Constraint-Flow Nets: A Model for Building Constraints from Resource Dependencies

Simon Bliudze, Alena Simalatsar, Alina Zolotukhina

▶ **To cite this version:**

Simon Bliudze, Alena Simalatsar, Alina Zolotukhina. Constraint-Flow Nets: A Model for Building Constraints from Resource Dependencies. 19th International Conference on Coordination Languages and Models (COORDINATION), Jun 2017, Neuchâtel, Switzerland. pp.197-216, 10.1007/978-3-319-59746-1_11 . hal-01657340

HAL Id: hal-01657340

https://inria.hal.science/hal-01657340v1

Submitted on 6 Dec 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Constraint-Flow Nets: A Model for Building Constraints from Resource Dependencies

Simon Bliudze[1], Alena Simalatsar[2], and Alina Zolotukhina[1]

[1] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
firstname.lastname@epfl.ch
[2] University of Applied Sciences Western Switzerland, Sion, Switzerland
alena.simalatsar@hevs.ch

Abstract. The major research in the resource management literature focuses primarily on two complementary sub-problems: 1) specification languages for formulating resource requests and 2) constraint problems modelling allocation and scheduling. Both directions assume the knowledge of the underlying platform architecture and the dependencies it induces on the usage of the various resources. In this paper, we bridge this gap by introducing constraint-flow nets (cfNets). A cfNet is defined by a set of resources and dependencies between them, each dependency having an associated constraint schema. The model is inspired by Petri nets, with resources corresponding to places and dependencies—to transitions. Given an architecture of dependent resources, an initial resource request is propagated through the dependencies. The generated constraints are then conjuncted into the global allocation constraint. We study the notion of conflicts in cfNets and prove that for *conflict-free* cfNets the global allocation constraint can be constructed unambiguously. Furthermore, we provide an SMT-based algorithm for conflict detection and discuss the use of priorities to dynamically resolve conflicts at run-time. Finally, we illustrate the use of cfNets on a case study inspired by the Kalray MPPA architecture.

Keywords: Resource management • Resource dependencies • Constraint-flow nets • Petri nets • Marking reachability • Conflict detection

1 Introduction

Providing resource management is of key importance to many different areas, from embedded systems domain to distributed resource management in large-scale systems or in a cloud.

In the literature, two main complementary sub-problems are investigated: specification languages for formulating resource requests [8,13,20,31] and resource management architectures [10,11,17,19,24]. The former provides application developers with the means to specify application resource requirements, whereas the latter is using the request information to build a constraint problem, which is then solved by a satisfiability modulo theories (SMT) [2,26] or a constraint solver [29] to find a satisfactory resource allocation. However, for

non-trivial architectures, this approach presents a substantial gap. Indeed, on one hand, the resource manager assumes that an application completely specifies all its resource requirements. On the other hand, specification languages provide request primitives formulated in terms of ⟨required amount/resource type⟩ pairs, e.g. "5Mb of memory" or "1 thread". Ignoring the physical nature of the resources and the dependencies among them makes it impossible for applications to define sufficiently complete resource requests. Furthermore, we argue that such completeness is not desirable. In order to avoid strong platform dependencies, applications should have the possibility to operate on a more abstract level. For a simple example, consider a multicore Network-on-Chip (NoC) platform (e.g. [16]), where each core has a dedicated local memory, but can also access that of the other cores through the NoC. Depending on the location of the requested memory and under the assumptions above, application developers must also explicitly request access to the NoC. Another example is provided by modular platforms, where resources, such as memory, channels or threads, can be created dynamically: applications should be allowed to specify requests for a certain type of resources without having the knowledge of their structure. While some advanced compilation tools, e.g. [15,26,27] provide ad-hoc solutions for specific target platforms, the objective of the work presented in this paper is to bridge this gap in a generic manner, sufficient to describe resource dependencies for a wide class of platforms.

