A Petri Net Perspective on the Resource Allocation Problem in Software Engineering*

Juan-Pablo López-Grao[1] and José-Manuel Colom[2]

[1] Dpt. of Computer Science and Systems Engineering (DIIS)
[2] Aragon Institute of Engineering Research (I3A)
University of Zaragoza, Spain
{jpablo,jm}@unizar.es

Abstract. Resource Allocation Systems (RAS) were intensively studied in the last years for Flexible Manufacturing Systems (FMS). The success of this research stems from the identification of subclasses of Petri Nets that correspond to an RAS abstraction of these systems. In this paper we take a parallel road to that travelled through for FMS, but for the case of software applications. These applications present concurrency, and deadlocks can happen due to the allocation of shared resources. We reveal that the existing subclasses of Petri Nets used to study this kind of deadlock problems are insufficient, even for very simple software systems. We propose a new subclass of Petri Nets that generalizes the previously known RAS subclasses and we present a taxonomy of anomalies that can be found in the context of software systems.

1 Introduction

Among the most recurrent patterns in a wide range of engineering disciplines, the competition for shared resources among concurrent processes takes a prominent position. The perspective of discrete event systems theory proves appropriate and powerful as a framework in which solutions are provided to the so-called Resource Allocation Problem (RAP) [14]. Systems of this kind are often called Resource Allocation Systems (RAS) [3,15].

RAS are usually conceptualized around two distinct entities, processes and resources, thanks to a prior abstraction process which is inherent in the discipline. RAP refers to satisfying successfully the requests for resources made by the processes, ensuring that no process ever falls in a deadlock. A set of processes is deadlocked when they indefinitely wait for resources that are already held by other processes of the same set [2].

We focus on Sequential RAS with serially reusable resources [19]. Therefore, a process can increase or decrease the quantity of free resources during its execution. Still, resources are used conservatively.

Although other models of concurrency have also been considered [8], Petri nets [17] have arguably taken a leading role in the family of formal models

* This work has been partially supported by the European Community's 7th Framework Programme under Project DISC (Grant Agreement n. INFSO-ICT-224498).

K. Jensen, S. Donatelli, and J. Kleijn (Eds.): ToPNoC V, LNCS 6900, pp. 181–200, 2012.

used for dealing with RAP [4,6]. One of the strengths of this approach is the smooth mapping between the main entities of RAS and the basic elements of Petri nets. A resource type can be modelled using a place; its instances are modelled with tokens. Meanwhile, sequential processes are modelled with tokens progressing through state machines. Arcs from resource places to transitions (from transitions to resource places) represent the acquisition (release) of some resources by a process. Petri nets thus provide a natural formal framework for the analysis of RAS, besides benefiting from the goods of compositionality.

This fact is well recognized in the domain of Flexible Manufacturing Systems (FMS), where Petri net models for RAS have widely succeeded since the work of Ezpeleta et al. was introduced [4]. This is based upon two solid pillars: 1) the definition of a rich syntax from a physical point of view, which enables the natural expression of a wide disparity of plant configurations; and 2) the contribution of sound scientific results which let us characterize deadlocks from the model structure, as well as provide a well-defined methodology to automatically correct them in the real system.

Nowadays, there exists a plethora of Petri net models for modelling RAS in the context of FMS, which often overcome syntactic limitations of the S^3PR class [4]. S^4PR nets [22,18] generalize the earlier, while allowing multiple simultaneous resource allocations per process. S^*PR nets [7] extend the expressive power of the processes to that of state machines. Other classes such as NS-RAP [6], ERCN-merged nets [23] or PNR nets [13] extend the capabilities of S^3PR/S^4PR models beyond Sequential RAS by way of lot splitting or merging operations.

Most analysis and control techniques in the literature are based on a structural element which characterizes deadlocks in many RAS models: the so-called *bad siphon*. A bad siphon is a siphon which is not the support of a p-semiflow. If bad siphons become (sufficiently) emptied, their output transitions die since the resource places of the siphon cannot regain tokens anymore, thus revealing the *deadly embrace*. Control techniques thus rely on the insertion of monitor places [12] which limit the leakage of tokens from the bad siphons.

Although there exist obvious resemblances between RAP in FMS and that of parallel or concurrent software, previous attempts to bring these well-known RAS techniques into the field of software engineering have been, to the best of our knowledge, either too limiting or unsuccessful. In this work, we analyze why the net classes and results introduced in the context of FMS can fail when brought to the field of concurrent programming.

Section 2 presents a motivating example and discusses the elements that an RAS net model should desirably feature in order to successfully explore RAP within the software engineering discipline. Taking into account those considerations, Sect. 3 introduces a new Petri net class, called PC^2R. Section 4 relates the new class to those defined in our previous works and forewarn us about new behavioural phenomena. Some of these anomalies highlight the fact that previous theoretical results in the context of FMS are insufficient in the new framework. Section 5 summarizes the results of the paper. Finally, some useful definitions and basic concepts of Petri nets are provided as reference in appendix A.

2 The RAS View of a Software Application

Example 1 presents a humorous variation of Dijkstra's classic problem of the dining philosophers which adopts the beautiful writing by Hoare at [11].

Example 1. The postmodern dining philosophers. "Five philosophers spend their lives thinking and eating. The philosophers share a common dining room where there is a circular table surrounded by five chairs, each belonging to one philosopher. A microwave oven is also available. In the center of the table there is a large bowl of spaghetti which is frequently refilled (so it cannot be emptied), and the table is laid with five forks. On feeling hungry, a philosopher enters the dining room, sits in his own chair, and picks up the fork on the left of his place. Then he touches the bowl to feel its temperature. If he feels the spaghetti got too cold, he leaves his fork and takes the bowl to the microwave. Once it is warm enough, he comes back to the table, sits on his chair and leaves the bowl on the table after recovering his left fork. Unfortunately, the spaghetti is so tangled that he needs to pick up and use the fork on his right as well. If he can do this before the bowl gets cold again, he serves himself and starts eating. When he has finished, he puts down both forks and leaves the room."

According to the classic RAS nomenclature, each philosopher is a sequential process, and the five forks plus the bowl are serially reusable resources which are shared among the five processes. From a software perspective, each philosopher can be a process or a thread executed concurrently.

Algorithm 1 introduces the code for each philosopher. Notationally, we modelled the acquisition / release of resources by way of the `wait()` / `signal()` operations, respectively. Both of them have been generalized for the acquisition of multiple resources (separated by commas when invoking the function). Finally, the `trywait()` operation is a non-blocking wait operation. If every resource is available at the time `trywait()` is invoked, then it acquires them and returns `TRUE`. Otherwise, `trywait()` will return `FALSE` without acquiring any resource. For the sake of simplicity, it is assumed that the conditions with two or more literals are evaluated atomically.

Figure 1 depicts the net for Algorithm 1, with $i = 1$, after abstracting the relevant information from an RAS perspective. Figure 2 renders the composition of the five philosopher nets via fusion of the common shared resources. Note that if we remove the dashed arcs from Fig. 2, then we can see five disjoint strongly connected state machines plus six isolated places.

Each state machine represents the control flow for a philosopher. Every state machine is composed of seven states (places). Tokens in a state machine represent concurrent processes/threads which share the same control flow. At the initial state, every philosopher is thinking (outside the room), i.e. the unique token in each machine is located at the so-called *idle place*. In general, the idle place can be seen as a mechanism which limits the number of concurrent *active threads*. Here, at most one philosopher of type i can be inside the room, for each $i \in \{1, ..., 5\}$.

The six isolated places are called *resource places*. A resource place represents a resource type, and the number of tokens in it represents the quantity of free

instances of that resource type. In this case, every resource place is monomarked. Thus, at the initial state there is one fork of type i, for every $i \in \{1, ..., 5\}$, plus one bowl of spaghetti (modelled by the resource place at the centre of the figure).

Finally, the dashed arcs represent the acquisition or release of resources by the active threads when they change their execution state. Every time a transition fires, the total amount of resources available is altered. Please note, however, that moving one isolated token of a state machine (by firing its transitions) until the token reaches back the idle state, leaves the resource places marking unaltered. Thus, the resource usage is conservative.

