Structuring and Composability Issues in Petri Nets Modeling

Luís Gomes, *Member, IEEE,* and João Paulo Barros

***Abstract*—Along Petri nets' history, numerous approaches have been proposed that try to manage model size through the introduction of structuring mechanisms allowing hierarchical representations and model composability. This paper proposes a classification system for Petri nets' structuring mechanisms and discusses each one of them. These include node fusion, node vectors, high-level nets, and object-oriented inspired Petri nets extensions, among others. One running example is used emphasizing the application of the presented mechanisms to specific areas, namely to automation systems modeling, and software engineering, where object-oriented modeling plays a major role.**

***Index Terms*—Hierarchies, model abstraction, model composition, model refinement, objects, Petri nets, structuring mechanisms.**

I. Introduction

IT IS A FACT easily confirmed by experience that "regular" Petri nets, also known as low-level Petri nets, are often difficult to use in practice due to the problem of rapid model growth. In fact, although Petri nets have been extensively studied since Carl Adam Petri work in 1962 [10], only the last 15 years have seen significant work on ways to compose Petri net models. The 1989 paper by Huber *et al.*, [32] stated that "In the literature there is almost no work on hierarchies in Petri nets". This situation has changed since then due to numerous additional proposals for hierarchical structuring and abstraction mechanisms in Petri nets (e.g., [3], [6], [8], [9], [12], [14], [23], [26], [31], [32], [42], [46], []). Just like in programming languages, those proposals are in fact abstraction constructs. However, anecdotal evidence seems to indicate that those hierarchical structuring mechanisms are not well-known outside the Petri nets community. In the authors' experience it is common to find people who are perfectly aware of Petri nets but still see them as extended state machines with no composition capabilities. This is even made more surprising because Petri nets have also entered the object-oriented arena for some time now, which implicitly brought additional and sophisticated abstraction constructs. The book [1] offers an up-to-date detailed survey of some of these approaches. This situation is probably due to the absence of an overview of the available composition mechanisms for Petri nets models. Although some Petri nets' tutorials have been available for some time (e.g., [16], [17], [25], [44], [47], [50], [52], [53], [59]), they give little and sparse attention to structuring mechanisms for Petri net models. Although not relevant from a theoretical point of view, in the sense that they do not increase the modeling capacity, they are extremely relevant from a practical or from an engineering point of view. Even the simplest hierarchical decomposition of a large Petri net model can mean the difference between a manageable and an unmanageable model.

The addition of structuring mechanisms to Petri nets has originated or contributed to the large diversity of Petri nets classes (or types). The unification of those numerous dialects is already an important research topic [20] and some authors even wonder what a Petri net is [36]. Here we do not intend to review or classify all the different Petri net classes: they are simply far too many. Instead, we focus on the structuring mechanisms for Petri net models; when they are part of well-known Petri nets' classes, we mention them too. To that end, we present a novel classification for those structuring mechanisms.

At the first level, we classify Petri net structuring mechanisms as usually found on the Petri nets' theoretical literature: **composition** versus **refinement/abstraction**.

Composition means the interconnection of several Petri net models. It is based upon the concepts of place and transition fusion. It corresponds to the programming languages concepts of block, module, or class composition. We will present the folding in high-level Petri nets as a particular case of net composition. Interestingly, high-level nets usually also include some kind of modular composition abstraction. Refinement/abstraction mechanisms correspond to the programming languages concepts of macros or procedures. We will see these can be supported by node fusions, the typical net composition construct.

The following section presents the classification for Petri nets structuring mechanisms. Along the text, the referred structuring techniques are presented in the framework provided by the proposed classification. The review emphasizes the most useful constructs, from an engineering point of view, for several selected areas, namely automation systems, and software engineering.

II. Classification for Petri Nets Structuring Mechanisms

Fig. 1 presents a novel view for the classification on Petri nets structuring mechanisms. It starts by the usual dual classification: composition versus refinement/abstraction.

Besides the well-known place and transition fusions, high-level nets folding is seen as a especial kind of fusion, based on tokens. A less well-known kind of folding, based on nodes and node vectors, is also presented.

Manuscript received October 4, 2004; revised January 21, 2005.

L. Gomes is with the Universidade Nova de Lisboa/UNINOVA-Portugal, Lisboa, Portugal (e-mail: lugo@uninova.pt).

J. P. Barros is with the Universidade Nova de Lisboa/UNINOVA, Lisboa, Portugal, and also with the Instituto Politécnico de Beja, ESTIG, Beja, Portugal (e-mail: jpb@uninova.pt).

Digital Object Identifier 10.1109/TII.2005.844433

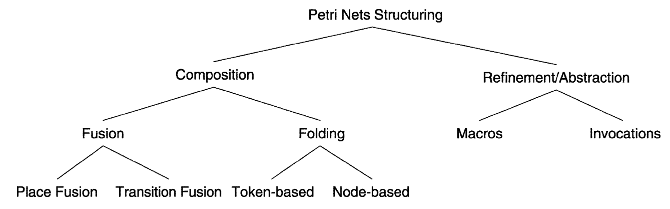

Fig. 1. Classification for Petri nets structuring mechanisms.

The refinement/abstraction structuring mechanism includes macro-based structuring and net modules dynamic creation. This includes the object-oriented Petri net classes.

In the following sections, the structuring mechanisms are presented and a running example is used to support illustration of different modeling situations. The example is centered on the modeling of a car parking lot controller. We start considering a car parking lot with one entrance and one exit. Internally, the parking lot area is divided into three areas. For the introductory example, each area has a maximum capacity of one car; latter, we will use a finite capacity associated with the parking lot sections. Fig. 2(a) presents the layout of the parking lot and Fig. 2(b) and (c) present the low-level net models associated with each area of the parking lot when occupied with one car or when a free place is available, respectively.

Afterwards, several variants will be considered, namely adding more entrances, adding more exits, and several refinements of some parts of the model using different techniques, from top-down decomposition strategies to object-orientation.

Is has to be stressed that this simple "neutral" running example can be kept close to generic modeling situations typically found in the related literature, like producer-consumer systems, FIFO systems, serial, and parallel buffers, just to mention a few. The referred modeling situations are common in several application areas, where concurrency formalisms play a major role, ranging from manufacturing systems, data communication networks, digital hardware design and, more recently, software engineering. We preferred to use a running example from the automation area, where the modeling capabilities can be presented, although keeping some generality.

A. Composition

Net composition is usually defined as the merging of nets into a single one through node fusion. This can be seen as a direct support for bottom-up approach to system design, or for the reusability of models already available. Here we generalize this idea: we see net folding as another form of net composition, where multiple and structurally identical nets are composed together in a single high-level net.

1) Fusion: In some earlier works, node fusion was used as a drawing convenience to fold a set of nodes into a single one, avoiding the use of long arcs connecting distant nodes, which could lead to legibility problems in the net model. Afterwards, node fusion took its place also as a fundamental concept to support model composability.

From the compositional point of view, node fusion offers a **horizontal composition**, as the nets are glued together at specific points (places or transitions) "side by side." A node fusion based model is like a puzzle where each piece can contain a full drawing but also has parts that connect it to others.

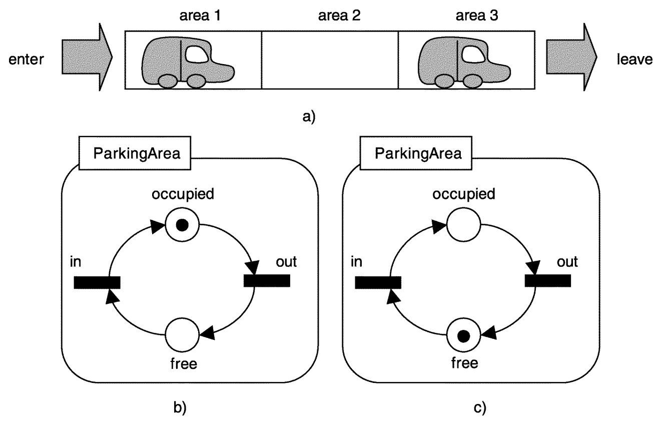

Fig. 2. (a) Queue in a car parking lot as a FIFO system, (b) net model for an occupied place, and (c) for a free place.

Unsurprisingly, node fusion can take the form of transition fusion and place fusion. The first has a clear connection to process algebras synchronous communication and this fact is deeply explored in the "Petri Net Algebra" [7]. The second allows asynchronous communication among processes. As pointed out in [19] and [22], synchronous communication among processes can not be naturally used to analyze state-based properties.

