \setminus S_N$

Definition 6.3 (Dynamic Net). *$N \in DN$ is a* dynamic net *if $fn(N) = \emptyset$.*

As mentioned above, a well-defined net N does not generate tokens in places that do not belong to it. Note that the condition on the free names imposed for dynamic nets (i.e., $fn(N) = \emptyset$) assures that tokens are always generated in the same net. In this case, any name is bound to a particular place defined in the net, which is guaranteed to be different to any other place.

As for coloured and reconfigurable nets, the firing of a transition t requires the postset to be instantiated with the received colours of t, i.e., the parameters of the t ($rn(t)$). Hence, we need a suitable notion of substitution on nets.

Definition 6.4 (Instantiation of a net). *Let $\sigma : \mathcal{V} \to \mathcal{P} \cup \mathcal{V}$ be a substitution. The* instantiation of a transition *$t = m_1[\rangle N_1$ with σ s.t. $rn(t) \cap dom(\sigma) = \emptyset$ is defined as $t\sigma = m_1[\rangle N_1\sigma$. Given a dynamic net $N = (S_N, T_N, \delta_{0N}, \delta_{1N}, m_{0N})$, the* instantiation of *N with σ s.t. $dom(\sigma) \cap S_N = \emptyset$ is defined as $N\sigma = (S_N, T_N, \delta_{0N}, \delta_{1N}\sigma, m_{0N}\sigma)$, where $\delta_{1N}\sigma(t) = (\delta(t)_{1N})\sigma$.*

The condition imposed on the substitution used to instantiate a net (or a transition) avoids the capture of free names appearing in the substitution. If this side condition is not satisfied, an α-conversion on the places of the net (or on the received names of the transition) can be applied before.

Definition 6.5 (Composition of nets). *Let N_1 and N_2 be dynamic nets. The* addition *of N_2 to N_1 (written $N_1 \otimes N_2$) defined as $N_1 \otimes N_2 = (S_{N_1} \cup S_{N_2}, T_{N_1} \cup T_{N_2}, \delta_{0N_1} \cup \delta_{0N_2}, \delta_{1N_1} \cup \delta_{1N_2}, m_{0N_1} \oplus m_{0N_2})$ provided with the fact that $N_1 \cap N_2 = \emptyset$ and $fn(N_1) \cap S_{N_2} = \emptyset$. The addition $N_1 \otimes N_2$ is said the* parallel composition *of N_1 and N_2 (written $N_1 \oplus N_2$) if also $fn(N_2) \cap S_{N_1} = \emptyset$.*

Observe that side conditions for the parallel composition avoid free names in one net to be captured by the transitions defined by of the other. Nevertheless, when a subnet N_2 is added to a net N_1 ($N_1 \otimes N_2$) we allow the free names of N_2 to be capture by the definitions in N_1. We remind that we are considering nets up-to α-conversion in the name of the places, thus it is always possible to rename places in order to satisfy the side conditions mentioned above.

In order to provide the operational semantics for dynamic nets, we remark that the state of a net is not given just in terms of the markings, but also in the structure of the net. The operational semantics is presented in Figure 19. For simplicity we write (S, T, m) as a shorthand for $(S, T, \delta_0, \delta_1, m)$. Rule DYN-FIRING stands for the firing of t when the marking contains an instance of the

$$\text{(DYN-FIRING)}\quad \frac{t = m \,[\rangle\, N_1 \in T \qquad m'' \in \mathcal{M}_{S_N,\mathcal{C}}}{(S,T,m \star \sigma \oplus m'') \to (S,T,m'') \otimes N_1\sigma} \quad \begin{array}{l} dom(\sigma) = rn(t), \text{ and} \\ \sigma(v) \in S \text{ for } v \in dom(\sigma) \end{array}$$

$$\text{(DYN-STEP)}\quad \frac{(S,T,m_1) \to (S,T,m_1') \otimes N_1 \quad (S,T,m_2) \to (S,T,m_2') \otimes N_2}{(S,T,m_1 \oplus m_2) \to (S,T,m_1' \oplus m_2') \otimes (N_1 \oplus N_2)}$$

Fig. 19. Operational semantics of dynamic nets.

preset of t (for a suitable substitution on colours σ). The resulting net consists of the original net, where the consumed tokens have been removed, and a new instance of N_1 (i.e., the postset of t). Note that the composition $\otimes$ of nets assures that the names of the added components are fresh. Rule DYN-STEP stands for the parallel composition of computations when the initial marking contains enough tokens to execute them independently. By requiring $(N_1 \oplus N_2)$, the components added by concurrent activities are guaranteed to be disjoint.