Algorithm 1 – Code for Philosopher i (where $i \in \{1, 2, 3, 4, 5\}$)

```
var
     fork: array [1..5] of semaphores; // shared resources
     bowl: semaphore;                  // shared resource
begin
   do while (1)
      THINK;
      Enter the room;
(T1)  wait(fork[i]);                        (T6)
      do while (not(trywait(bowl, fork[i mod 5 +1]))
                or the spaghetti is cold)
(T2)     if (trywait(bowl)
                and the spaghetti is cold) then
(T3)        signal(fork[i]);
            Go to the microwave;
            Heat up spaghetti;
            Go back to table;
(T4)        wait(fork[i]);
(T5)        signal(bowl);
         end if;
      loop;
      Serve spaghetti;
(T7)  signal(bowl);
      EAT;
(T8)  signal(fork[i], fork[i mod 5 +1]);
      Leave the room;
   loop;
```

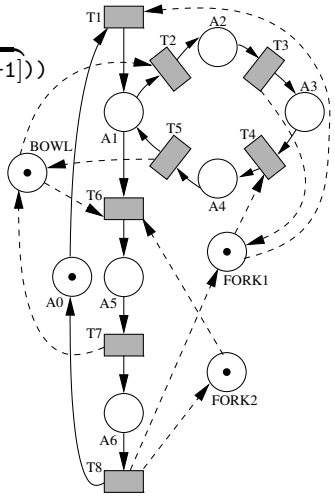

Fig. 1. Philosopher 1

At this point, we discuss some capabilities that an RAS model should have so as to support the modelling of concurrent programs.

State machines without internal cycles are rather versatile for modelling sequential processes in the context of FMS. The sucess of the S^3PR and S^4PR classes, in which every circuit in the state machines traverses the idle place, proves this. Nevertheless, this is clearly too constraining even for very simple software systems. Considering Böhm and Jacopini's theorem [10], however,

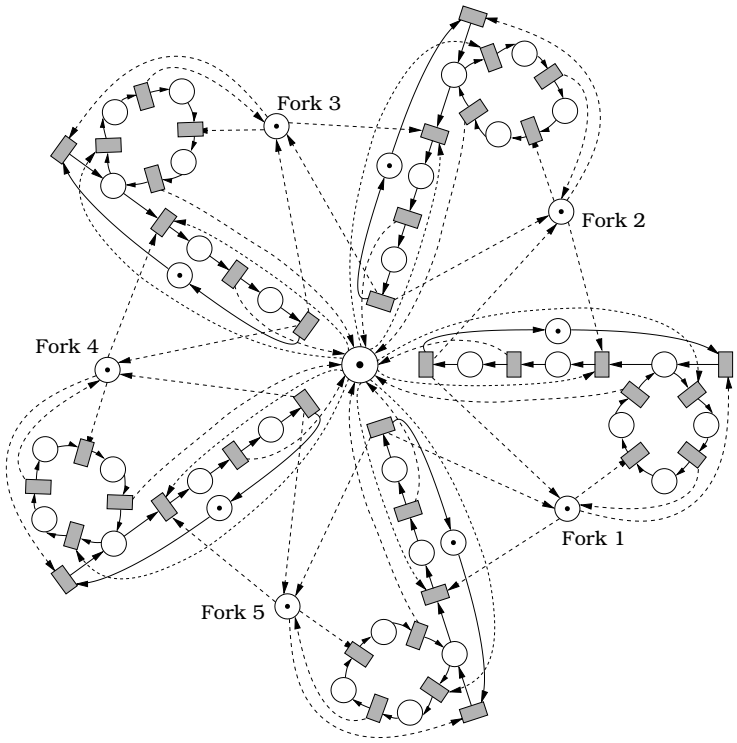

Fig. 2. The dining philosophers are thinking. Arcs from/to P_R are dashed for clarity.

we can assume that every non-structured sequential program can be refactored into a structured one using `while-do` loops. Meanwhile, calls to procedures and functions can be substituted by inlining techniques. Let us also remind that `fork`/`join` operations can also be unfolded into isolated concurrent sequential processes, as evidenced in [6]. As a result, we can restrict process models to state machines in which decisions and iterations (in the form of `while-do` loops) are supported, but not necessarily every kind of unstructured branch.

Another significant difference between FMS and software systems from an RAS perspective is that resources in the latter are not necessarily physical (e.g., a file) but can also be logical (e.g., a semaphore). This has strong implications in the degree of freedom in allocating those resources: we return to this issue later.

In this domain, a resource is an object that is shared among concurrent processes/threads and must be used in mutual exclusion. Since the number of resources is limited, the processes compete for the resource and use it in a non-preemptive way. This particular allocation scheme can be imposed by the resources' own access primitives, which may be blocking. Otherwise, the resource can be protected by a binary semaphore/mutex/lock (if there is only one instance of that resource type) or by a counting semaphore (multiple instances). Note that this kind of resources can be of varied nature (e.g., shared memory

locations, storage space, database table rows) but the required synchronization scheme is inherently similar.

On the other hand, it is well-known that semaphores used in that context can also be seen as non-preemptive resources which are used in a conservative way. For example, a counting semaphore that limits the number of connections to a database can be interpreted in that way from an RAS point of view. Here processes wait for the semaphore when attempting to establish a database connection, and release it when they decide to close the aforementioned connection.

However, semaphores also perform a relevant role as an interprocess signaling facility, which can also be a source of deadlocks. In this work, our goal is the study of RAP, so this mode of use is out of scope. We propose fixing deadlock problems due to resource allocation issues firstly, and later apply other techniques for amending those due to message passing.

Due to their versatility, semaphore primitives are interesting for studying how resources can be allocated by a process/thread. For instance, XSI semaphores (also known as System V semaphores) have a multiple wait primitive (`semop` with `sem_op<0`). An example of multiple resource allocation appears in Algorithm 1. Besides, an XSI semaphore can be decremented atomically in more than one unit. Both POSIX semaphores (through `sem_trywait`) and XSI semaphores (through `semop` with `sem_op<0` and `sem_flag=IPC_NOWAIT`) have a non-blocking wait primitive. Again, Algorithm 1 could serve as an example. Finally, XSI semaphores also feature inhibition mechanisms (through `semop` with `sem_op=0`), i.e. processes can wait for a zero value of the semaphore.

As we suggested earlier, the fact that resources in software engineering do not always have a physical counterpart is a peculiar characteristic with consequences. In this context, processes do not only consume resources but also can *create* them. A process will destroy the newly created resources before its termination. For instance, a process can create a shared memory variable (or a service!) which can be allocated to other processes/threads. Hence the resource allocation scheme is no longer *first-acquire-later-release*, but it can be the other way round too. Still, all the resources are used conservatively by the processes (either by a create-destroy sequence or by a wait-release sequence). As a side effect, and perhaps counterintuitively, there may not be free resources during the system startup (as they still must be created), yet the system is live.

Summing up, for successfully modelling RAS in the context of software engineering, a Petri net model should at least fulfill the following requirements:

1. The control flow of the processes should be represented by state machines with support for decisions (`if-then-else` blocks) and nested internal cycles (`while-do` blocks).
2. There can be several resource types and multiple instances of each one.
3. State machines can have multiple tokens (representing concurrent threads).
4. Processes/threads use resources in a conservative way.
5. Acquisition/release arcs can have non-ordinary weights (e.g., a semaphore value can be atomically incremented/decremented in more than one unit).
6. Atomic multiple acquisition/release operations must be allowed.

7. Processes can have decisions dependent of the allocation state of resources (due to the non-blocking wait primitives, as in Fig. 2).
8. Processes can lend resources. As a side effect, there could exist processes that depend on resources which must be created/lent by other processes.

3 PC²R Nets

In this section, we present a new Petri net class, which fulfills the list of abstract requirements enunciated in Sect. 2: the class of Processes Competing for Conservative Resources (PC²R). This class generalizes other subclasses of the S^{n}PR family while respecting the design philosophy on these. Hence, previous results are still valid in the new framework. However, PC²R nets can deal with more complex scenarios which were not yet addressed from the domain of S^{n}PR nets.