In spite of the complementary nature of transition fusion and place fusion, some proposals only use one or the other. For instance, hierarchical colored Petri nets [33] rely exclusively on place fusion. This is used not only to connect distinct Petri net models but also as a graphical convenience allowing the multiple appearance of any specific place in different locations across the graph model. This can bring a significant increase in model readability. It has to be noted that transition fusion was also referred in previous works on hierarchies for colored Petri nets [32], but was not included in the formal definition of hierarchical colored Petri nets [33] and associated tools (as Design-CPN [18] and CPN Tools [15]).

Probably due to the strong relation to process algebra compositions, several proposals rely exclusively on transition fusion (e.g., [5], [45]).

Some proposals use both fusions. Among them, it is important to mention the modular colored Petri nets [12], [13], [] and modular place/transition nets [14] that rely on place fusion and also on transition fusion to allow a modular state space analysis. The used transition fusion does not force the transition fusion sets to be disjoint. This differs from a previous proposal by Huber *et al.* [32], which can be seen as a first attempt to unify proposals for hierarchical structuring mechanisms for colored Petri nets.

The composition by node fusion is common in the proposals for state space analysis based on model decomposition. For example, Notomi and Murata [45] use transition fusion, while Juan *et al.* [19], [22], and Valmari [57] use place fusion. One proposal [11], referred as an example of place fusion in [19], uses additional places to bridge together two nets.

Coming back to our running example, we start considering a car parking lot as a FIFO system composed by three cascaded sections, as presented in Fig. 2(a). The whole system model can be obtained through replication of the models in Fig. 2(b) and (c), as needed, followed by composition through node fusion, as

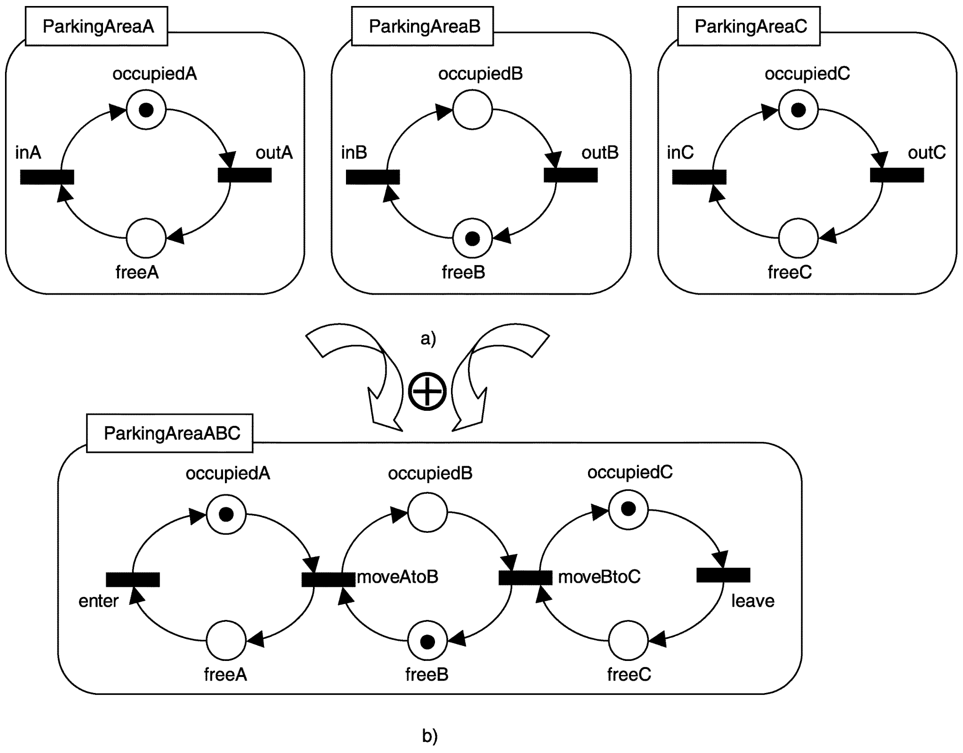

Fig. 3. Composition, through transition fusion, of the net models in (a) into the net model in (b).

a partially overlapping puzzle. In the current example, the components' composition is accomplished through transition fusion, starting with three submodels seen as components to be interconnected [see Fig. 3(a)]. All labels associated with places and transitions are renamed in order to be made unique. Then, transition fusion is used to compose the three models into the global model. As shown in Fig. 3(b), the transitions $outA$ and inB are fused into transition $moveAtoB$, transitions $outB$ and inC are fused into transition $moveBtoC$, transition inA is renamed $enter$, and transition $outC$ is renamed $leave$.

Net operations: From an engineering perspective, and besides the well-known top-down and bottom-up approaches, we can think of a third type of net composition where modifications can be spread across any number of the currently existent model parts. Yet, this is not necessarily seen as a bottom-up construction where one model module grows by connecting it to another, but as the modification of one or more existent model modules by a single new module. This new module imposes structural and behavioral modifications to one or more modules of the initial model. This corresponds to the realization of **crosscutting** requirements [2], [37], [55], at the net level. It has to be noted that, from the theoretical point of view, one main concern with model composability has been propriety preserving, in the sense that one can apply propriety analysis results obtained from the separated components to the resulting model. With this third attitude (crosscutting), the emphasis is shifted from "composability with propriety preserving" to "composability for propriety addition or propriety avoidance."

One simple operation named **net addition** [2], [3], formally defined as a net disjoint union followed by node fusions, supports all three net composition approaches (top-down, bottom-up, and crosscutting). The reverse operation is, appropriately, called **net subtraction**. These two simple operations can compose and decompose, in an orthogonal way, the existing nets or modules. They can operate on net instances [3], [26] and offer support for bottom-up construction and crosscutting modifications. As net composition also supports refinements (to be presented later), the two operations also offer a generalized support for top-down development [26].

A net addition operation is defined as a disjoint union followed by a set of node fusion among the nodes of the resulting net. The syntax is the following:

$$\begin{aligned} NewNet = {} & (OperandNetInstance_1 \oplus \ldots \\ & \oplus OperandNetInstance_p) \\ & (node_{1_1}/node_{1_2}/\ldots/node_{1_n} \mapsto newNode_1, \\ & \ldots, node_{k_1}/node_{k_2}/\ldots/node_{k_m} \mapsto newNode_k). \end{aligned} \tag{1}$$

Where the list of net instances to be added is presented first, followed by the identification of different node fusion sets to obtain the desired final composition.

The nodes to be fused belong to the operand net instances. The node fusions are specified by the set of nodes to be fused (separated by "/"); and the name of the resulting new nodes (after "$\mapsto$").[1] For example, the net in Fig. 3(b) can be defined, by net addition, based on the three nets in Fig. 3(a):

$$\begin{aligned} & ParkingAreaABC \\ & = (ParkingAreaA \oplus ParkingAreaB \oplus ParkingAreaC) \\ & \quad (inA \mapsto enter, outC \mapsto leave, outA/inB \mapsto moveAtoB, \\ & \quad outB/inC \mapsto moveBtoC). \end{aligned} \tag{2}$$

[1]It is clear that notations like "$\oplus$" and "$\mapsto$" should be replaced by ASCII representations, for adequate handling by computational tools.

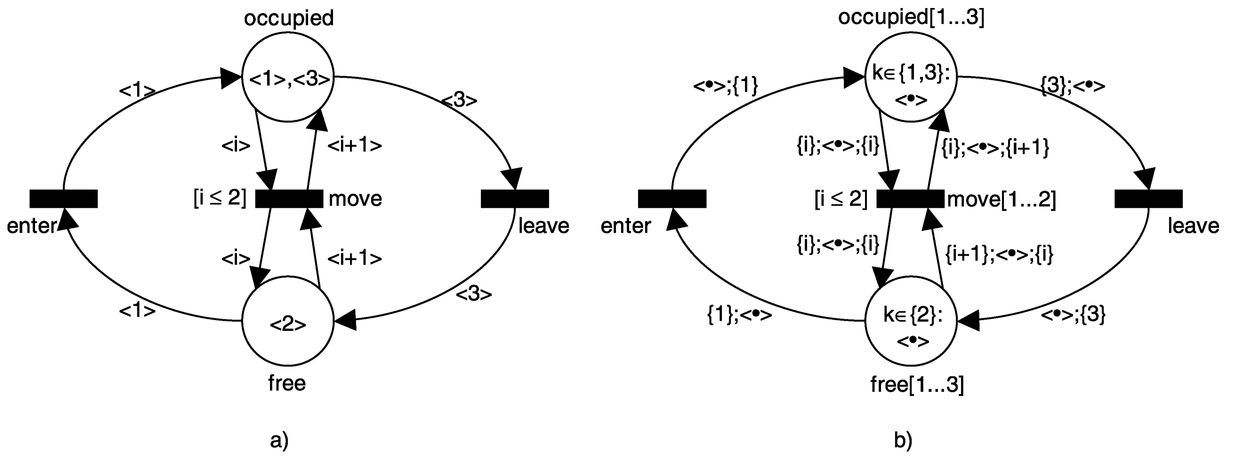

Fig. 4. Folding of net models (a) using a high-level Petri net model and (b) using a node vector notation.