We consider an environment with a global set of resources $\mathcal{R}$ and an entity (application) that makes a request for a subset of these resources. In general, the information contained in the request is not sufficient to find a satisfactory resource allocation, due to potential dependencies among the resources (in the above example, remote memory access requires the use of the NoC). To model such dependencies we introduce the notion of *Constraint-Flow Nets* (cfNets), inspired by Petri nets with inhibitor arcs. Inhibitor arcs are used to limit dependency applications (e.g. there is no need to repeat a request for a given resource, if it has already been requested). In order to specify relations between the amounts of the resources requested by the application and the necessary amounts of the resources introduced by dependencies, we associate *constraint schemata* to all transitions of a cfNet. These constraint schemata are then used to build the global constraint problem associated to the initial resource request. We prove that such global constraint problems can be unambiguously built for *conflict-free* cfNets. Furthermore, we provide a technique for detection of conflicts and their resolution by introducing priority relations among the conflicting transitions. Hence, given a cfNet with a *priority model*, the global constraint can always be built unambiguously.

The paper is structured as follows. Section 2 presents the motivating example that we use to illustrate our theory throughout the paper. Section 3 introduces cfNets and their semantics in terms of the process leading to constraint problems corresponding to resource requests. Section 4 focuses on the notion of conflict in cfNets, providing an algorithm for detecting conflicts and introducing priorities to resolve them. Section 5 provides a complete cfNet modelling the Kalray archi-

tecture described in Sect. 2. Section 6 provides a short overview of related work. Section 7 concludes the paper and discusses some future research directions.

Additional material and proofs of all results in this paper are provided as a technical report available online [7].

2 Motivating example

Our running example is inspired by the many-core architecture of Kalray MPPA-256 [16], which consists of 256 processing elements (PE), i.e. cores, grouped into compute clusters of 16 cores each. Within a cluster, cores communicate through a shared memory, which consists of 16 independent memory banks grouped into two sides: *left* and *right*. In this paper, we will consider a simplified Kalray cluster composed of four cores and four memory banks as shown in Fig. 1.

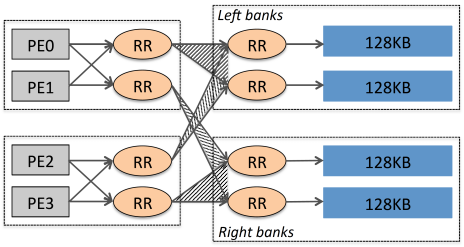


Fig. 1: Compute cluster reference architecture with arbitration points

Two cores cannot access the same memory bank at the same time. Cores are organised in pairs. Each pair shares two data-buses: one for each of the memory sides [12]. Therefore, the access to memory banks is arbitrated by two stages of arbiters implementing the Round Robin (RR) arbitration policy. Our goal in this paper will be to allocate cores, buses and memory banks, such that there will be at most one request for any arbiter queue, making the resource unavailable otherwise. Thus, we assume that two cores of one pair can access different memory sides simultaneously and two cores from different pairs may access different memory banks of the same side.

3 Modelling resource dependencies

3.1 Flow nets

In this section, we introduce fNets, which we use to model resource dependencies. Syntactically, fNets are Petri nets with inhibitor arcs. The semantics of fNets can be compared to that of Coloured Petri nets with inhibitor arcs and capacities (each place has capacity 1 with respect to each token colour). The colour of a token in an fNet depends on the transition that has produced this token. The main difference between fNets and Petri nets is the following: *firing a transition does not remove tokens from its pre-places*. Therefore, the capacity restriction effectively prevents any transition from being fired more than once.

Definition 1. *Consider a tuple* $\mathcal{N} = (\mathcal{R}, T, F, I)$*, where* $\mathcal{R}$ *is a finite set of* places *(resources);* T *is a finite set of* transitions *(dependencies), such that* $\mathcal{R}$

and T are disjoint; $F \subseteq (\mathcal{R} \times T) \cup (T \times \mathcal{R})$ is a set of arcs *and $I \subseteq \mathcal{R} \times T^a \times T$, with $T^a \stackrel{def}{=} T \cup \{*\}$, for some fresh symbol $* \notin T$, is a set of* inhibitor arcs.