It should be remarked that no S^{n}PR net class fulfills all requirements presented at the end of Sect. 2. Although Requirements 2–6 are satisfied by the S^{4}PR class, requirement 1 is only verified by the S*PR class, and fulfillment of Requirement 8 has never been addressed before, justifying the next definition.

Definition 1 presents a subclass of state machines used for modelling the control flow of the processes in isolation. Iterations are allowed, as well as decisions within internal cycles, in such a way that the control flow of structured programs can be fully supported, in fulfillment of Requirement 1.

Definition 1. *An* iterative state machine $\mathcal{N} = \langle \{p_k\} \cup P_1 \cup P_2, T, C \rangle$ *is a strongly connected state machine such that: (i)* P_1, P_2 *and* $\{p_k\}$ *are disjoint sets, (ii)* $P_1 \neq \emptyset$*, (iii) The subnet generated by* $\{p_k\} \cup P_1, {}^\bullet P_1 \cup P_1{}^\bullet$ *is a strongly connected state machine in which every cycle contains* p_k*, and (iv) If* $P_2 \neq \emptyset$*, the subnet generated by* $\{p_k\} \cup P_2, {}^\bullet P_2 \cup P_2{}^\bullet$ *is an iterative state machine.*

As Fig. 3 shows, P_1 contains the set of places of an outermost iteration block, while P_2 is the set of places of the rest of the state machine (the inner structure, which may contain multiple loops within). The place p_0 represents the place "p_k" that we choose after removing every iteration block. In the net of Fig. 1, if we remove the resource places $FORK1$, $FORK2$ and $BOWL$, we obtain an iterative state machine, with $P_1 = \{A2, A3, A4\}$, $P_2 = \{A0, A5, A6\}$ and $p_k = A1$.

PC²R nets are modular models composed by iterative state machines and shared resources. Two PC²R nets can be composed into a new PC²R model via

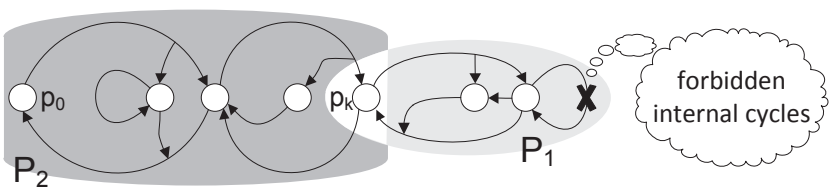

Fig. 3. Schematic diagram of an iterative state machine

fusion of the common resources. Note that a PC^{2}R net can simply be one process modelled by an iterative state machine along with the set of resources it uses.

The class supports iterative processes, multiple resource acquisitions, non-blocking wait operations and resource lending. Inhibition mechanisms are not natively supported (although some cases can still be modelled with PC^{2}R nets).

Definition 2. *Let $I_{\mathcal{N}}$ be a finite set of indices. A PC^{2}R net is a connected generalized self-loop free P/T net $\mathcal{N} = \langle P, T, C \rangle$ where:*

1. *$P = P_0 \cup P_{\mathrm{S}} \cup P_{\mathrm{R}}$ is a partition such that: (a)* [idle places] *$P_0 = \{p_{0_1}, ..., p_{0_{|I_{\mathcal{N}}|}}\}$; (b)* [process places] *$P_{\mathrm{S}} = P_1 \cup ... \cup P_{|I_{\mathcal{N}}|}$, where $\forall i \in I_{\mathcal{N}}$: $P_i \neq \emptyset$ and $\forall i, j \in I_{\mathcal{N}}$: $i \neq j$, $P_i \cap P_j = \emptyset$; (c)* [resource places] *$P_{\mathrm{R}} = \{r_1, ..., r_n\}, n > 0$.*
2. *$T = T_1 \cup ... \cup T_{|I_{\mathcal{N}}|}$, where $\forall i \in I_{\mathcal{N}}, T_i \neq \emptyset$, and $\forall i, j \in I_{\mathcal{N}}, i \neq j, T_i \cap T_j = \emptyset$.*
3. *For all $i \in I_{\mathcal{N}}$ the subnet generated by $\{p_{0_i}\} \cup P_i, T_i$ is an iterative state machine.*
4. *For each $r \in P_R$, there exists a unique minimal p-semiflow associated to r, $Y_r \in \mathbb{N}^{|P|}$, fulfilling: $\{r\} = \|Y_r\| \cap P_{\mathrm{R}}, (P_0 \cup P_{\mathrm{S}}) \cap \|Y_r\| \neq \emptyset$, and $Y_r[r] = 1$.*
5. *$P_{\mathrm{S}} \subseteq \bigcup_{r \in P_{\mathrm{R}}} (\|Y_r\| \setminus \{r\})$.*

In fulfillment of Requirement 8, the support of the Y_r p-semiflows (point 4 of Definition 2) may include P_0: this is new with respect to S^{4}PR nets. Such a resource place r is called a *lender* resource place. If r is a lender, then there exists a process which creates (*lends*) instances of r. As a consequence, there might exist additional minimal p-semiflows containing more than one resource place. This is also new and will be discussed in subsection 4.1.

The next definition generalizes the notion of acceptable initial marking introduced for the S^{4}PR class. In software systems all processes/threads are initially inactive and start from the same point (the `begin` statement). Hence, all of the corresponding tokens are in the idle place at the initial marking (the process places being therefore empty).

Definition 3. *Let $\mathcal{N} = \langle P_0 \cup P_{\mathrm{S}} \cup P_{\mathrm{R}}, T, C \rangle$ be a PC^{2}R. An initial marking m_0 is acceptable for $\mathcal{N}$ iff $||m_0|| \setminus P_{\mathrm{R}} = P_0$, and $\forall p \in P_{\mathrm{S}}, r \in P_{\mathrm{R}} \ : \ Y_r^{\mathrm{T}} \cdot m_0 \geq Y_r[p]$.*

Note that lender resource places may be empty for an acceptable initial marking. Figure 2 shows a PC^{2}R net with an acceptable initial marking which does not belong to the S^{4}PR class.

4 Properties of PC^{2}R Nets: Pitfalls and Fallacies

4.1 Hierarchy of Classes and P-Semiflows

Definition 4. *Previous classes of the S^{n}PR family are defined as follows:*

- *An* S^{5}PR *[16] is a PC^{2}R where $\forall r \in P_{\mathrm{R}}$: $\|Y_r\| \cap P_0 = \emptyset$.*
- *An* S^{4}PR *[22] is an S^{5}PR where $\forall i \in I_{\mathcal{N}}$ the subnet generated by $\{p_{0_i}\} \cup P_i, T_i$ is a strongly connected state machine in which every cycle contains p_{0_i} (i.e., a iterative state machine with no internal cycles).*

- *An* $\mathrm{S^3PR}$ *[4] is an* S^4PR *where* $\forall p \in P_\mathrm{S}$: $|{}^{\bullet\bullet}p \cap P_\mathrm{R}| = 1, ({}^{\bullet\bullet}p \cap P_\mathrm{R} = p^{\bullet\bullet} \cap P_\mathrm{R})$.
- *An* L-$\mathrm{S^3PR}$ *[5] is an* S^3PR *where* $\forall p \in P_\mathrm{S}$: $|{}^{\bullet}p| = |p^{\bullet}| = 1$.

Property 1. *L-*$S^3PR \subseteq S^3PR \subseteq S^4PR \subseteq S^5PR \subseteq PC^2R$.

The preceding property is straightforward from Definition 4. It is worth noting that Definition 3 collapses with the definition of acceptable initial markings respectively provided for those subclasses [4,5,22].

Besides, there exists another class for Sequential RAS, called SPQR [16], which does not strictly contain or is contained by the $\mathrm{PC^2R}$ class. Yet, there exist transformation rules to travel between $\mathrm{PC^2R}$s and Structurally Bounded (SB) SPQRs. Note that, by construction, $\mathrm{PC^2R}$ nets are conservative, and hence SB, but this is not true for SPQRs. The SPQR class is interesting from an analytical point of view thanks to its syntactic simplicity, as discussed in subsection 4.4.