The syntax reflects the three fundamental operations involved: 1) the disjoint union of the operand nets; 2) the node fusions among the resulting net nodes; and 3) the naming of resulting net nodes at the resulting net.

Several tools specify the structure definition of the whole model by annotations added to the net components. These annotations establish the connection among the several pages or modules. For example: node fusion is specified by annotating one node with the identifier of another node in the same or in a different subnet. Differently, when specifying net compositions with net addition and net subtraction we get a total separation between the composition information and the net components annotations. A new net can be defined by describing the way other nets are related: the net definition is based on operations that refer to the set of operand nets. This allows the quick specification of a large net model without any kind of graphical editing: we simply specify the textual expressions composing the nets; the original nets definitions remain exactly the same. As both operations are amenable to a simple textual representation to be made available by tools, this allows the representation, in a compact and readable format, of a large number of net compositions. These can be seen as contributing either to system development, or to future modification of a "completed" system.

2) Folding: Folding abstraction is another kind of composition. It is made possible by structural symmetries inside a low-level net model.

We consider two folding composition types:

- the first one leading to high-level nets is here named **token-based folding**: the folding is accomplished through the use of tokens as data structures.
- in the second one, here named **node-based folding**, the folding is accomplished by node fusion, but the folded nodes maintain their identity through the use of associated indexes. The nodes to be merged can be places, transitions, or other types of nodes, namely the **macronodes**, which will be presented in the following sections, within refinement/abstraction structuring mechanisms.

Token-based folding: Token-based folding is, by definition, the essence of high-level Petri nets [24], [33], [35]. High-level nets avoid repetitions in the net structure by increasing token complexity. As tokens can carry data, data transformation modeling is made possible, and usually necessary. As transitions and places are fused, the respective arcs can also be seen as "fused." These arc fusions are modeled by algebraic expressions associated to arcs, named *arc expressions*, which are able to model data transformations. In this sense, high-level Petri nets are also much more dependent on textual annotations than low-level nets. This is no surprise as it stems from the typical use of text to reduce graphic notation.

High-level nets folding can be seen as an "internal composition" made possible by structural symmetries inside a low-level net model. To illustrate this point we go back to our parking lot example in Fig. 3. We can easily identify symmetries and compose a folded colored net model, as presented in Fig. 4(a). There, the *occupied* places, for example, are composed (fused) into one. The same happens with the $free$ places and the $move$ transitions.

Considering the associated impact factor (measured in terms of the number of published works), we can identify three main classes of high-level nets: colored Petri nets [33], [34], predicate/transition nets [24], [30] and nets with individual tokens [49], [51].

Among these three, colored Petri nets (CPNs) are clearly the more popular class of high-level nets. This is testified by the worldwide use of two CPN-based tools: the CPN Tools [15] and the RENEW tool [58]. The CPN Tools extend the structuring capabilities of CPNs with macro transitions and place fusion. The RENEW tool extends CPNs in several ways, namely as a base for object nets (where tokens can be nets), several arc types, and a high-level version of transition fusion named synchronous channels.

Predicate/transition (PrT-nets) nets were the first class of high-level Petri nets which were defined with no especial application in mind. As mentioned in [33], compared to CPNs, PrT-nets can be seen as a slightly different dialect of the same language. Yet, probably due to the much stronger tool support for CPNs, PrT-nets have become much lesser known than CPNs. Notwithstanding, significant theoretical work has been conducted with PrT-nets (e.g., [24]), and also with hierarchical PrT-nets (e.g., [30]).

Nets with individual tokens can be seen as simplified CPNs as they do not allow functions in guards or in arc expressions.

They offer a nice theoretical model also useful for pedagogical purposes [51].

Node-based folding: Node-based folding is a complementary structuring mechanism that can also lead to a more compact model. It is based on the *node vector* concept introduced in [26], [27]. It can be used together with the net instance and net instance vector concepts. For this reason we refer to both node vectors and net instance vectors as *component vectors* [3].

Node-based folding relies on the association of a multiplicity factor (cardinality) to every node of the Petri net model. In this sense, every node can be seen as a vector of identical nodes. As an example to help clarifying the notation we consider Fig. 4(b). We use *occupied[1...3]* to represent the folding of three places represented in Fig. 3 as *occupiedA*, *occupiedB*, and *occupiedC*, which will be referred by *occupied[1]*, *occupied[2]*, and *occupied[3]*, respectively. A similar vector notation applies to other nodes in the example. In this sense, Fig. 4(a) net model using a colored Petri net model and Fig. 4(b) net model using node vectors are behaviorally equivalent.

Both foldings can be seen as graphical conveniences to allow model compactness and complexity management. In fact, it is always possible to obtain a low-level net model that is behaviorally equivalent to the folded net model. The set of procedures to allow unfolding of a net model using node-based folding was presented in [26], [27], and considers the following aspects.

- How to deal with the unfolding of arcs that are interconnecting nodes having a vector notation.
- How to deal with inscriptions associated with arcs connecting nodes with a vectorial attribute.
- How to deal with initial marking of place vectors.
- How to deal with guards associated with transition vectors.

As already referred, the vector notation can be used together with the net instance concept, leading to the net instance vector concept.

When we use more than one instance of a given net, we avoid the need to rename all the net components through the use of net instances [3], [26]. These are distinguished by appending an index between square brackets to the net identifier. In the present example, we use three instances of the same net, considering three empty parking lot sections. Accordingly, we also define a net instances vector. In our example, for three instances of $ParkingArea$ net, we get the composition in Fig. 5.

The composition in Fig. 5 can be expressed for any number $(nAreas)$ of $ParkingArea$ nets using a net addition of the several instances. This is expressed as a net instance vector. Assuming $nAreas = 3$, we write

$$\begin{aligned}&ParkingArea3\\&\quad= (\oplus ParkingArea[1\ldots nAreas])\\&\quad\quad(ParkingArea[1].in \mapsto enter,\\&\quad\quad\ ParkingArea[nAreas].out \mapsto leave,\\&\quad\quad\ (ParkingArea[i].out/ParkingArea[i+1].in\\&\quad\quad\quad \mapsto move[i]) \quad [i : 1\ldots nAreas-1]). \qquad (3)\end{aligned}$$

Note the use of the constant $nAreas$ denoting the number of added instances, and the fully qualified names used to denote the nodes: the name of the originating net followed by an instance number between square brackets, a dot, and the initial node identifier.

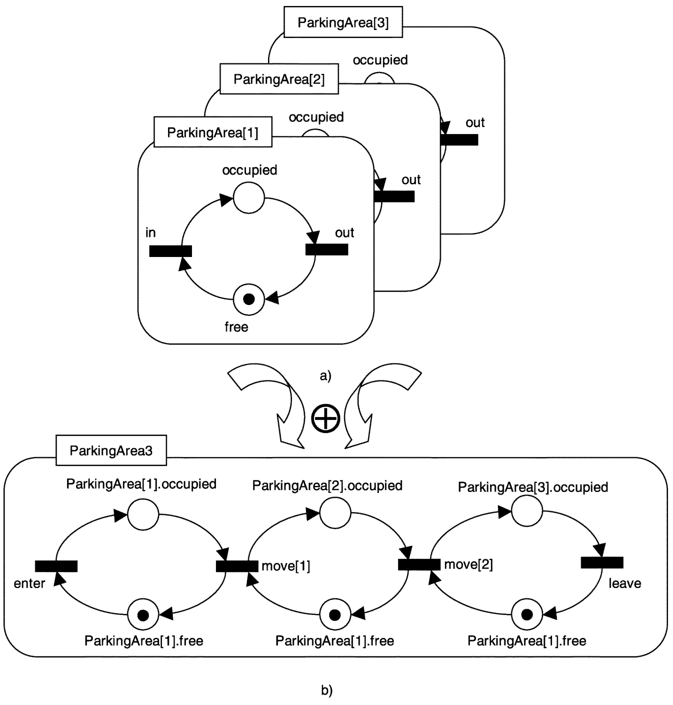

Fig. 5. Composition using net instances, through transition fusion, of the net models in (a) into the net model of (b).

Node-based folding together with hierarchical structuring mechanisms (to be discussed in the next section), can be a valuable support for the use of subnets as modules and their reuse in different designs [26]. Although, we only present single-dimensional vectors, the extension to support multidimensional vectors is straightforward.