It is worth noting that reconfigurable nets are a particular case of dynamic nets. In fact when t is a reconfigurable rule, i.e. $N_1 = (\emptyset, \emptyset, m_1)$, the expression $(S,T,m) \otimes N_1 = (S,T,m \oplus m_1)$ corresponds to the RECONF-FIRING rule.

6.2 Applying the ZS approach to Dynamic nets

The evolving structure of dynamic nets opens several possibilities when applying the ZS approach. The more obvious option is to provide transactions by allowing any net to define stable and zero places, as done for the other kind of nets. Nevertheless, other options can take advantage of the possibility of creating subnets to specify subactivities that should be executed atomically, providing in this way a hierarchy of atomic activities, i.e., nested transactions. On the rest of this section we describe the operational semantics for the first case, called *flat dynamic* ZS nets. We left as interesting problems to be investigated in the future the characterization of the abstract net, and the different possibilities for applying the ZS approach to dynamic nets.

Flat dynamic zs nets. As mentioned above, flat dynamic ZS nets correspond to a direct application of the ZS approach where the places of a net B are either stable, i.e. in $L_B = S_B \backslash Z_B$, or zero, i.e., in Z_B. As for reconfigurable nets, we rely on two disjoint set of variables: $\mathcal{V}_{L_B}$, ranged over by $V, W, \ldots$ for stable variables, and $\mathcal{V}_{Z_B}$ for zero variables $v, w, \ldots$. Similarly, we use $s \in \mathcal{M}_{L,L}$ for denoting a stable marking, and $z \in \mathcal{M}_{Z,S}$ for zero markings. Moreover $\mathcal{W}_{L,Z} = \mathcal{M}_{L,L} \cup \mathcal{M}_{Z,L\cup Z}$ stands for the set of well-defined markings.

(DYN-FIRING)
$$\frac{t = s \oplus z[\rangle N_1 \in T \qquad s'' \in \mathcal{M}_{L,L} \qquad z'' \in \mathcal{M}_{Z,S}}{(S,T,(s\star\sigma \oplus s'', z\star\sigma \oplus z''),Z) \to (S,T,(s'',z''),Z) \circledcirc N_1\sigma} \quad \begin{array}{l} dom(\sigma) = rn(t), \text{ and} \\ \sigma(v) \in S \text{ for } v \in dom(\sigma) \end{array}$$

(DYN-STEP)
$$\frac{(S,T,m_1,Z) \to (S,T,m_1',Z) \circledcirc N_1 \quad (S,T,m_2,Z) \to (S,T,m_2',Z) \circledcirc N_2}{(S,T,m_1 \oplus m_2,Z) \to (S,T,m_1' \oplus m_2',Z) \circledcirc (N_1 \oplus N_2)}$$

(DYN-CONCATENATION)
$$\frac{(S,T,s_1 \oplus z_1,Z) \to (S'',T'',z'',Z'') \circledcirc s_1' \quad (S'',T'',s_2 \oplus z'',Z'') \to (S',T',s_2' \oplus z',Z')}{(S,T,(s_1 \oplus s_2,z),Z) \to (S',T',(s_1' \oplus s_2',z'),Z)}$$

(DYN-CLOSE)
$$\frac{(S,T,s_1,Z) \to (S',T',s_1',Z')}{(S,T,s_1,Z) \Rightarrow (S',T',s_1',Z')}$$

Fig. 20. Operational semantics of flat dynamic SZ nets.