For $t \in T$, we denote by $R^-(t) \stackrel{def}{=} \{r \in \mathcal{R} \mid (r,t) \in F\}$ the set of its pre-places *and by $R^+(t) \stackrel{def}{=} \{r \in \mathcal{R} \mid (t,r) \in F\}$ the set of its* post-places. *Similarly, for $r \in \mathcal{R}$, we denote $T^-(r) \stackrel{def}{=} \{t \in T \mid (t,r) \in F\}$ the set of its* incoming *transitions. If $(r,t',t) \in I$, for some $t' \in T^a$, we say that r is an* inhibitor place *for t. Finally, we denote $I(t) \stackrel{def}{=} \{(r,t') \in \mathcal{R} \times T^a \mid (r,t',t) \in I\}$.*

$\mathcal{N}$ is a flow net *(fNet), if 1) $R^-(t) \cap R^+(t) = \emptyset$, for any $t \in T$ (i.e. there are no looping transitions), and 2) $t' \in T^-(r)$, for all $(r,t',t) \in I$.*

As will be apparent from the following definitions, an inhibitor arc (r,t',t) checks for the absence of a token in the place r produced by the transition t'. The asterisk $*$ represents a *virtual initial transition* (see Def. 2 below).

Definition 2. *A* marking *of an fNet $(\mathcal{R},T,F,I)$ is a set of tokens $M \subseteq \mathcal{R} \times T^a$. We say that a token $(r,t) \in \mathcal{R} \times T^a$, has the* colour *$t$ and denote $T_M \stackrel{def}{=} \{t \in T^a \mid (r,t) \in M\}$ the set of colours involved in the marking M. A marking M is* initial *if $T_M = \{*\}$.*

Below, we will identify a marking M with its characteristic function $M : \mathcal{R} \times T^a \to \mathbb{B}$, where $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$. We now provide the formal semantics of fNets.

Definition 3. *A transition $t \in T$ of an fNet $(\mathcal{R},T,F,I)$ is* enabled *with a marking M if the following three conditions hold: 1) for each $r \in R^-(t)$, there is a token $(r,t') \in M$; 2) for each $r \in R^+(t)$, the corresponding token is not in M, i.e. $(r,t) \notin M$; 3) for each $(r,t',t) \in I$, the corresponding token is not in M, i.e. $(r,t') \notin M$. A marking is* final *if it does not enable any transitions.*

The marking M' obtained by firing a transition $t \in T$ enabled with M (denoted $M \xrightarrow{t} M'$) is defined by putting $M' \stackrel{def}{=} M \cup \{(r,t) \mid r \in R^+(t)\}$.

Notice that each transition can consume any token regardless of its colour: colours are relevant only for post-places and inhibitor arcs of transitions. Furthermore, transitions do not remove tokens from their pre-places.

In the rest of the paper, we use the following convention for the graphical representation of fNets: transitions that have not been fired are shown in black, whereas transitions that have already been fired—and therefore cannot be fired again—are shown in white. Moreover, in all the illustrations in the paper, token colours can be unambiguously derived by considering which transitions have been fired (visible from the black or white colour of the transition in the diagram). Therefore, we use the usual graphical notation for tokens in Petri nets, i.e. a bullet within the corresponding place.

Example 1 (Memory and Bus). Whenever an application requires a core and a memory bank on a Kalray platform, access to the bus is required implicitly. This dependency is modelled by the fNet shown in Fig. 2. The fNet has three places, p, m and b, corresponding to the processor, the memory and the bus. The

(a) Initial marking for the request $\{p, m\}$ (b) Final marking for the request $\{p, m\}$

Fig. 2: The cfNet modelling the dependency from Ex. 1

resource dependency is modelled by the transition t with incoming arcs from p and m, and one outgoing arc to b.

Consider an initial resource request $R = \{p, m\}$. The corresponding initial marking M_0 of the fNet has two tokens: $(p, *)$ and $(m, *)$ (Fig. 2a). Transition t is enabled and can be fired, generating the token (b, t). Thus, we have $M_0 \xrightarrow{t} M$ with M shown in Fig. 2b. Since t is not enabled with M, this marking is final.