Expressiveness equivalence: It has to be noted that both folding techniques (token-based and node-based) have the same modeling capability, in the sense that both can be translated to the same class of low-level Petri nets. The folding supported by the node-based vectorial notation can also be modeled through the addition of a new color attribute (as accomplished in Fig. 4). However, we believe that the vectorial notation in node-based folding, improves readability while allowing simpler and re-usable models. This is made more evident when we have submodels associated with nodes and hierarchical structuring techniques are used (to be presented in the next section). Using vectorial notation, relying on the usage of a multiplicity factor, there is no need to change the model associated with the component to be replicated and folded afterwards. This was exemplified in Fig. 5, where the initial low-level net model was used directly to compose the final model.

B. Refinement/Abstraction

In the Petri net literature, top-down decomposition and bottom-up construction are usually referred as **refinement** and **abstraction**, respectively.[2]

[2]Although in the context of a specific Petri nets class, the article by Xudong He and John Lee [31] offers a comprehensive presentation of both concepts.

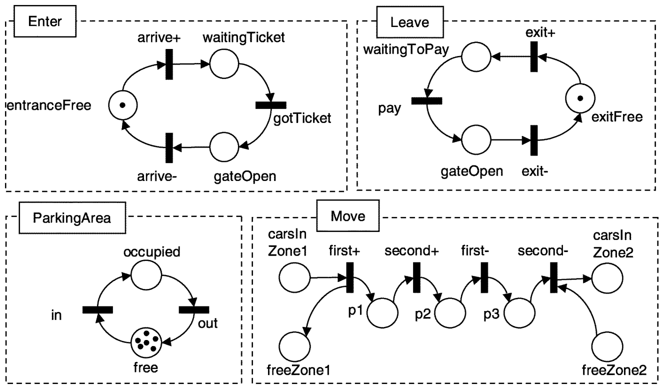

Fig. 6. Submodel (components) of a parking lot model.

A refinement adds an encapsulated net to an existing net. The former is named a **subnet**, whereas the latter is named a **supernet**. One of its nodes becomes a representation for the subnet. An abstraction is the reverse process: a subnet is replaced by a single node.

These hierarchical structuring mechanisms have been introduced to handle system model complexity, and associated difficulties of having too many details at one time, and losing a global overview of the system behavior. Model structuring through hierarchical refinement/abstraction mechanisms supports the definition of components and their reusability, taking advantage of graphical expressiveness capabilities in order to hide/show details of the model in a consistent manner.

The refinement/abstraction mechanism can be defined in static or dynamic form. Compared to the programming languages area, the static form corresponds to the concept of *macro*, whereas the dynamic form corresponds to a procedure invocation. The following subsections present these two forms.

1) Macros: Petri nets have two node types. As such, one intuitive way to hierarchically compose models is by node substitution: either place substitution, transitions substitution, or both. In all three, a node can be seen as containing another Petri net model. This use of places and transitions as encapsulated Petri nets goes back to the works of Carl Adam Petri [10] and was extensively presented in [54]. These nodes are named **macro nodes** and the associated Petri net is named a **Petri net macro** or simply a **macro**. These macros can also include macro-nodes, supporting a multilevel hierarchical composition mechanism. Hence, macros offer a vertical structuring mechanism, which originates a tree-like composition structure similar to object composition in object-oriented languages. We get nets inside nets through the use of nodes (places or transitions). This corresponds to a static composition of nets. Any page can be used repeatedly in the same design, although recursive (circular) references are not allowed, in order to avoid infinite substitution.

We consider that macro-nodes can assume three forms: macro-places, macro-transitions, and macro-blocks. Macro-transitions and macro-places are extensively used by different tools and analysis methods (namely reduction methods that can preserve proprieties). They were also initially considered for colored Petri nets [32], yet only the substitution transition (similar to the macro-transition concept) is currently supported by the associated tools Design-CPN [18] and CPN Tools [15]. The macro-place concept has been extensively used for reduction methods support, and within some application areas, like hardware design and manufacturing systems [54]. Macro-blocks were presented in [26]–[28], and correspond to the representation of a general module through a Petri net model.

At the super-page level, macro-places nodes have arcs connected to transitions (or to nodes with "transition semantics"), macro-transitions have arcs connected to places (or to nodes with "place semantics"), and macro-blocks have arcs connected to both places and transitions. Yet, for the later, as only the flat model is executed (the model that results after removing the hierarchical structuring mechanisms), the bipartite characteristic of Petri nets is not violated. In some works the interconnection between macro-nodes is not allowed (two macro-nodes cannot be adjacent, namely in [32]), whereas in others this feature is considered to be important from the engineering point of view (namely in [26]–[28]), as they allow to compose the model in a similar way as physical components.

By definition, a macro-transition corresponds to a macro-node where all the arcs are connected to places (or to nodes with "place semantics"). This implies that transitions constitute the boundary of the subnet. However, to allow syntax and semantic verifications at the subpage level, necessary for high-level nets, the actual connections to input places, and the respective arc inscriptions, are needed for those transitions. This is especially true for hierarchical colored Petri nets and for the inscriptions related to the colored binding evaluation. In this sense, tools that need to handle macro-modeling within high-level nets need to consider at the subpage level (where the model associated with the macro-node is stored) representations of the incoming/outgoing arcs and associated inscriptions and connecting places (to allow colored binding evaluation). It has to be stressed that for low-level nets those connecting arcs are not strictly necessary to be considered at the subpage level.

Therefore, in high-level nets, references to places in the super-page (the one containing the macro-transition) have to be included in the subpage. From the model execution point of view, the places connected to a macro-transition exist only at the super-page level; the places presented at the subpage are void (sometimes called reference or "ghost" places) and will be merged with the associated places at the super-page. This approach is followed for macro-transitions support by hierarchical colored Petri nets tools and others.

On the other hand, a macro-place corresponds to a macro-node where all the arcs are connected to transitions (or to nodes with "transition semantics"). The boundary of the subnet is composed by (real) places. From model execution point of view, the places that compose the boundary of the subpage exist only at the subpage level. To allow proper merging of the subnet into the super-page, a set of "interface ports" are used to assure correct connectivity for incoming/outgoing arcs at the super-page level into the model of the subpage level. This allows the direct interconnection of macro-nodes.

Finally, for macro-blocks, the merging process is also identical; for arcs with a macro-transition semantics, the procedures associated with macro-transition have to be followed; for arcs

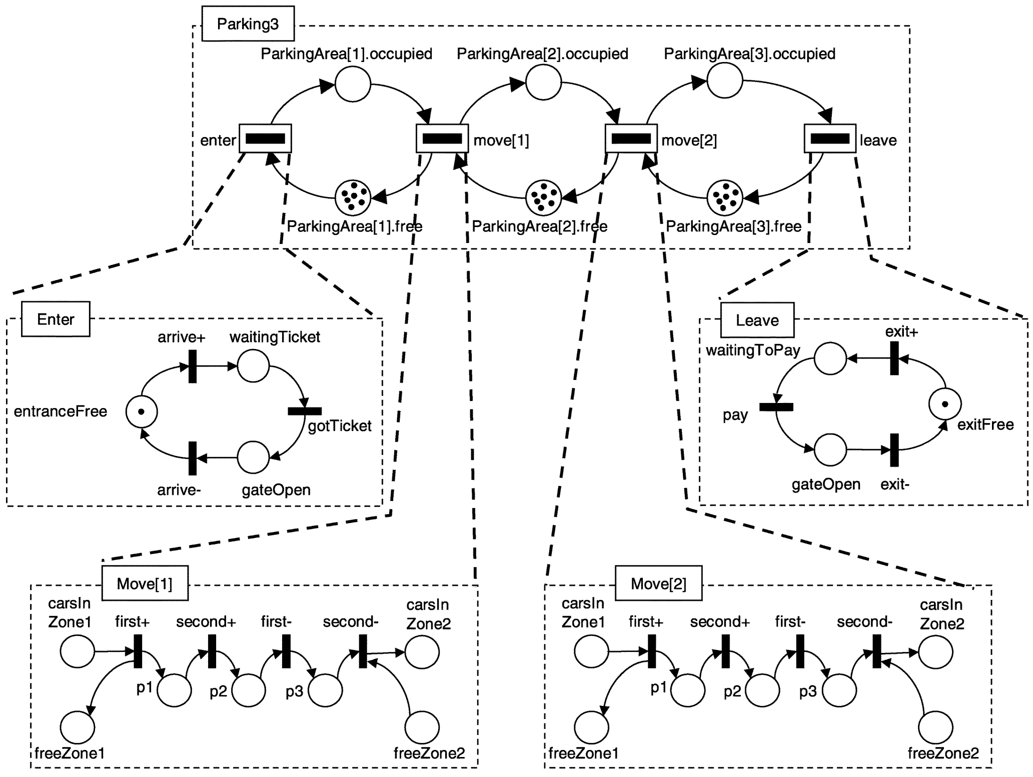

Fig. 7. Hierarchical decomposition of a parking lot model.