Definition 6.6 (Flat d-zs net). *A* flat dynamic ZS net *is a 6-tuple* $B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B}, Z_B)$ *where* $N_B = (S_B, T_B, \delta_{0B}, \delta_{1B}, m_{0B})$ *is the underlying dynamic net and the set* $Z_B \subseteq S_B$ *is the set of* zero places. *The places in* $S_B \backslash Z_B$ *(denoted by* L_B*) are called* stable places. *A stable marking* m *is a coloured multiset of stable places (i.e.,* $m \in \mathcal{M}_{L_B,L_B}$*), and the initial marking* m_{0B} *must be stable. Moreover, we impose the pre and postset functions to be defined over well-defined markings.*

Rules in Figure 20 shows the operational semantics of flat D-ZS nets. The rules are the straightforward extension of rules corresponding to R-ZS nets for the case of dynamic transitions.

Example 6.2 (Private Mailing Lists). Consider the mailing list problem presented in Example 5.1. Suppose there are n users u_i, each of them needs to send atomically messages present in m_i to listeners whose names are in s_i. (Every user has its own list of subscribers and messages). The system can be modelled as a flat dynamic ZS net by reusing the mailing list structure in Figure 14. Consider the dynamic net in Figure 21 for the case of two users. The net N_1 (appearing in the postset of $t = \mathtt{new}(V,W)[\rangle N_1)$ corresponds exactly to the net in Figure 14 plus the initial marking $m_{0N_1} = V(\mathtt{newSubs}) \oplus W(\mathtt{message}) \oplus \mathtt{top}(\mathbf{Nil})$.

There are n transitions $\mathtt{subsc_i}$ and $\mathtt{dist_i}$, i.e., a pair for each user u_i. The listeners for the user u_i are in place s_i, while the messages are in m_i. For instance, the listeners for u_1 are j_1 and j_2, and it has only the message l_1 to send. To create a list for u_1 it is necessary to put a token on new with colour $(\mathtt{a_{s_1}}, \mathtt{d_{m_1}})$ (we omitted it in Figure 14 for space limitations) . The obtained net after firing new with colours $(\mathtt{a_{s_1}}, \mathtt{d_{m_1}})$ is shown in Figure 22. Note that a new instance of N_1 has been created. For convenience in the graphical representation we renamed newSubs with 1_s and message with 1_m. Observe that tokens corresponding to the initial marking of N_1 have been produced: token **Nil** in top, 1_s in $\mathtt{a_{s_1}}$, and 1_m in

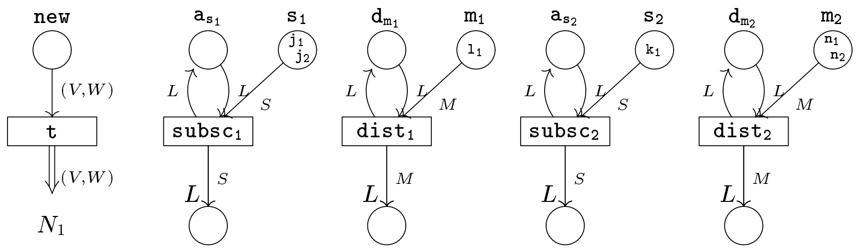


Fig. 21. Private Mailing Lists.

$\mathtt{d}_{\mathtt{m}_1}$. Now, listeners in $\mathtt{s}_1$ can be subscribed to the list by firing the reconfigurable transition $\mathtt{subsc}_1$. Note that any coloured token W in s_1 is forwarded to the place 1_s, which will enable the transition `add` of the mailing list structure. Similar is the case for tokens in $\mathtt{m}_1$, which are forwarded by $\mathtt{dist}_1$ to the place 1_m.

Suppose that t is fired again for a token $(\mathtt{a}_{\mathtt{s}_2}, \mathtt{d}_{\mathtt{m}_2})$ in `new`. In this case, a new mailing list structure is created, which is guaranteed to be independent of the first structure.