One perspective for inspecting the differences between the subclasses of $\mathrm{PC^2R}$ is that of the form and number of minimal p-semiflows. All subclasses are conservative by definition. Let Y_{S_i} denote the unique minimal p-semiflow induced by the iterative state machine generated by restricting $\mathcal{N}$ to $\langle \{p_{0_i}\} \cup P_i, T_i \rangle$.

Lemma 1. *Let* $\mathcal{N} = \langle P, T, C \rangle$ *be a* PC^2R. *A basis of the left annuller space of the incidence matrix* C *contains* $|P_\mathrm{R}| + |I_\mathcal{N}|$ *vectors.*

Proof. For every $r \in P_\mathrm{R}$, let $Y_r^- = Y_r - \sum_{i \in I_\mathcal{N}} Y_r[p_{0_i}] \cdot Y_{\mathrm{S}_i}$. The set of vectors $A = \{Y_{\mathrm{S}_i} \mid i \in I_\mathcal{N}\} \cup \{Y_r^- \mid r \in P_\mathrm{R}\}$ contains $|P_\mathrm{R}| + |I_\mathcal{N}|$ vectors and they are linearly independent, because for each $p \in P_\mathrm{R} \cup P_0$ there exists one and only one distinct vector $y \in A$ such that $y[p] = 1$ and $\forall y' \in A$, $y' \neq y$, $y'[p] = 0$. Moreover, A is a basis because there is no other vector linearly independent with the vectors of A. We prove by contradiction this last statement. Let us suppose that there exists y being a left annuller of C and y cannot be generated from vectors of A. Construct the vector $y' = y - \sum_{i \in I_\mathcal{N}} y[p_{0_i}] \cdot Y_{\mathrm{S}_i} - \sum_{r \in P_\mathrm{R}} y[r] \cdot Y_r^-$. Obviously, y' is a left annuller of $\mathcal{N}$ because so are the vectors of A and y, but $||y'|| \subseteq P_\mathrm{S}$, and this is not possible because there is no left annullers of the iterative state machines without their idle places. □

Now let B be a matrix of dimensions $(|P_\mathrm{R}| + |I_\mathcal{N}|) \times |P|$ of integers such that the rows of B are the set of vectors A defined in the previous proof.

Lemma 2. *If* $\mathcal{N}$ *is an* S^5PR, B *is a non-negative canonical basis of p-semiflows.*

Proof. By reordering columns in B so that the first ones correspond to $P_\mathrm{R} \cup P_0$, and subsequently reordering the rows of B, we obtain a matrix of the form $[I|B']$, where I is the identity matrix of dimension $|P_\mathrm{R}| + |I_\mathcal{N}|$, and B' is a matrix of dimensions $(|P_\mathrm{R}| + |I_\mathcal{N}|) \times |P_\mathrm{S}|$ of non-negative integers, since in $\mathrm{S^5PR}$ nets, $\forall r \in P_\mathrm{R}$, $Y_r[P_0] = \mathbf{0}$ and $Y_r^- = Y_r$. □

Corollary 1. *If* $\mathcal{N}$ *is an* S^3PR, *then every row in* B *belongs to* $\{0,1\}^{|P|}$.

However, nets belonging to the $\mathrm{S^4PR}$ class may have non-binary minimal p-semiflows. Furthermore, $\mathrm{PC^2R}$ nets feature a new kind of minimal p-semiflows:

Lemma 3. *If $\mathcal{N}$ is a PC^{2}R but not an S^{5}PR, there exists at least one minimal p-semiflow whose support contains more than one resource place.*

Proof. Since $\mathcal{N}$ is not an S^{5}PR, there exists $r \in P_{\mathrm{R}}$, $i \in I_{\mathcal{N}}$ such that $p_{0_i} \in \|Y_r\|$. In that case, there exists at least one $p \in P_{\mathrm{S}_i}$ such that $p \notin ||Y_r||$ (otherwise, $Y_r - Y_{\mathrm{S}_i} \geq \mathbf{0}$ and Y_r is not minimal). Let A be the minimal set of minimal p-semiflows, $\{Y_u \mid u \in P_{\mathrm{R}}\}$, that are essential to cover every $p \in P_{\mathrm{S}_i}$ such that $Y_r[p] = 0$. This means that $Y_u \in A$ iff $\exists p \in P_{\mathrm{S}_i}$, $Y_r[p] = 0$, $Y_u[p] \neq 0$ and $\forall Y_v \in A$, $Y_v \neq Y_u : Y_v[p] = 0$. By Definition 2.5, this set A exists. We construct $Y'_r = Y_r + \sum_{Y_u \in A} Y_u$, that obviously contains, at least, the p-semiflow Y_{S_i}. Therefore, $Y_r^- = Y'_r - k \cdot Y_{\mathrm{S}_i}$, $k = \min_{p \in ||Y_{\mathrm{S}_i}||} \{Y'_r[p]\}$, is a p-semiflow by construction, and $Y_r^-[P_{\mathrm{R}}] = Y'_r[P_{\mathrm{R}}]$. In the same way, we detract other Y_{S_j} that can be contained in Y_r^-.

Y_r^- is minimal. By contradiction. Let us suppose that Y_r^- contains Y_v. The resource v must be either r or a resource for which its minimal p-semiflow Y_v is in A. If v is r then $Y_v[r] = Y_r^-[r]$, and $(Y_r^- - Y_v)[p_{0_i}] < 0$. Therefore, we reach a contradiction. If v holds $Y_v \in A$ then $Y_v[v] = Y_r^-[v]$ because the weight $Y_r^-[u]$ is not modified by the previous operations. And $(Y_r^- - Y_v)[p] < 0$ for some $p \in P_{\mathrm{S}_i}$ for which Y_v becomes essential in the set A, since $Y_r^-[p] \leq Y_v[p] - k$. □

In other words, the above result reveals that the set of minimal p-semiflows contains strictly a basis of the left annuller space of the incidence matrix C.

4.2 Liveness Characterization and Siphons

Traditionally, empty or insufficiently marked siphons have been a fruitful structural element for characterizing non-live RAS. The more general the net class, however, the more complex the siphon-based characterization is. The following results can be easily obtained from previously published works. The originality here is to point out the strict conditions that the siphons must fulfill.

Theorem 1. *Let $\langle \mathcal{N}, m_0 \rangle$ be a marked S^{3}PR with an acceptable initial marking. $\langle \mathcal{N}, m_0 \rangle$ is non-live iff $\exists m \in RS(\mathcal{N}, m_0)$ and a <u>minimal</u> siphon D: $m[D] = \mathbf{0}$.*

Proof. In [4] it is proved that $\langle \mathcal{N}, m_0 \rangle$ is non-live iff $\exists m \in RS(\mathcal{N}, m_0)$ and an empty siphon D at m, i.e. $m[D] = \mathbf{0}$. Hence, the sufficient part is straightforward. Now suppose that the empty siphon D is not minimal. Then there must exist a minimal siphon $D' \subset D$. Since $m[D'] = \mathbf{0}$, an empty minimal siphon exists. □

For instance, the marked S^{3}PR in Fig. 4 is non-live with $K_0 = K_1 = 1$, $K_3 = 2$. From this acceptable initial marking, the marking $(A4 + B4 + R2 + 2 \cdot R3)$ can be reached by firing σ, where $\sigma = \langle TB1, TA1, TB2, TA2, TB3, TA3, TB4, TA4 \rangle$. This firing sequence empties the siphon $\{A1, B1, A5, B5, R1, R4\}$.

However, this characterization is sufficient, but not necessary, in general, for S^{4}PR nets. Hence, the concept of *empty siphon* had to be generalized. Given a marking m in an S^{4}PR net, a transition t is said to be m-process-enabled (m-process-disabled) iff it has (not) one marked input process place, and m-resource-enabled (m-resource-disabled) iff its input resource places have (not) enough tokens to fire it, i.e., $m[P_{\mathrm{R}}, t] \geq Pre[P_{\mathrm{R}}, t]$ ($m[P_{\mathrm{R}}, t] \ngeq Pre[P_{\mathrm{R}}, t]$).