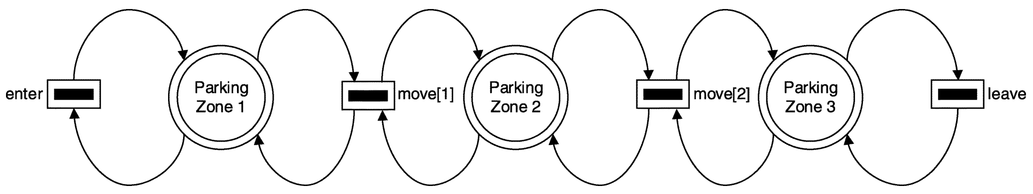

Fig. 8. Higher level model using macro places for the parking lot model of Fig. 7.

with a macro-place semantics, macro-place semantics has to be chosen.

For Petri net model execution, it is necessary to obtain the associated flat model resulting from the iterative merging of the lower level pages into the upper level pages. The referred hierarchical relations between super-nets and subnets are specified by macro-nodes at the upper page level. The following steps roughly describe the merging process of the subpage into the super-page.

- One copy of the subpage is inserted at the super-page (assuring that a unique reference is kept for all nodes); the macro-node reference is removed.
- The arcs connected with a macro-place semantics are connected to the referred boundary place; for arcs connected with a macro-transition semantics, void boundary places of the subpage are merged with the associated places at the super-page (arcs and associated arc inscriptions in the subpage are kept).

Now, coming back to our running example, and taking advantage of graphical simplicity associated with low-level nets, consider the net components in Fig. 6. The top of Fig. 7 shows the composition of three *ParkingZones* [as in Fig. 3(b)]. Consider that the four transitions on the model $Parking3$ in Fig. 7 are seen as abstractions for the net components $Enter$, $Move$, and $Leave$. Conversely, regarding the four transitions, two are refined by instances of the $Move$ component, one by the $Enter$, and one by the $Leave$.

In Fig. 7 (and in Fig. 8, as well), a distinctive notation is used to represent macro-nodes (containing an envelope), keeping original shape as reference. It is not mandatory to use such differentiation; as a matter of fact, many tools, e.g., CPN Tools, do not make any distinction at the graphical representation between transition and macro-transition.

A higher level view for the park model is possible if we also use macro places. This is illustrated in Fig. 8, where all nodes are macro-nodes. Notice the Petri net like appearance of this top-level model, allowing a higher abstraction model representation. There, every node can be associated with a high-level component of the system, either with a static one or with a dynamic one. Their interrelations are also explicitly represented. This constitutes a significant advantage when using macro-nodes.

Fig. 9 shows the equivalent "flat" net model, the model that can be used for execution and for analysis purposes, as well.

Interestingly, the abstraction/refinement structuring mechanism is supported by node fusion. This is discussed in the following section.

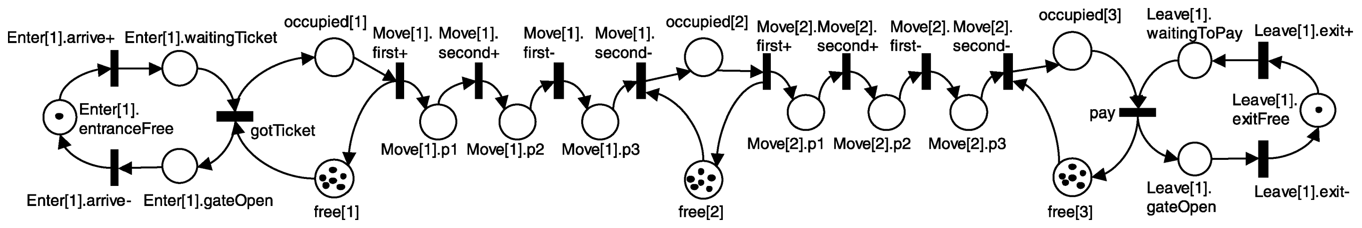

Fig. 9. Flat net model associated with the parking lot model of Fig. 7.

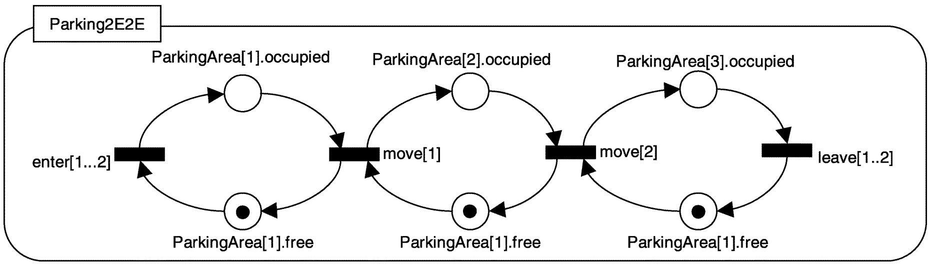

Fig. 10. Introduction of two entrances and two exits using node vectors.

Refinement as composition: In [32] macro transitions and macro places are presented in the context of high-level nets. Macro transitions are called substitution transitions and are connected to the super-net by means of place fusion. Substitution transitions are also part of hierarchical colored Petri nets [33]. They are implemented in the Design CPN [48] and CPN Tools [15] applications. In this way, hierarchical colored Petri nets support folding and composition.

In fact, a refinement is a conceptual construct. In concrete terms, it reduces to net composition by node fusion, as presented in Section II-A. For example, the model in Fig. 7, resulting in the Fig. 9 model, can be defined by the following net addition, i.e., the refinements are defined as compositions (where Enter and Leave are aliases for Enter[1] and Leave[1], as only one instance of Enter and Leave exist):

$$\begin{aligned}&Parking3\\&\quad=(ParkingArea3\oplus Enter\oplus Move[1\ldots2]\oplus Leave)\\&\qquad(Enter.gotTicket/ParkingArea3.enter\mapsto gotTicket,\\&\qquad\ (ParkingArea[i].ocuppied/Move[i].carsInZone1\\&\qquad\qquad\mapsto occupied[i],\\&\qquad\ ParkingArea[i].free/Move[i].freeZone1\mapsto free[i],\\&\qquad\ ParkingArea[i+1].ocuppied/Move[i].carsInZone2\\&\qquad\qquad\mapsto occupied[i+1],\\&\qquad\ ParkingArea[i+1].free/Move[i].freeZone2\\&\qquad\qquad\mapsto free[i+1])\quad[i:1\ldots2],\\&\qquad Leave.pay/ParkingArea3.leave\mapsto pay).\end{aligned}\tag{4}$$

In this sense, the concept of composability of models can be seen as the foundation for hierarchical representation through refinement/abstraction mechanisms to be used within tool implementation [26].

As an example of system evolution, illustrating the concept of incremental modeling, very important in complex systems design process and using several composition and structuring mechanisms at the same time, we now consider the need for two entrances and two exits for the previously presented parking lot model. This means that we need two instances of net $Enter$ and two instances of net $Leave$, that can be easily integrated using the vector notation, previously introduced. This is illustrated in Fig. 10.

This can be specified by another net addition:

$$\begin{aligned}&Parking2E2E\\&\quad=(ParkingArea3\oplus Enter[1\ldots2]\oplus Move[1\ldots2]\\&\qquad\quad\oplus Leave[1\ldots2])\\&\qquad(Enter[i].gotTicket/ParkingArea3.enter\\&\qquad\quad\mapsto gotTicket[i],\\&\qquad ParkingArea[i].ocuppied/Move[i].carsInZone1\\&\qquad\quad\mapsto ocuppied[i],\\&\qquad ParkingArea[i].free/Move[i].freeZone1\\&\qquad\quad\mapsto free[i],\\&\qquad ParkingArea[i+1].ocuppied/Move[i].carsInZone2\\&\qquad\quad\mapsto ocuppied[i+1],\\&\qquad ParkingArea[i+1].free/Move[i].freeZone2\\&\qquad\quad\mapsto free[i+1],\\&\qquad Leave[i].pay/ParkingArea3.leave\mapsto pay[i])\\&\qquad\quad[i:1\ldots2].\end{aligned}\tag{5}$$

The equivalent flat model is shown in Fig. 11.

From the previous paragraphs, it is clear that even using structuring mechanisms, Petri nets still hold long term characteristics: graphical expressiveness allied to formal representation (allowing support for analysis and propriety verification).