About the abstract view of flat dynamic ZS *nets.* The main difficulty when defining the abstract dynamic net describing the atomic movements of the concrete ZS nets is to figure out a suitable notion for a process. A process, viewed as morphism from a causal net into a P/T net, identifies elements of the causal net as particular instances of elements in P/T net. For P/T, C-P/T, and R-P/T nets, where the elements of the net are fixed, the correspondence between instances and general elements is quite clear. In particular, states are mapped into states and transitions (instance of some pattern) to transitions (representing a general pattern). Instead, when describing the execution of a dynamic net D it could be necessary to talk about states and transitions that are not present in D (although D describes how to create them). This question is still open and remains as an interesting problem that bears further investigation.

7 Conclusions

In this paper we have extended the zero-safe approach along the hierarchy of increasingly expressive models characterized in [15]. The results are summarized in Figure 1. Although the more general case of dynamic nets presents some technical difficulties in reconciling the operational and abstract view, the zero-safe approach has been shown somehow orthogonal to the whole hierarchy. Notably, when defining the operational semantics of C-ZS nets and R-ZS nets only the rule describing the firing of a transition is modified. Instead, for dynamic nets all rules are rewritten to consider also the structure of the net as part of the state. Regarding the abstract semantics, it is clear that the description of the abstract

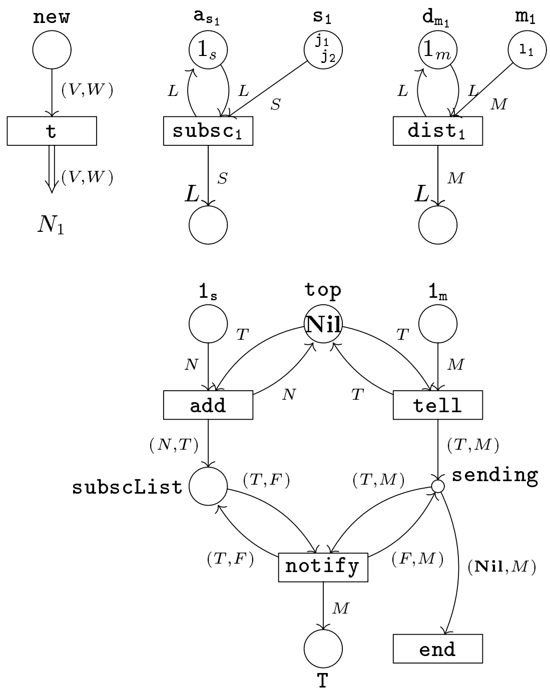


Fig. 22. Private Mailing Lists after firing t with colours $(\mathtt{a}_{\mathtt{s}_1}, \mathtt{d}_{\mathtt{m}_1})$

view associated to a dynamic ZS net — in particular the characterization of a process in such an evolving structure — remains as an open problem.

On the other hand, the extensions proposed here account only for flat transactions. We plan to investigate alternative extensions of dynamic nets for modelling nested transactions. In particular, by exploiting the capability of creating new subnets to describe sub-transactions. Moreover, the description of compensations in this framework is an ambitious goal that we leave to future work.

Finally, we think that the distributed two phase commit protocol proposed in [8] (used to encode ZS nets in Join) can be reused or extended to implement dynamic ZS nets.

References

1. M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inform. and Comput.*, 148(1):1–70, 1999.
2. A. Asperti and N. Busi. Mobile petri nets. Technical Report UBLCS96-10, University of Bologna, May 1996. 1996.
3. P. Baldan, H. Ehring, R. Heckel, K. Hoffmann and H. Ehrig. High-level net processes. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal*