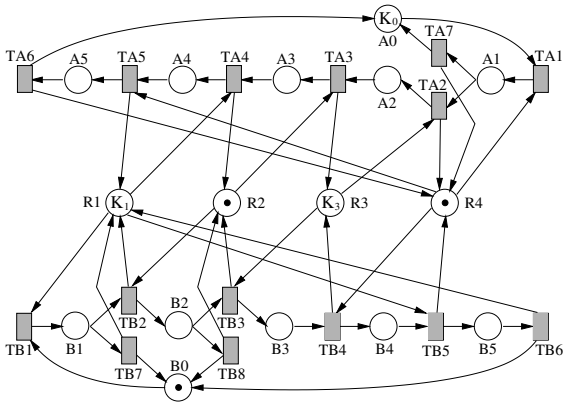

Fig. 4. An S^{3}PR which is non-live iff $(K_0 \geq K_1,\ K_3 \geq 2) \vee (K_0 \cdot K_1 \cdot K_3 = 0)$

Theorem 2. *[22] Let $\langle \mathcal{N}, m_0 \rangle$ be a marked S^4PR with an acceptable initial marking. $\langle \mathcal{N}, m_0 \rangle$ is non-live iff $\exists m \in RS(\mathcal{N}, m_0)$ and a siphon D such that:* i) *There exists at least one m-process-enabled transition;* ii) *Every m-process-enabled transition is m-resource-disabled by resource places in D;* iii) *Process places in D are empty at m.*

Such a siphon D is said to be insufficiently marked at m. In Theorems 1 and 2, the siphon captures the concept of circular wait, revealing it from the underlying net structure. In contrast to the S^{3}PR class, it is worth noting the following fact about *minimal* siphons in S^{4}PR nets, which emerges because of their minimal p-semiflows not being strictly binary.

Property 2. *There exist non-live S^4PR nets with an acceptable initial marking for which every siphon characterizing the non-liveness is non-minimal, i.e. minimal siphons are insufficient to characterize non-liveness.*

Proof. The net in Fig. 5 is non-live, but there is no minimal siphon containing both resource places $R1$ and $R2$. Note that the siphon $D = \{R1, R2, A3, B2\}$ becomes insufficiently marked at m, where $m = A1 + B1 + R1 + R2$, but it contains the minimal siphon $D' = \{R2, A3, B2\}$. D' is not insufficiently marked for any reachable marking. It is also worth noting that no siphon is ever emptied. □

Thus we must take care of non-minimal siphons for dealing with deadlocks in systems more complex than S^{3}PR. On the other hand, we have noticed that insufficiently marked siphons (even considering those non-minimal) are not enough for characterizing liveness for more complex systems such as S^{5}PR models. This means that siphon-based control techniques for RAS do not work in general for concurrent software, even in the 'good' case in which every `wait`-like operation precedes its complementary `signal`-like operation.

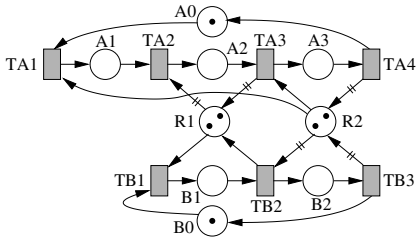

Fig. 5. A non-live S^{4}PR with no minimal siphon becoming insufficiently marked

Property 3. *There exists an S^5PR with an acceptable initial marking $\langle \mathcal{N}, m_0 \rangle$ which is non-live but insufficiently marked siphons do not characterize non-liveness (dead markings).*

Proof. The net in Fig. 6 models a special case of Example 1 with only two philosophers. This system is non-live, while there exist three bad siphons, which are $D_1 = \{A2, A3, A4, A5, A6, B2, B4, B5, B6, FORK2, BOWL\}$, $D_2 = \{A2, A4, A5, A6, B2, B3, B4, B5, B6, FORK1, BOWL\}$ and $D_3 = \{A2, A4, A5, A6, B2, B4, B5, B6, FORK1, FORK2, BOWL\}$. Besides, every transition in the set $\Omega = \{TA2, TA3, TA4, TA5, TB2, TB3, TB4, TB5\}$ is an output transition of D_1, D_2 and D_3. After firing $m_0[TA1\ TB1\rangle$, the state $A1+B1+BOWL$ is reached. This marking belongs to a livelock with other six markings. The reader can check that there exists a firable transition in Ω for every marking in the livelock, and in any case there is no insufficiently marked siphons. □

Revisiting Example 1 and its associated Algorithm 1, it is not difficult to see that, if every philosopher enters the room, sits down and picks up the fork on the left of himself, the philosophers will be trapped in a livelock. Every philosopher can eventually take the bowl of spaghetti and heat it up in the microwave. This pattern can be repeated infinitely often, but it is completely useless, since no philosopher will ever be able to have dinner.

4.3 Deadlock-Freeness, Liveness, Reversibility and Livelocks

In general, livelocks with dead transitions are not a new phenomenon in the context of Petri net models for RAS. Figure 7 shows that, even for L-S^{3}PR nets, deadlock freeness does not imply liveness.

Property 4. *There exists a marked L-S^3PR with an acceptable initial marking such that it is deadlock-free but not live.*

This net system has no deadlock but two reachable livelocks: (i) $\{(A0+B2+C0+D1+R1), (A1+B2+C0+D1)\}$, and (ii) $\{(A0+B1+C0+D2+R3), (A0+B1+C1+D2)\}$. Nevertheless, these livelocks are captured by insufficiently marked siphons. Unfortunately, this no longer holds for some kind of livelocks in S^{5}PR or more complex systems. Indeed, PC^{2}R nets feature some complex properties which complicate the finding of a liveness characterization.

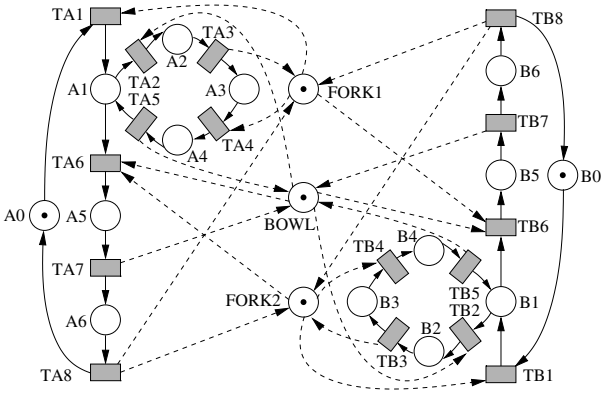

Fig. 6. Two postmodern dining philosophers

Property 5. *There exists an S^3PR such that liveness is not monotonic, neither with respect to the marking of the idle/process places, nor that of the resource places, i.e., liveness is not always preserved when those are increased.*

Proof. With respect to P_{R}: The system in Fig. 4 is live with $K_0 = K_1 = K_3 = 1$ and non-live with $K_0 = K_1 = 1$, $K_3 = 2$ (however, it becomes live again if we increase enough the marking of $R1$, $R2$ and $R4$; so as to make every resource place an implicit place).

With respect to P_0: The system in Fig. 4 is live with $K_0 = 1$, $K_1 = K_3 = 2$ and non-live with $K_0 = K_1 = K_3 = 2$. □

Note that liveness is monotonic for every net belonging to the L-S^3PR class [9] with respect to the resource places. But, from S^3PR nets upwards, there is a discontinuity zone between the point where the resource places are empty enough so that every transition is dead (also held for lower markings), and the point where every resource place is implicit (liveness is preserved if their marking is increased). Markings within these bounds fluctuate between liveness and non-liveness. The location of those points also depends on the marking of the idle/process places: the more tokens in them, the farther the saturation point.

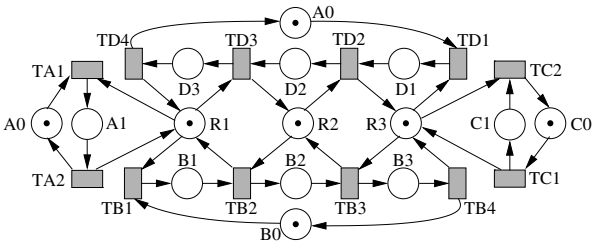

Fig. 7. A non-live L-S^3PR which is deadlock-free

Nevertheless, an interesting property of S^{4}PR nets is that liveness equals reversibility. This, along with the fact that the idle place does not belong to any p-semiflow Y_r, is a powerful feature. If every token in a process net can be moved to the idle place, then the net is not dead (yet).