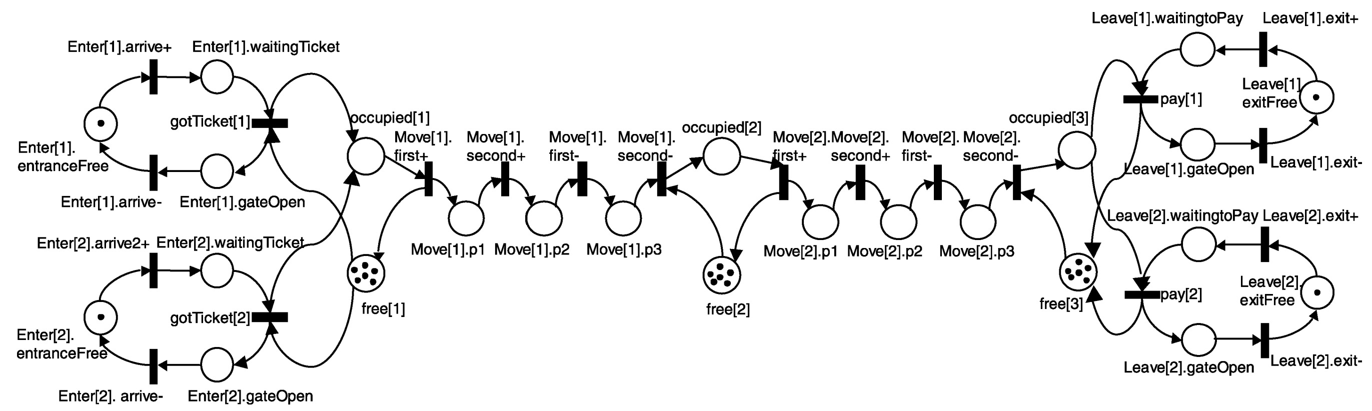

Fig. 11. Flat net model associated with the parking lot model of Fig. 10, with two entrances and two exits.

2) Invocations: Invocation is the dynamic counterpart for the static composition based on macros. Here, a node can be an **invocation node**, which creates new nets. When a transition fires, a new net is dynamically created. We can call this net a **procedure** and these dynamic net creations **procedure-invocations**. This construction is rarely proposed for Petri nets. Basically, because it radically changes the common Petri net semantics where the net structure is not modifiable. It also complicates the available analysis techniques. Even so, a few Petri nets extensions offer this type of vertical dynamic structuring.

Invocations are typically associated with the dynamic part of the Petri net: although we can think of a new net being invoked (created) by a place marking change, clearly this is not a "natural" solution. In fact, that token arrival was generated by a transition. So, the transition is the "natural" invoker (creator) of a new net.

If the tokens are seen as references to newly created nets, we are referring to **object nets** (see [56] for an up-to-date tutorial style presentation). Intuitively, object nets offer an "internal" (and also dynamic) composition just like the traditional Russian dolls (Matryoshkas): we get nets inside nets through the use of tokens instead of nodes. As tokens can be created and destroyed, we get a seamless way to create and destroy nets; this offers a much more dynamic structuring mechanism.

Among the numerous proposals for object based Petri nets [1], Lakos' proposal [41] also complies with the described structuring of dynamic objects as tokens inside places. Yet, we will focus on the object Petri nets by Valk, in particular the **reference nets** [38] variant due to its support by the RENEW tool [39], [58]. Differently from object nets, the reference nets are also colored nets: each token can be not only a low-level token but hold also a more complex data structure.

The main feature of reference nets is the ability to specify references to nets as another type of token. The transitions are able to create new net instances while returning the respective reference, which is then deposited in one or more places. This newly created net is then executed in parallel with all other net instances, including the net that created it. All the net instances can communicate among them through a high-level type of transition fusion called **synchronous channels**. These assume a direction of invocation between the transitions: the receiving transition only fires when the sending transition fires. This is different from the initial synchronous channel definition [12]. The synchronous channel acts very much as a method (or procedure) calling: we have parameters and a direction of invocation. Also, the direction of information transfer (through the parameters) can be different from the direction of invocation: the parameters can act as input, output, or input/output parameters.[3] Basically, a synchronous channel allows a unidirectional "fusion" of the involved transition, including the respective variables.

The model execution starts by a chosen net for which an initial instance is automatically created. The other net instances are then created, directly or indirectly by this net instance. Next, we apply a reference net to model some additional behaviors in the parking lot example.

A reference net model: As tokens can be nets, a common interpretation for object net models is in terms of objects, or *agents*, going from one location to another. Naturally, these objects should have an interesting behavior, which is then modeled by another net. This is the case in our running example (the parking lot controller with parking zones 1 to 3), if we want to model the possibility to react to cars moving backward between two parking zones. In that case, from the model point of view, the cars' behavior becomes significant: they can be moving forward, from zone i to zone $i+1$, or backward, from zone i to zone $i-1$; they can also start forward or backward, and go reverse during the passage. We consider that it is not allowed to go from zone i to zone $i-1$, or coming back to zone i after started the passage to zone $i+1$; in those situations an alarm should be issued, and a photo should be taken for those that returned to previous zone. Fig. 12 shows the corresponding model, named *Passage*. It is an expanded model relatively to the $Move$ net (see Fig. 7), but modeled as a reference net, like the one supported by the RENEW tool. As before, the tokens represent cars, but now

[3]A recent proposal for synchronous channels by the authors of this article makes this distinction explicit [4].

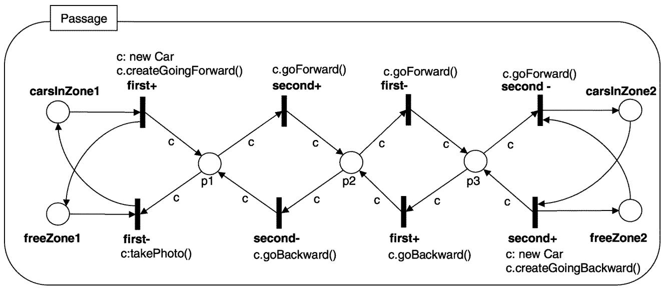

Fig. 12. Introducing reverse direction detection capability.

these tokens are also nets.[4] These nets are instances of a *Car* net (see Fig. 13), in other words they are net *Car instances*, each one with its own behavior. The *Car* instances communicate with the *Passage* net instance through **synchronous channels**. We will have one *Car* instance for each car that is currently crossing the passage.

This *Passage* net foresees the possibility of a car going back (from parking zone $i + 1$ to parking zone i) in the middle of its move from parking zone i to parking zone $i + 1$. It also foresees the possibility of a car entering the passage in reverse direction, from parking zone $i + 1$ to parking zone i, through transition *second+* at the bottom-right of Fig. 12. In both cases an alarm is fired until the car returns to the allowed movement direction (from area i to area i + 1). The names *first+/second+*, *first-/second-*, model input event signals corresponding to passage detection sensors with positive or negative edge triggers, respectively. If a car returns to a previous zone in the reverse direction, a photo is taken (transition *first-* at the bottom-left of Fig. 12).

Transition *first+* at the top-left of Fig. 12 creates a new net of class *Car* (see Fig. 13) and assigns the respective reference to variable *c*: "*c: new Car*". It also fuses through a synchronous channel named *createGoingForward*, with the transition *createForward* in net *Car* which has the same associated synchronous channel. The *Car* net models two possible states for the *Car* objects that flow along the passage: *movingForward* or *movingBackward*. From the passage net point of view, these either means going from parking area 1 to parking area 2, or going from parking area 2 to parking area 1, respectively.

When a car, that is moving forward, decides to move backward (transition *goBack* in Fig. 13), a new alarm net (see Fig. 14) is created and the transition *create*, with the *start* synchronous channel, is fired. Note that the *start* synchronous channel has one parameter, the *Car* instance, which in the *Car* net is specified by the reserve word *this*. The alarm stops when the transitions *goForth* or *takePhotoB* of Fig. 13 are fired. These transitions also consume the reference a to the *Alarm* net. As no other reference exists and the *Alarm* net is dead (no further firings are possible), the alarm net instance can be garbage collected. Something totally similar happens with the references to *Car* nets, through transitions *first-* at the bottom-left of Fig. 12 and *second-* at the top-right of Fig. 12 in the *Passage* net.

[4]As already mentioned, they are *references* to net instances.

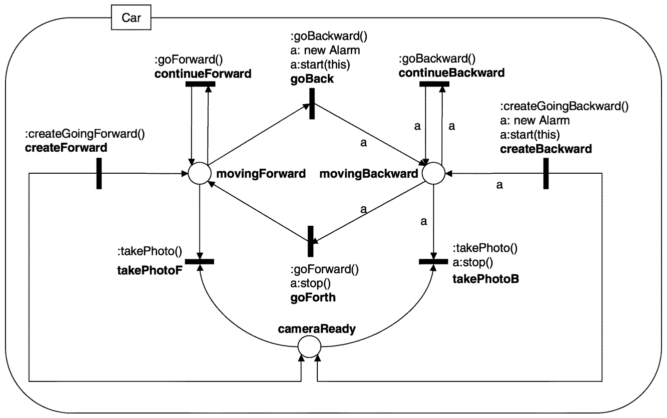

Fig. 13. The *car* net.