and Natural Computing: Essays Dedicated to Grzegorz Rozenberg, Volume 2300 of *Lect. Notes in Comput. Sci.*, pages 191–219, Springer Verlag, 2002.
4. E. Best, R. Devillers, and J. Hall. The Petri Box Calculus: A new causal algebra with multi-label communication. 609:21–69, 1992.
5. L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In *Proc. of Sixth IFIP International Conference on Formal Methods for Open-Object Based Distributed Systems (FMOODS'03)*, Lect. Notes in Comput. Sci. Springer Verlag, 2003. To appear.
6. BPEL Specification. version 1.1. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`, May 2003.
7. R. Bruni, C. Laneve, and U. Montanari. Centralized and distributed orchestration of transactions in the jo in calculus. Technical Report TR-02-12, Computer Science Department, University of Pisa, 2002.
8. R. Bruni, C. Laneve, and U. Montanari. Orchestrating transactions in join calculus. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *Proceedings of CONCUR 2002, 13th International Conference on Concurrency Theory*, volume 2421 of *Lect. Notes in Comput. Sci.*, pages 321–336. Springer Verlag, 2002.
9. R. Bruni, H. Melgratti, and U. Montanari. Nested commits for mobile calculi: extending Join, 2003. Submitted.
10. R. Bruni and U. Montanari. Executing transactions in zero-safe nets. In M. Nielsen and D. Simpson, editors, *Proceedings of ICATPN 2000, 21st Int. Conf. on Application and Theory of Petri Nets*, volume 1825 of *Lect. Notes in Comput. Sci.*, pages 83–102. Springer Verlag, 2000.
11. R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Inform. and Comput.*, 156(1-2):46–89, 2000.
12. R. Bruni and U. Montanari. Transactions and zero-safe nets. In H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, editors, *Advances in Petri Nets: Unifying Petri Nets*, volume 2128 of *Lect. Notes in Comput. Sci.*, pages 380–426. Springer Verlag, 2001.
13. R. Bruni and U. Montanari. Zero-safe net models for transactions in Linda. In U. Montanari and V. Sassone, editors, *Proceedings of ConCoord 2001, International Workshop on Concurrency and Coordination*, volume 54 of *Elect. Notes in Th. Comput. Sci.*, 2001.
14. R. Bruni and U. Montanari. Concurrent models for linda with transactions. Math. Struct. in Comput. Sci., 2003. To appear.
15. M. Buscemi and V. Sassone. High-level Petri nets as type theories in the Join calculus. In F. Honsell and M. Miculan, editors, *Proceedings of FoSSaCS 2001, Foundations of Software Science and Computation Structures*, volume 2030 of *Lect. Notes in Comput. Sci.*, pages 104–120. Springer Verlag, 2001.
16. N. Busi. On zero safe nets. Private communication, April 1999.
17. N. Busi and G. Zavattaro. On the serializability of transactions in javaspaces. In U. Montanari and V. Sassone, editors, *Elect. Notes in Th. Comput. Sci.*, volume 54. Elsevier Science, 2001.
18. M. Butler, M. Chessell, C. Ferreira, C. Griffin, P. Henderson, and D. Vines. Extending the concept of transaction compensation. *IBM Systems Journal*, 41(4):743–758, 2002.
19. L. Cardelli and A.D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS'98, Foundations of Software Science and Computational Structures*, volume 1378 of *Lect. Notes in Comput. Sci.*, pages 140–155. Springer Verlag, 1998.
20. D Duggan. An architecture for secure fault-tolerant global applications. *TCS*. To appear.

21. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the Join calculus. In *Proceedings of POPL'96, 23rd Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages*, pages 372–385. ACM Press, 1996.
22. H. Garcia-Molina and K. Salem. Sagas. In U. Dayal and I.L. Traiger, editors, *Proceedings of the ACM Special Interest Group on Management of Data Annual Conference*, pages 249–259. ACM Press, 1987.
23. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
24. K. Jensen. *Coloured Petri Nets. Basic Concepts.* EATCS Monographs on Theoretical Computer Science. SV, 1992.
25. F. Leymann. WSFL Specification. version 1.0. `http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.
26. R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.
27. C.A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
28. W. Reisig. *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1985.
29. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets.* PhD thesis, Computer Science Department, University of Pisa, 1994.
30. U. Roxburgh. Biztalk orchestration: Transactions, exceptions, and debugging, 2001. Microsoft Corporation. Available at `http://msdn.microsoft.com/library/en-us/dnbiz/html/bizorchestr.asp`.
31. Sun Microsystem, Inc. JavaSpaces^{TM} service specifications, v.1.1, 2000.
32. S. Thatte. XLANG: Web Services for Business Process Design. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001.