Theorem 3. *Let $\langle \mathcal{N}, m_0 \rangle$ be an S^4PR with an acceptable initial marking. $\langle \mathcal{N}, m_0 \rangle$ is live iff m_0 is a home state (i.e., the system is reversible).*

Proof. $\Rightarrow$) Let us suppose that m_0 is not a home state, i.e. $\exists m' \in RS(\mathcal{N}, m_0)$ such that $m_0 \notin RS(\mathcal{N}, m')$. Let $m \in RS(\mathcal{N}, m')$ obtained by moving forward all the active processes (firing transitions $T \setminus P_0^\bullet$) until no process enabled transition can be fired. Since $m_0 \notin RS(\mathcal{N}, m')$, $m \neq m_0$, and the set of m-process-enabled transitions is non-empty, and each one of these transitions is m-resource-disabled. Hence, by Theorem 2, $\langle \mathcal{N}, m_0 \rangle$ is non-live.

$\Leftarrow$) Let m be a reachable marking from m_0, $m \in RS(\mathcal{N}, m_0)$, and t a transition of the net. We prove that there exists a successor marking of m, m', from which t is firable. Since m_0 is a home state, there exists a firable sequence from m such that $m[\sigma\rangle m_0$. Taking into account that for acceptable initial markings in S^{4}PR, every t-semiflow can be fired in isolation from the initial marking [21], then there exists a sequence X whose characteristic vector $\overline{X}$ is equal to the t-semiflow containing t (this t-semiflow exists because the net is consistent). Let $X = \sigma' t \sigma''$ be a decomposition of the firing sequence X to point out the first firing of t. Therefore $\sigma\sigma'$ is firable leading to a marking m' from which t is firable. Because m and t have been selected without constraints, the net is live. □

However, Theorem 3 is false in general for S^{5}PR nets. In fact, the directedness property [1] does not even hold. This implies that an S^{5}PR may not have a home state, even being live.

Property 6. *There exists an S^5PR with an acceptable initial marking $\langle \mathcal{N}, m_0 \rangle$ such that the system is live but there is no home state.*

Proof. The net system in Fig. 8 has no home state in spite of being live. Due to the large size of the net, complete state space enumeration has been omitted, but a sketch of its reachability graph is included. □

Having said that, S^{5}PR nets still retain an interesting property: its minimal t-semiflows are eventually realizable from an acceptable initial marking.

Theorem 4. *Let $\langle \mathcal{N}, m_0 \rangle$ be an S^5PR with an acceptable initial marking. For every (minimal) t-semiflow x, there exists a reachable marking $m \in RS(\mathcal{N}, m_0)$ such that x is realizable from m, i.e. $\exists \sigma$ such that $m[\sigma\rangle$, $\overline{\sigma} = x$.*

Proof. This can be easily proven from Definitions 2 and 3. As in S^{4}PR nets [21], an acceptable initial marking m_0 guarantees the executability from m_0 of every minimal t-semiflow (elementary circuit) whose corresponding t-component contains an idle place. Once the token is carried to the entry place p of an internal cycle, the number of available resources guarantees the executability of every minimal t-semiflow such that its t-component contains p. Following an inductive reasoning, every (minimal) t-semiflow is eventually realizable from m_0. □

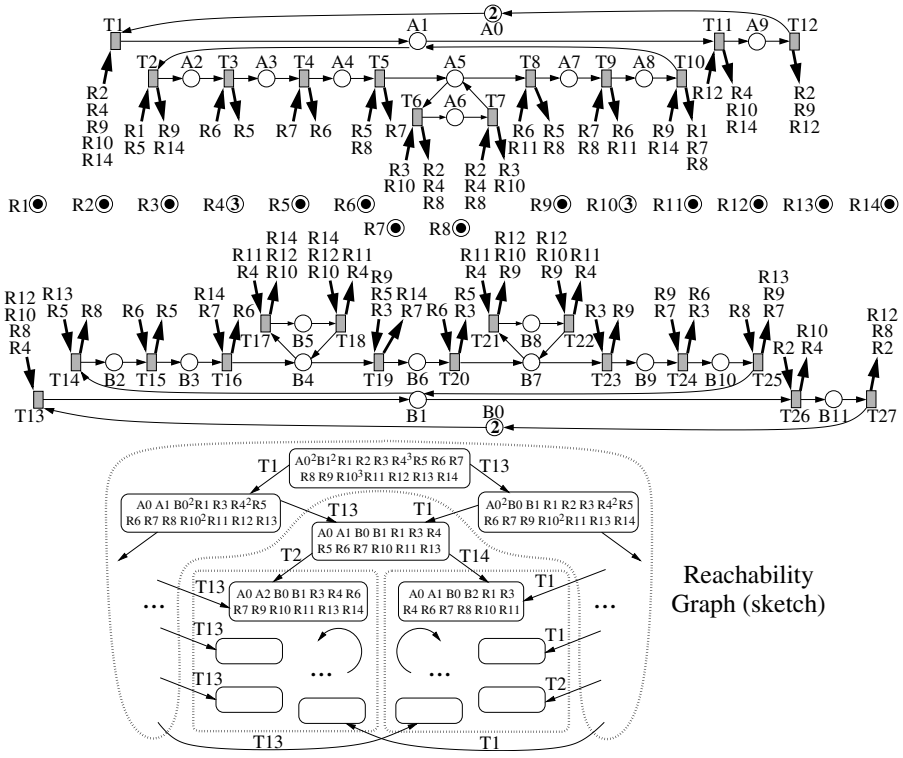

Fig. 8. A live S^5PR which has no home state. The arcs from/to P_R are omitted for clarity. Instead, the set of input and output resource places are listed next to each transition. The net is ordinary. Below, a sketch of its reachability graph.

However, for PC^2R nets there may not exist minimal t-semiflows being eventually realizable; even for live systems.

Property 7. *There exists a PC^2R with an acceptable initial marking $\langle \mathcal{N}, m_0 \rangle$ such that the system is live and there exists a minimal t-semiflow x such that $\forall m \in RS(\mathcal{N}, m_0)$, $\nexists \sigma$ such that $m[\sigma\rangle$ and $\overline{\sigma} = x$, i.e. x is realizable from m.*

Proof. The reader can check that the net system in Fig. 9 has no home state in spite of being live. Depending on which transition we fire first (either $T1$ or $T8$) we fall in a different livelock. Besides, for every reachable marking, there is no minimal t-semiflow such that it is realizable, i.e. firable in isolation. Instead, both state machines need each other to progress from the very beginning. Again, state space enumeration has been omitted for the sake of concision. □

4.4 Relation with the SPQR Class

Finally, we feel it is worth bringing to attention that in [16] we introduced a new class of Petri net models for RAS, called SPQR (Systems of Processes

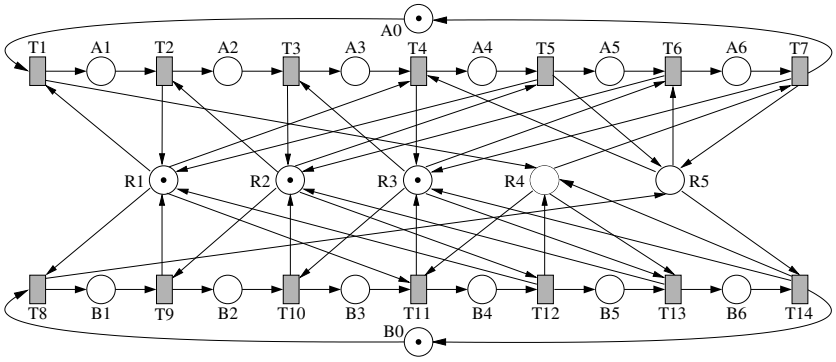

Fig. 9. A marked PC^{2}R for which no minimal t-semiflow is ever realizable

Quarreling over Resources). SPQR nets feature an appealing syntactic simplicity and expressive power though they are very challenging from an analytical point of view. They can be roughly described as RAS nets in which the process subnets are acyclic and the processes can lend resources in any possible (conservative) manner. Every PC^{2}R can be transformed into a Structurally Bounded SPQR net (SB SPQR net). Note that SB SPQRs are conservative since they are consistent by construction, and consistency plus structurally boundedness implies conservativeness [20].