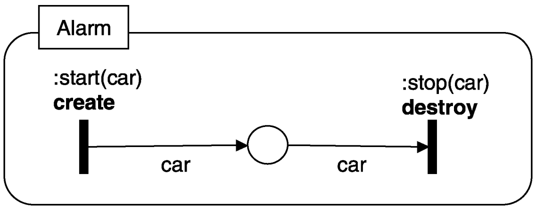

Fig. 14. The *alarm* net.

A car that is moving forward and continues its movement without modification is modeled by the *goForward* synchronous channel in *Passage* net transitions *second+*, *first-*, and *second-*. Similarly, a car that is moving backward and continues its movement without modification is modeled by the *goBackward* synchronous channel in *Passage* net transitions *second-* and *first+*.

The control system takes a photo to each car that crosses the sensor *first-*. This is modeled as a synchronous channel, named *takePhoto()*, between transition *first-* in net *Passage*, and transitions *takePhotoF* and *takePhotoB* in net *Car*: one for each of the two possible car states, *movingForward* and *movingBackward*, respectively.

III. Conclusion

This paper presented a condensed view of the main structuring mechanisms for general system modeling with Petri nets. Starting from a clearly pragmatic point of view, it proposes a general classification as a starting point for a broad, although brief, presentation of the most significant structuring constructs.

In spite of the numerous Petri nets based tools available [48], we briefly referred to the CPN Tools and the Renew tool as these are the two most significant Petri net tools for general system modeling. Other tools (e.g., Maria [43]) are especially interesting for verification purposes.

We conclude by identifying three main research areas capable of providing significant advances for Petri nets structuring and composability mechanisms: abstraction, synchronous communication, and net transformations.

Abstraction: Unfortunately, neither macros nor invocations are clear examples of abstractions in the sense defined in [29]: "By abstraction we mean the act of singling out a few properties of an object for further use or study, omitting from consideration other properties that don't concern us for the moment."

This definition clearly views abstractions as a property filter; also, it clearly supports a view of abstraction as a simplified representation for a given object. At the super-net, macros and invocations are usually represented as a special kind of place or transition associated to another net. So as to preserve a minimal semantic similarity with places and transitions, macro-places must interface with the super-net elements exclusively by places, macro-transitions must interface with the super-net elements exclusively by transitions. Yet macro-places do not have markings and macro-transition firing is not instantaneous as it is made dependent on subpage transitions. Clearly, this semantic similarity falls short on what should be expected from a node abstraction.

To the authors best knowledge, only a few works in literature have tried to address this problem, namely [9], [40], and [42]. Yet, to our best knowledge no tool implements the discussed proposals.

Synchronous communication: Synchronous channels also allow several possible, and potentially useful, semantics. In particular, one can think of, at least, two aspects, each one allowing different possible semantics.

1) **Channel symmetry**: Channels can be defined as either symmetrical or directional. The former is the most similar to plain transition fusion, where no direction of communication is intended. This is the case with the original proposal of Christensen and Hansen [12]. The latter corresponds to unidirectional invocation: send and receive sides for each synchronous channel. This approach is used in the RENEW tool and also in the authors' proposal [4],
2) **Multiplicity**: Each transition can have one or more associated channels. Then, each one can correspond to a send or a receive side. This originates several distinct possibilities, namely, multicast, broadcast, and point-to-point synchronous communication.

These are particular relevant because asynchronous message passing can easily be modeled by synchronous communication.

Net transformations: Finally, another interesting research direction is provided by the strong connections between graph grammars and Petri net transformations, which provide a solid theoretical ground for net transformation specifications. See [21] for an up-to-date tutorial presentation.

We believe that a simple and intuitive algebraic notation for net modifications, as the one provided by net addition [3], [26] and referred before in this paper, is an interesting basis for future developments that would allow the effective use, re-use and modification of Petri net models.

REFERENCES

[1] G. Agha, F. de Cindio, and G. Rozenberg, Eds., *Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets*. New York: Springer, 2001, vol. 2001, Lecture Notes in Computer Science.

[2] J. P. Barros and L. Gomes, "Modifying Petri net models by means of crosscutting operations," in *Proc. 3rd Int. Conf. Application of Concurrency to System Design*, Jun. 2003, pp. 177–186.

[3] ——, "Net model composition and modication by net operations: a pragmatic approach," in *Proc. 2nd IEEE Int. Conf. Industrial Informatics (INDIN'04)*, Berlin, Germany, Jun. 2004.

[4] ——, "On the use of colored Petri nets for object-oriented design," in *Applications and Theory of Petri Nets 2004 25th Int. Conf., ICATPN 2004*, vol. 3099, Proceedings Series: Lecture Notes in Computer Science, J. Cortadella and W. Reisig, Eds., Bologna, Italy, Jun. 2004, pp. 117–136.

[5] E. Battiston, F. de Cindio, and G. Mauri, "OBJSA nets: a class of high-level nets having objects as domains," in *Advances in Petri Nets 1988*, 1988, Lecture Notes in Computer Science, vol. 340, pp. 20–43.

[6] L. Bernardinello and F. De Cindio, A survey of basic net models and modular net classes, in Lecture Notes in Computer Science; Advances in Petri Nets 1992, vol. 609, pp. 304–351, 1992.

[7] E. Best, R. Devillers, and M. Koutny, *Petri Net Algebra*. New York: Springer, Monographs in Theoretical Computer Science. An EATCS Series, 2001.

[8] W. Brauer, R. Gold, and W. Vogler, A survey of behavior and equivalence preserving refinements of Petri nets, in Advances in Petri Nets 1990, vol. 483, pp. 1–46, 1991, Lecture Notes in Computer Science.

[9] P. Buchholz, Hierarchical high level Petri nets for complex system analysis, in *Application and Theory of Petri Nets 1994, Proc. 15th Int. Conf.*, R. Valette, Ed., Zaragoza, Spain, vol. 815, pp. 119–138, 1994, Lecture Notes in Computer Science.

[10] C. A. Petri, "Kommunikation mit Automaten," Ph.D. dissertation, Univ. Bonn, Bonn, Germany, 1962.

[11] Y. Chen, W. T. Tsai, and D. Chao, "Dependency analysis-a Petri-net-based technique for synthesizing large concurrent systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 4, pp. 414–426, Apr. 1993.

[12] S. Christensen and N. D. Hansen, "Coloured Petri Nets extended with channels for synchronous communication," in *Lecture Notes in Computer Science; Application and Theory of Petri Nets, Proceedings of the 15th International Conference*, R. Valette, Ed. Zaragoza, Spain: Springer-Verlag, 1994, vol. 815, pp. 159–178.

[13] S. Christensen and L. Petrucci, "Modular state space analysis of colored Petri nets," in *Proc. 16th Int. Conf. Application and Theory of Petri Nets*, Turin, Italy, Jun. 1995, [Online] Available at: http://www.daimi.aau.dk/CPnets/publ/full-papers/ChrPet1995.pdf, pp. 201–217.

[14] ——, "Modular analysis of Petri nets," *Comput. J.*, vol. 43, no. 3, pp. 224–242, 2000.

[15] (2004) CPN Tools. [Online] Available at: http://wiki.daimi.au.dk/cpn-tools

[16] R. David and H. Alla, *Petri Nets & Grafcet; Tools for Modeling Discrete Event Systems*. Hempstead, U.K.: Prentice-Hall International, 1992.

[17] R. David, "Modeling of dynamic systems by Petri nets," in *Proc. Eur. Control Conf.*, Grenoble, France, Jul. 1991, pp. 136–147.

[18] (2004) Design/CPN. [Online] Available at: http://www.daimi.au.dk/designCPN/

[19] E. Y. T. Juan, J. J. P. Tsai, and T. Murata, "A new compositional method for condensed state-space verification," in *Proc. 1st IEEE High-Assurance Systems Engineering Workshop*, Ontario, Canada, Oct. 22, 1996, pp. 104–111.

[20] H. Ehrig, G. Juhás, J. Padberg, and G. Rozenberg, Eds., *Unifying Petri Nets, Advances in Petri Nets*. Berlin, Germany: Springer, 2001, vol. 2128, Lecture Notes in Computer Science.

[21] H. Ehrig and J. Padberg, "Graph grammars and Petri net transformations.," in *Lectures on Concurrency and Petri Nets*, J. Desel, W. Reisig, and G. Rozenberg, Eds. Berlin/Heidelberg, Germany: Springer-Verlag, 2004, vol. 3098, Lecture Notes in Computer Science: Advances in Petri Nets, pp. 496–536.