Definition 4. *A run of an fNet from a marking M_0 is a sequence $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$. When such a run exists, we say that M_n is* reachable *from M_0 and write $M_0 \xrightarrow{\langle t_1, \ldots, t_n \rangle} M_n$. We say that a marking is* reachable *if it is reachable from some initial marking.*

Notice that, for any marking M obtained by firing a sequence of transitions, T_M (see Def. 2) is the set comprising $*$ and these transitions (see Prop. 1 below).

Definition 5. *A marking M of an fNet $(\mathcal{R}, T, F, I)$ is* well-formed *if, for all $t \in T_M \setminus \{*\}$, the following three conditions hold:*

1. *for all $r \in R^-(t)$, there exists a token $(r, t') \in M$, for some $t' \in T^-(r) \cup \{*\}$;*
2. *for all $r \in R^+(t)$, $(r, t) \in M$;*
3. *for all $(r, *) \in I(t)$, $(r, *) \notin M$.*

In Def. 5, conditions 1 and 3 are necessary for the transition t to have been enabled. They are not sufficient, since, for the transition to be enabled, inhibitor tokens referring to colours other than $*$ must also be absent from the marking. However, we cannot include this stronger requirement in the definition of well-formedness. Indeed, such inhibitor tokens can appear once t has already been fired. Condition 2 requires that all the tokens generated by firing t be, indeed, present in the marking.

Proposition 1. *Let M' be a marking reachable from an initial marking M_0 with $M_0 \xrightarrow{\langle t_1, \ldots, t_n \rangle} M'$. Then M' is well-formed and $T_{M'} = \{*, t_1, \ldots, t_n\}$.*

Marking well-formedness over-approximates reachability: all reachable markings are well-formed, but some well-formed markings are not reachable.

3.2 Constraints

For each resource $r \in \mathcal{R}$, we assume that possible amounts form an additive group $\langle D_r, +, 0\rangle$. We extend the definition of fNets by associating to each transition a constraint schema, instantiated into a constraint for a given final marking.

Definition 6. *Consider an fNet $(\mathcal{R}, T, F, I)$. For any transition $t \in T$, denote $X_t \stackrel{def}{=} \{x_r \,|\, r \in R^-(t) \cup R^+(t)\}$, where each x_r is a variable ranging over D_r. A* constraint schema *c_t associated to t is a predicate over X_t.*

Definition 7. *A* constraint-flow net *(cfNet) is a tuple $(\mathcal{R}, T, F, I, \mathcal{C})$, where $(\mathcal{R}, T, F, I)$ is an fNet and $\mathcal{C} = \{c_t \,|\, t \in T\}$ is a set of constraint schemata associated to the transitions in T.*

We build global constraint problems encoding resource allocations compatible with the causal dependencies defined by a cfNet. A constraint problem is based on an initial resource request and the constraint schemata associated to the transitions constituting a run of the cfNet. To this end, we introduce, for each place-colour pair $(r, t) \in \mathcal{R} \times T^a$, a variable d_r^t with the domain value D_r.

Definition 8. *Let M be a well-formed marking of a cfNet $(\mathcal{R}, T, F, I, \mathcal{C})$. We define a* platform constraint

$$C[M] \stackrel{def}{=} \bigwedge_{t \in T_M} c_t \left[\left(\sum_{t' : (r,t') \in M} d_r^{t'} \right) \Big/ x_r \;\middle|\; r \in R^-(t) \right] \left[d_r^t \Big/ x_r \;\middle|\; r \in R^+(t) \right], \quad (1)$$

where we denote by $E[x/y \,|\, C]$ the expression obtained by substituting in E all occurrences of y, which satisfy the condition C, by x. Thus, each conjunct in (1) is obtained by replacing, in the corresponding constraint schema c_t, 1) for each $r \in R^-(t)$, the variable x_r with the sum of all variables $d_r^{t'}$ corresponding to all the tokens $(r, t') \in M$; 2) for each $r \in R^+(t)$, the variable x_r with the corresponding variable d_r^t.