We believe that the transformation of PC^{2}R nets into SB SPQR can be useful to understand the above phenomena from a structural point of view. Intuitively speaking, the concept of *lender resource* seems a simple yet powerful instrument which still remains to be fully explored. Still, SB SPQRs can present very complex behaviour [16].

The transformation rule is based on the idea of converting every `while-do` block into an acyclic process which is activated by a lender resource place. This lender place gets marked once the thread reaches the while-do block. The token is removed at the exit of the iteration. This transformation must be applied from the innermost loops outwards. Figure 10 depicts the transformation rule. The rule preserves the language accepted by the net (and thus liveness) since it

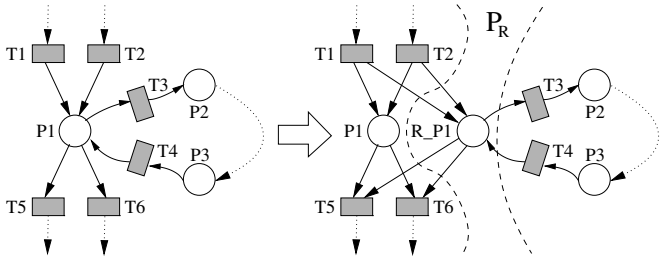

Fig. 10. Transforming PC^{2}Rs into SB SPQRs: From iterative to acyclic processes

basically consists in the addition of a implicit place (place $P1$ in the right hand net of Fig. 10, since R_P1 can be seen as a renaming of $P1$ in the left hand net).

Figure 11 illustrates the transformation of the PC^{2}R in Fig. 6 into the corresponding SB SPQR.

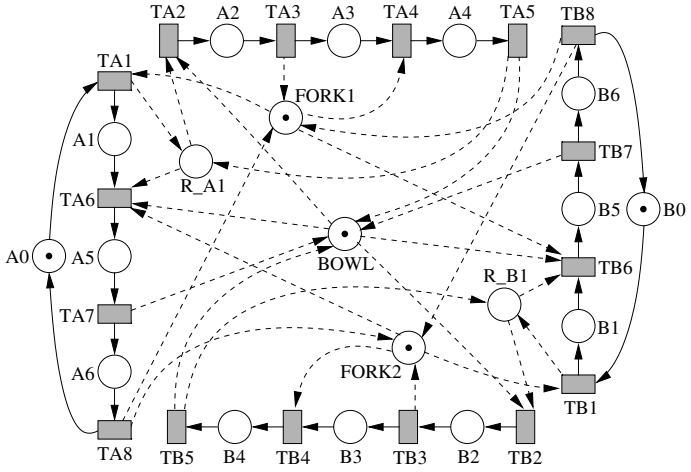

Fig. 11. From PC^{2}R to SB SPQR: Two postmodern dining philosophers

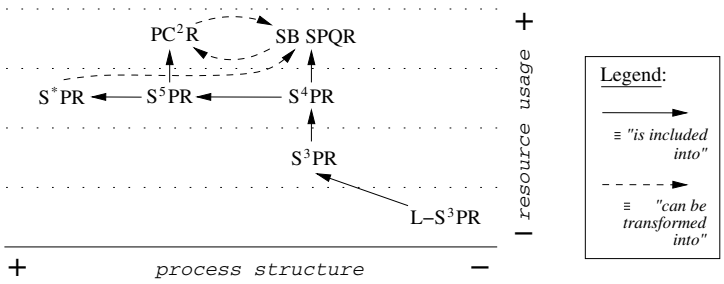

Fig. 12. Inclusion relations between Petri net classes for RAS

Out of curiosity, the concept of *acceptable initial marking* has not been defined for the SPQR class, since it was conceived as an instrumental model class with no physical meaning. On the other hand, the concept of acceptable initial marking does have a physical meaning in the context of PC^{2}R nets modelling real-world systems in software engineering. As a lateral effect, the transformation of a SB SPQR $\langle \mathcal{N}, m_0 \rangle$ into a marked PC^{2}R $\langle \mathcal{N}', m_0' \rangle$ may not produce a net system with an acceptable initial marking m_0', even when $m_0[P_S] = \mathbf{0}$.

Figure 12 introduces the inclusion relations between a variety of Petri net classes for Sequential RAS.

5 Conclusion and Future Work

Although there exist a variety of Petri net classes for RAS, many of these definition efforts have been directed to obtain powerful theoretical results for the analysis and synthesis of this kind of systems. Nevertheless, we believe that the process of abstraction is a central issue in order to have useful models from a real-world point of view, and therefore requires careful attention. In this work, we have followed that path and constructed a requirements list for obtaining an interesting Petri net subclass of RAS models applied to the software engineering domain. Considering that list, we defined the class of $\mathrm{PC}^2\mathrm{R}$ nets, which fulfills those requirements while respecting the design philosophy on the RAS view of systems. We also introduced some useful transformation and class relations so as to locate the new class among the myriad of previous models. Finally we proved that the problem of liveness in the new context is non-trivial and presented some cases of bad behaviour which will be subject of subsequent work.

A Petri Nets: Basic Definitions

A *place/transition net* (P/T net) is a 3-tuple $\mathcal{N} = \langle P, T, W \rangle$, where W is a total function $W : (P \times T) \cup (T \times P) \to \mathbb{N}$, being P, T non empty, finite and disjoint sets. Elements belonging to the sets P and T are called respectively *places* and *transitions*.

The *preset* (*poset*) or set of input (output) nodes of a node $x \in P \cup T$ is denoted by ${}^\bullet x$ ($x^\bullet$), where ${}^\bullet x = \{y \in P \cup T \mid W(y,x) \neq 0\}$ ($x^\bullet = \{y \in P \cup T \mid W(x,y) \neq 0\}$). The preset (poset) of a set of nodes $X \subseteq P \cup T$ is denoted by ${}^\bullet X$ ($X^\bullet$), where ${}^\bullet X = \{y \mid y \in {}^\bullet x, x \in X\}$ ($X^\bullet = \{y \mid y \in x^\bullet, x \in X\}$

An *ordinary P/T net* is a net with unitary arc weights (i.e., $W : (P \times T) \cup (T \times P) \to \{0,1\}$). Otherwise, the P/T net is called *generalized*. A *state machine* is an ordinary net such that for every $t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$. An *acyclic state machine* is an ordinary net such that for every $t \in T$, $|{}^\bullet t|, |t^\bullet| \leq 1$, and there is no circuit in it.

A self-loop place $p \in P$ is a place such that $p \in p^{\bullet\bullet}$. A *self-loop free P/T net* is a net with no self-loop places that can be also defined by the 3-tuple $\mathcal{N} = \langle P, T, C \rangle$, where C is called the *incidence matrix*, $C[p,t] = W(p,t) - W(t,p)$.

A *p-flow* is a vector $Y \in \mathbb{Z}^{|P|}$, $Y \neq \mathbf{0}$, which is a left annuler of the incidence matrix, $Y \cdot C = \mathbf{0}$. The support of a p-flow is denoted $\|Y\|$. A *p-semiflow* is a non-negative p-flow. The P/T net $\mathcal{N}$ is *conservative* iff every place is covered by a p-semiflow. A *minimal p-semiflow* is a p-semiflow such that the g.c.d of its non-null components is one and its support $\|Y\|$ is not an strict superset of the support of another p-semiflow.

A set of places $D \subseteq P$ is a *siphon* iff every place $p \in {}^\bullet D$ satisfies $p \in D^\bullet$. The support of a p-semiflow is a siphon but the opposite does not hold in general.

Let $\mathcal{N} = \langle P, T, W \rangle$ be a P/T net, and let $P' \subseteq P$ and $T' \subseteq T$, where $P', T' \neq \emptyset$. The P/T net $\mathcal{N}' = \langle P', T', W' \rangle$ is the subnet generated by P', T' iff $W'(x,y) \Leftrightarrow W(x,y)$, for every pair of nodes $x, y \in P' \cup T'$.