[22] E. Y. T. Juan, J. J. P. Tsai, and T. Murata, "Compositional verification of concurrent systems using Petri net based condensation rules," *ACM Trans. Program. Lang. Syst.*, vol. 20, no. 5, pp. 917–979, 1998.

[23] R. Fehling, "A concept of hierarchical Petri nets with building blocks," in *Proc. 12th Int. Conf. Application and Theory of Petri Nets, 1991*, Gjern, Denmark, Jun. 1991, pp. 370–389.

[24] H. J. Genrich, "Predicate/transition nets," in *Lecture Notes in Computer Science: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course*, W. Brauer, W. Reisig, and G. Rozenberg, Eds., 1987, vol. 254, pp. 207–247.

[25] C. Girault and R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*, 2003.

[26] L. Gomes and J. P. Barros, "On structuring mechanisms for Petri nets based system design," in *Proc. 2003 IEEE Conf. Emerging Technologies and Factory Automation (ETFA 2003)*, Sep. 2003, pp. 431–438.

[27] L. Gomes, J. P. Barros, and A. Costa, "Petri net model node structuring techniques for embedded system design," in *Proc. 5th Portuguese Conf. Automatic Control (CONTROLO'2002)*, Aveiro, Portugal, Sep. 2002.

[28] ——, "Structuring mechanisms in Petri net models: from specification to FPGA based implementations," in *Design of Embedded Control Systems*, M. Adamski, A. Karatkevich, and M. Wegrzyn, Eds. Berlin, Germany: Springer, 2004.

[29] D. Gries, *The Science of Programming*. Berlin, Germany: Springer-Verlag, 1981.
[30] X. He, "A formal definition of hierarchical predicate transition nets," in *Lecture Notes in Computer Science; Proc. 17th Int. Conf. Application and Theory of Petri Nets (ICATPN'96)*, vol. 1091, Osaka, Japan, Jun. 1996, pp. 212–229.
[31] X. He and J. A. N. Lee, "A methodology for constructing predicate transition net specifications," *Software-Practice Exper.*, vol. 21, no. 8, pp. 845–875, Aug 1991.
[32] P. Huber, K. Jensen, and R. M. Shapiro, "Hierarchies in colored Petri nets," in *Proc. 10th Int. Conf. Application and Theory of Petri Nets, 1989*, Bonn, Germany, 1989, pp. 192–209.
[33] K. Jensen, *Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use—Volumes 1–3*. Berlin, Germany: Springer-Verlag, 1992–1997.
[34] ——, "Condensed state spaces for symmetrical colored Petri nets," *Formal Meth. Syst. Design*, vol. 9, pp. 7–40, 1996.
[35] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*. Berlin, Germany: Springer-Verlag, 1991.
[36] G. Juhs and J. Desel, What is a Petri Net, pp. 1–25.
[37] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proc. 11th European Conf. Object-Oriented Programming*, vol. 1241, LNCS, M. Akşit and S. Matsuoka, Eds., Berlin/Heidelberg/New York, 1997, pp. 220–242.
[38] O. Kummer, *Referenznetze*. Berlin, Germany: Logos Verlag, 2002.
[39] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, and R. Valk, "An extensible editor and simulation engine for Petri nets: RENEW," in *Applications and Theory of Petri Nets 2004 25th Int. Conf.*, vol. 3099, Proceedings Series: Lecture Notes in Computer Science, J. Cortadella and W. Reisig, Eds., Bologna, Italy, Jun. 2004, pp. 484–493.
[40] C. Lakos, "Composing abstractions of colored Petri nets," in *Lecture Notes in Computer Science: 21st Int. Conf. Application and Theory of Petri Nets (ICATPN 2000)*, vol. 1825, M. Nielsen and D. Simpson, Eds., Aarhus, Denmark, Jun. 2000, pp. 323–345.
[41] ——, "Object oriented modeling with object Petri nets," in *Concurrent Object-Oriented Programming and Petri Nets*, G. A. Agha, F. De Cindio, and G. Rozenberg, Eds. Berlin/ Heidelberg, Germany: Springer-Verlag, 2001, vol. 2001, Lecture Notes in Computer Science: Advances in Petri Nets, pp. 1–37.
[42] ——, "On the abstraction of colored Petri nets," in *Lecture Notes in Computer Science: 18th Int. Conf. Application and Theory of Petri Nets*, vol. 1248, P. Azéma and G. Balbo, Eds., Toulouse, France, Jun 1997, pp. 42–61.
[43] M. Mäkelä. (2004) Maria the Modular Reachability Analyzer. [Online] Available: http://www.tcs.hut.fi/Software/maria/
[44] T. Murata, "Petri nets: properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[45] M. Notomi and T. Murata, "Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis," *IEEE Trans. Softw. Eng.*, vol. 20, no. 4, pp. 325–336, 1994.
[46] J. Padberg, "Petri net modules," *Trans. SDPS*, vol. 6, no. 3, pp. 121–196, Sep. 2002.
[47] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, 1977.
[48] Petri Nets. (2004) Tool Database. [Online] Available: http://www.daimi.au.dk/PetriNets/tools/db.html
[49] W. Reisig, "Petri nets with individual tokens," in *Informatik-Fachberichte 66: Application and Theory of Petri Nets—Selected Papers from the 3rd European Workshop on Application and Theory of Petri Nets*, A. Pagnoni and G. Rozenberg, Eds., Varanna, Italy, Sep. 1982, pp. 229–249.
[50] ——, *Petri Nets: An Introduction*. New York: Springer-Verlag, 1985.
[51] ——, *A Primer in Petri Net Design*. New York: Springer-Verlag, 1992.
[52] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets I: Basic Models.*. Berlin, Germany: Springer-Verlag, 1998, vol. 1491, Lecture Notes in Computer Science; Advances in Petri Nets.
[53] W. Reisig and G. Rozenberg, Eds., *Lectures on Petri Nets II: Applications.*, Germany: Springer-Verlag, 1998, vol. 1492, Lecture Notes in Computer Science; Advances in Petri Nets.
[54] M. Silva, *Las Redes de Petri: en la Automática y la Informática*. Madrid, Spain: Editorial AC, 1985.
[55] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton Jr., "N degrees of separation: multi-dimensional separation of concerns," in *Proc. 21st Int. Conf. Software Engineering*, 1999, pp. 107–119.
[56] R. Valk, "Object Petri nets," in *Lectures on Concurrency and Petri Nets*, J. Desel, W. Reisig, and G. Rozenberg, Eds. Berlin/Heidelberg: Springer-Verlag, 2004, vol. 3098, Lecture Notes in Computer Science: Advances in Petri Nets, pp. 819–848.
[57] A. Valmari, "Compositional analysis with place-bordered subnets," in *Proc. 15th Int. Conf.*, vol. 815, Lecture Notes in Computer Science; Application and Theory of Petri Nets, R. Valette, Ed., Zaragoza, Spain, 1994, pp. 531–547.
[58] (2004) RENEW The Reference Net Workshop. [Online] Available: http://www.renew.de/
[59] R. Zurawski and M. Zhou, "Petri nets and industrial applications—a tutorial," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 567–583, Dec. 1994.

Luís Gomes (M'96) received the Electrotech. Eng. degree from Universidade Técnica de Lisboa, Lisboa, Portugal, in 1981, and the Ph.D. degree in digital systems from the Universidade Nova de Lisboa in 1997.

He is a Professor at the Electrical Engineering Department, Faculty of Sciences and Technology, Universidade Nova de Lisboa, and a Researcher at UNINOVA Institute, Portugal. From 1984 to 1987, he was with EID, a Portuguese medium enterprise in the area of electronic system design, in the R&D Engineering Department. His main interests include the usage of formal methods, like Petri nets and other concurrency models, applied to reconfigurable and embedded systems co-design.

Dr. Gomes served as the General Co-Chair for the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA-2003) and was responsible for the organization of the Petri Nets'98 International Conference on Applications and Theory of Petri nets, both held in Lisbon.

João Paulo Barros received the Lic. and M.Sc. degrees in informatics engineering in 1993 and 1997, respectively, from the Universidade Nova de Lisboa, Lisboa, Portugal, where he is currently pursuing the Ph.D. degree.

He is currently an Adjunct Professor at the Instituto Politécnico de Beja, Beja, Portugal, and a Researcher at the UNINOVA Institute, Lisboa. His research interests include Petri nets, visual specification languages, and languages and tools for object-oriented software development. He is also particularly interested in themes related to computer science education.

Mr. Barros is a member of the ACM and the ACM Special Interest Group on Computer Science Education.