Notice that the conjuncts in (1) are unambiguously defined, since, by Def. 1, there are no looping transitions in the cfNet, i.e. $R^-(t) \cap R^+(t) = \emptyset$, for all $t \in T$. Hence the two substitutions operate on disjoint sets of variables.

Example 2 (Memory and Bus—continued). Building on Ex. 1, we introduce the constraint linking the actual resource requirements. Since any data to be written or read from the memory must transit through the bus, we associate to the transition t a constraint schema $c_t = (x_b \geq x_m)$, imposing that the required bus capacity be greater than or equal to the requested amount of memory.

Consider again the initial request $R = \{p, m\}$ with the corresponding initial marking in Fig. 2a. The variables corresponding to the initial tokens are d_p^* and d_m^*. Since the final marking M in Fig. 2b contains a token (b, t), we also introduce the corresponding variable d_b^t. Substituting these variables in the constraint schema for t, we obtain the platform constraint

$$C[M] = c_t\left[d_p^*/x_p, d_m^*/x_m, d_b^t/x_b\right] = (d_b^t \geq d_m^*)\,.$$

(a) Initial marking for the request $\{r\}$ (b) Final marking for the request $\{r\}$

Fig. 3: The cfNet modelling the dependency from Ex. 3

Example 3 (Virtual resources). Recall that the architecture of our running example consists of four identical processing elements p_1, p_2, p_3 and p_4, two identical memory sides left L and right R, each consisting of pairs of identical memory banks (m_1, m_2) and (m_3, m_4). An application may request a processing element and some memory. This request can be for a specific processing element, e.g. p_1 and a specific memory bank, e.g. m_1. However, if one of the requested resources is unavailable, the request will not be satisfied. Alternatively, the request can be made without specifying which of p_i and m_i is needed, allowing for a more flexible resource allocation. This can be modelled by introducing a "virtual" resource p for processing elements, L and R for memory sides or even more generally m for memory sides and banks as shown in Sect. 5.

Let us abstract from our example architecture and consider a system with two physical resources of the same type, r_1 and r_2, and a virtual resource r representing this resource type, modelled with a cfNet shown in Fig. 3. These resources could be, for example, two processing cores, memory sides or banks.

When the virtual resource is requested, the actual allocation depends on the policy that the system implements, for instance:

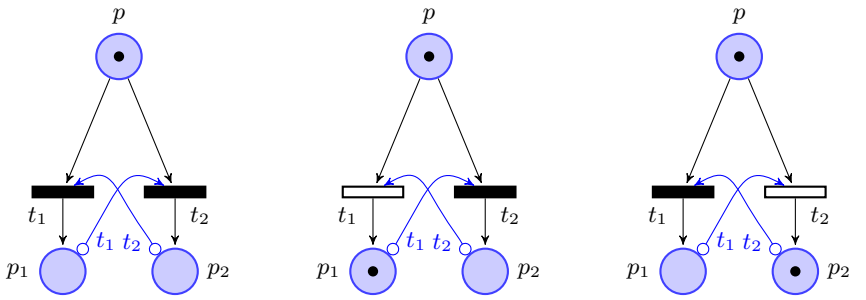
- *Dispatching the request:* one of r_1 or r_2 must provide the requested amount;
- *Redundant allocation:* both r_1 and r_2 must provide the requested amount;
- *Joint allocation:* part of the requested amount is provided by one of the two physical resources and the rest is provided by the other.

The request for *dispatching* allocation, when only one of the resources is actually allocated, is suitable for modelling the request of a processing core, while *redundant* and *joint* allocation can be used for memory request.

The constraint scheme of the transition depends on the policy:

$$\begin{aligned}
&\text{Dispatching the request:} && c_t = (x_{r_1} = x_r \wedge x_{r_2} = 0) \vee (x_{r_1} = 0 \wedge x_{r_2} = x_r)\,,\\
&\text{Redundant allocation:} && c_t = (x_{r_1} = x_r \wedge x_{r_2} = x_r)\,,\\
&\text{Joint allocation:} && c_t = (x_{r_1} + x_{r_2} = x_r)\,.
\end{aligned}$$

Consider the initial request $R = \{r\}$ with the joint allocation policy. The corresponding initial marking M_0 is shown in Fig. 3a. The variable corresponding to the initial token is d_r^*. Since the final marking M in Fig. 3b contains tokens (r_1, t) and (r_2, t), we also introduce the corresponding variables $d_{r_1}^t$ and $d_{r_2}^t$.