A *marking* m of a P/T net $\mathcal{N}$ is a vector $\mathbb{N}^{|P|}$, assigning a finite number of *tokens* $m[p]$ to every place $p \in P$. The *support* of a marking, $\|m\|$, is the set of places which are marked in m, i.e. $\|m\| = \{p \in P \mid m[p] \neq 0\}$. We define a *marked P/T net* (also P/T net system) as the pair $\langle \mathcal{N}, m_0 \rangle$, where $\mathcal{N}$ is a P/T net, and m_0 is a marking for $\mathcal{N}$, also called *initial marking*. $\mathcal{N}$ is said to be the structure of the system, while m_0 represents the system state.

Let $\langle \mathcal{N}, m_0 \rangle$ be a marked P/T net. A transition $t \in T$ is *enabled* (also *firable*) iff $\forall p \in {}^\bullet t \ : \ m_0[p] \geq W(p,t)$, which is denoted by $m_0[t\rangle$. The *firing* of an enabled transition $t \in T$ changes the system state to $\langle \mathcal{N}, m_1 \rangle$, where $\forall p \in P \ : \ m_1[p] = m_0[p] + C[p,t]$, and is denoted by $m_0[t\rangle m_1$. A *firing sequence* σ from $\langle \mathcal{N}, m_0 \rangle$ is a non-empty sequence of transitions $\sigma = t_1\, t_2 \ldots t_k$ such that $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{k-1}[t_k\rangle m_k$. The firing of σ is denoted by $m_0[\sigma\rangle m_k$. The *language* of $\langle \mathcal{N}, m_0 \rangle$, $\mathcal{L}(\mathcal{N}, m_0)$, is the set of firing sequences from $\langle \mathcal{N}, m_0 \rangle$. A marking m is *reachable* from $\langle \mathcal{N}, m_0 \rangle$ iff there exists a firing sequence σ such that $m_0[\sigma\rangle m$. The *reachability set* $RS(\mathcal{N}, m_0)$ is the set of reachable markings, i.e. $RS(\mathcal{N}, m_0) = \{m \mid \exists\ \sigma \ : \ m_0[\sigma\rangle m\}$. A place $p \in P$ is a *sequential implicit place* in $\langle \mathcal{N}, m_0 \rangle$ (or, simply, *implicit*) iff $\mathcal{L}(\mathcal{N}, m_0) = \mathcal{L}(\mathcal{N}', m_0')$, where $\langle \mathcal{N}', m_0' \rangle$ is the net system resulting from removing the place p from the original net.

A transition $t \in T$ is *live* iff for every reachable marking $m \in RS(\mathcal{N}, m_0)$, $\exists m' \in RS(\mathcal{N}, m)$ such that $m'[t\rangle$. The system $\langle \mathcal{N}, m_0 \rangle$ is *live* iff every transition is live. Otherwise, $\langle \mathcal{N}, m_0 \rangle$ is *non-live*. A transition $t \in T$ is *dead* iff there is no reachable marking $m \in RS(\mathcal{N}, m_0)$ such that $m[t\rangle$. The system $\langle \mathcal{N}, m_0 \rangle$ is a *total deadlock* iff every transition is dead, i.e. no transition is firable. A *home state* m_k is a marking such that it is reachable from every reachable marking, i.e. $\forall m \in RS(\mathcal{N}, m_0) \ : \ m_k \in RS(\mathcal{N}, m)$. The net system $\langle \mathcal{N}, m_0 \rangle$ is *reversible* iff m_0 is a home state. A non-empty set of markings is a *livelock* of $\langle \mathcal{N}, m_0 \rangle$ iff for every $m \in M : RS(\mathcal{N}, m) = M$ and $m_0 \notin M$.

References

1. Best, E., Voss, K.: Free Choice Systems Have Home States. Acta Informatica 21, 89–100 (1984)
2. Coffman, E.-G., Elphick, M., Shoshani, A.: System Deadlocks. ACM Computing Surveys 3(2), 67–78 (1971)
3. Colom, J.-M.: The Resource Allocation Problem in Flexible Manufacturing Systems. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 23–35. Springer, Heidelberg (2003)
4. Ezpeleta, J., Colom, J.M., Martínez, J.: A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. IEEE Transactions on Robotics and Automation 11(2), 173–184 (1995)
5. Ezpeleta, J., García-Valles, F., Colom, J.M.: A Class of Well Structured Petri Nets for Flexible Manufacturing Systems. In: Desel, J., Silva, M. (eds.) ICATPN 1998. LNCS, vol. 1420, pp. 64–83. Springer, Heidelberg (1998)
6. Ezpeleta, J., Recalde, L.: A Deadlock Avoidance Approach for Non-Sequential Resource Allocation Systems. IEEE Transactions on Systems, Man and Cybernetics. Part–A: Systems and Humans 34(1) (2004)

7. Ezpeleta, J., Tricas, F., García-Vallés, F., Colom, J.M.: A Banker's Solution for Deadlock Avoidance in FMS with Flexible Routing and Multiresource States. IEEE Transactions on Robotics and Automation 18(4), 621–625 (2002)
8. Fanti, M.P., Maione, B., Mascolo, S., Turchiano, B.: Event-based Feedback Control for Deadlock Avoidance in Flexible Production Systems. IEEE Transactions on Robotics and Automation 13(3), 347–363 (1997)
9. García-Vallés, F.: Contributions to the Structural and Symbolic Analysis of Place/Transition Nets with Applications to Flexible Manufacturing Systems and Asynchronous Circuits. Ph.D. thesis. University of Zaragoza, Zaragoza (April 1999)
10. Harel, D.: On Folk Theorems. Communications of the ACM 23(7), 379–389 (1980)
11. Hoare, C.A.R.: Communicating Sequential Processes. Communications of the ACM 21(8), 666–677 (1978)
12. Hu, H.S., Zhou, M.C., Li, Z.W.: Liveness Enforcing Supervision of Video Streaming Systems Using Non-Sequential Petri Nets. IEEE Transactions on Multimedia 11(8), 1446–1456 (2009)
13. Jeng, M.D., Xie, X.L., Peng, M.Y.: Process Nets with Resources for Manufacturing Modeling and their Analysis. IEEE Transactions on Robotics 18(6), 875–889 (2002)
14. Lautenbach, K., Thiagarajan, P.S.: Analysis of a Resource Allocation Problem Using Petri Nets. In: Syre, J.C. (ed.) Proc. of the 1st European Conf. on Parallel and Distributed Processing, Cepadues Editions, Toulouse, pp. 260–266 (1979)
15. Li, Z.W., Zhou, M.C.: Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach. Springer, New York (2009)
16. López-Grao, J.P., Colom, J.M.: Lender Processes Competing for Shared Resources: Beyond the S^4PR Paradigm. In: Proc. of the 2006 Int. Conf. on Systems, Man and Cybernetics, pp. 3052–3059. IEEE (2006)
17. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)
18. Park, J., Reveliotis, S.A.: Deadlock Avoidance in Sequential Resource Allocation Systems with Multiple Resource Acquisitions and Flexible Routings. IEEE Transactions on Automatic Control 46(10), 1572–1583 (2001)
19. Reveliotis, S.A., Lawley, M.A., Ferreira, P.M.: Polynomial Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems. IEEE Transactions on Automatic Control 42(10), 1344–1357 (1997)
20. Silva, M.: Introducing Petri Nets. In: Di Cesare, F., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, F. (eds.) Practice of Petri Nets in Manufacturing, pp. 1–62. Chapman and Hall (1993)
21. Tricas, F.: Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems. Ph.D. thesis. University of Zaragoza, Zaragoza (May 2003)
22. Tricas, F., García-Valles, F., Colom, J.M., Ezpeleta, J.: A Petri Net Structure-Based Deadlock Prevention Solution for Sequential Resource Allocation Systems. In: Proc. of the 2005 Int. Conf. on Robotics and Automation (ICRA), pp. 272–278. IEEE, Barcelona (2005)
23. Xie, X., Jeng, M.D.: ERCN-Merged Nets and their Analysis Using Siphons. IEEE Transactions on Robotics and Automation 29(4), 692–703 (1999)