(a) Initial marking (b) Firing transition t_1 (c) Firing transition t_2

Fig. 4: The cfNet modelling virtual processing cores using inhibitor arcs

Substituting these variables in the constraint schema for t for the joint allocation policy, we obtain the platform constraint

$$C[M] = c_t\left[d^*_r/x_r, d^t_{r_1}/x_{r_1}, d^t_{r_2}/x_{r_2}\right] = (d^t_{r_1} + d^t_{r_2} = d^*_r)\,.$$

Ex. 4, below, presents an alternative approach for modelling virtual resources with the *dispatching* policy. This approach relies on *inhibitors* to generate simpler platform constraints involving less variables.

Example 4 (Virtual resources with inhibitors). Consider a different model of "virtual" resources, shown in Fig. 4 representing two processing cores of the Kalray architecture. The constraint schemata associated, respectively, to transitions t_1 and t_2 are $c_{t_1} = (x_p = x_{p_1})$ and $c_{t_2} = (x_p = x_{p_2})$.

In the dispatching allocation of Ex. 3, the constraint schemata ensured that only one core can be allocated for a single request. In the cfNet of Fig. 4, this is ensured by the inhibitor arcs (p_1, t_1, t_2) and (p_2, t_2, t_1). The initial marking for the request of a "virtual" processing core p is shown in Fig. 4a. Figures 4b and 4c show the two possible runs of the cfNet, where the firing of transition t_1 inhibits the firing of transition t_2 and vice versa.

Notice that the constraint schemata associated to the transitions t_1 and t_2 involve less variables than the dispatching schema in Ex. 3, simplifying the task of the constraint solver.

3.3 Allocation constraint problem

In the following, we assume that a partial cost function $cost_r : D_r \to \mathbb{R}$ is associated with each resource $r \in \mathcal{R}$. When defined, the value $cost_r(d)$ represents the cost of allocating the amount $d \in D_r$ of the resource r. When $cost_r(d)$ is undefined, this means that it is not possible to allocate the amount d of the resource r (e.g. d is greater than the resource capacity).

Definition 9. *Let $R \subseteq \mathcal{R}$ be a set of resources. A* utility function *over R is a partial function $u : \prod_{r\in\mathcal{R}} D_r \to \mathbb{R}$ such that u is constant on all D_r for $r \notin R$ (i.e. u depends only on resources belonging to R).*

Definition 10. *An* allocation *over a set of resources $R \subseteq \mathcal{R}$ is a value $d = (d_r)_{r\in\mathcal{R}} \in \prod_{r\in\mathcal{R}} D_r$, such that $d_r = 0$ for all $r \notin R$.*

Consider a system of resource dependencies defined by a cfNet N. Let $R_0 \subseteq \mathcal{R}$ be a set of resources corresponding to an initial request, and let u be a utility function over R_0. Let $M_0 = \{(r, *) \mid r \in R_0\}$ be the initial marking corresponding to R_0, and let M be a final marking obtained by running N. Let $C[M]$ be the corresponding platform constraint (see Def. 8). Finally, let $cost_r$, for all $r \in \mathcal{R}$, be the corresponding cost functions.

Definition 11. *An allocation $d = (d_r)_{r\in\mathcal{R}}$ over R is* valid*, if the predicate*

$$C_M(d) \stackrel{def}{=} C[M] \wedge \bigwedge_{r\in R} \Big(d_r = \sum_{t:(r,t)\in M} d_r^t\Big) \tag{2}$$

evaluates to true and if the following value is defined:

$$U_M(d) \stackrel{def}{=} u(d) - \sum_{r\in\mathcal{R}} cost_r(d_r). \tag{3}$$

We call the function $U_M(d)$ the global utility *of the allocation d.*

Finding an optimal resource allocation for a request R_0 is then formalised by the following constrained optimisation problem: $\operatorname{argmax}_{\{d \mid C_M(d)\}} U_M(d)$.

Notice that both the notions of validity and global utility, and the optimisation problem above depend on the marking M obtained by running the cfNet. In the next section we characterise those cfNets, where this dependency does not hold and provide a disambiguating mechanism for the rest of cfNets.

4 Conflicting dependencies

In the previous section, we have introduced the notion of cfNets and shown how the constraint problem associated to a resource request is built by running one. In particular, we have shown (Ex. 3 and 4) that, in cfNets, where only one among a set of alternative dependencies is to be activated, the use of inhibitors leads to simpler constraint problems with fewer variables.

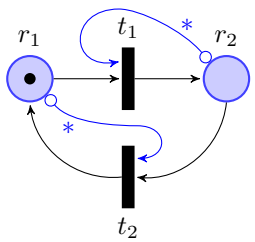


Fig. 5: Mutual dependency

Figure 5 shows another example, where inhibitors are useful. It models a system with two resources that must be used together: if one is requested, the other one should be included also; however, if both are requested initially, there is

no need to introduce additional constraints. This example cannot be realised without inhibitors.

Thus, inhibitor arcs increase the expressiveness of cfNets and are beneficial for the complexity of the resulting constraint problems. On the other hand, inhibitors can introduce conflicts between transitions, thereby introducing potential ambiguity in the definition of a constraint problem associated to a given initial marking. Below, we show that any cfNet that does not contain inhibitor arcs referring to token colours other than $*$, is conflict-free. We then provide a method for conflict detection in cfNets that do contain such inhibitors.

4.1 Conflicting transitions

Definition 12. *A cfNet $(\mathcal{R}, T, F, I, \mathcal{C})$ under marking M* has a conflict*, if there exist two distinct enabled transitions $t_1, t_2 \in T$, such that $M \xrightarrow{t_1} M'$ and t_2 is disabled with M'.*

We also say that transitions $t_1, t_2 \in T$ are in conflict under the marking M. *A cfNet is* conflict-free *if it does not have conflicts under any reachable marking.*

Proposition 2. *Any transition t, enabled with a marking M of a conflict-free cfNet, is also enabled with any marking reachable from M without firing t.*

An important consequence of Prop. 2 is that a platform constraint obtained by running a conflict-free cfNet depends only on the initial marking. Indeed, for a given initial marking the runs of the cfNet can only differ in the order of transition firing. However, the set of transitions is the same, generating the same conjuncts contributing to the platform constraint (1).

Proposition 3. *Let $(\mathcal{R}, T, F, I, \mathcal{C})$ be a cfNet and $t_1, t_2 \in T$ (with $t_1 \neq t_2$) be two transitions in conflict under some marking M. Then there exists a place $r \in \mathcal{R}$, such that either $(r, t_2, t_1) \in I$ or $(r, t_1, t_2) \in I$.*

A simple corollary of Prop. 3 is that any cfNet that does not contain inhibitor arcs referring to token colours other than $*$, is conflict-free. Notice, however, that Prop. 3 does not rely on the reachability of markings. Indeed, a conflict-free cfNet can still have conflicting transitions, provided that they are not enabled together under any reachable marking.

Definition 13. *Transitions t_1 and t_2 are* mutually exclusive *if no reachable marking enables them both.*

Figure 6 shows an example of two mutually exclusive transitions. Transitions t_1 and t_2 cannot be enabled simultaneously, since the place r_2 has a regular arc to t_2 and an inhibitor arc to t_1, thus one transition requires a token in r_2 while the other requires the place to be empty.

Definition 14. *An inhibitor arc (r, t', t) (with $t' \neq *$) is called* non-conflicting *if t is mutually exclusive with t'.*

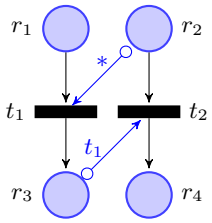


Fig. 6: A simple net with mutually exclusive transitions

Theorem 1. *A cfNet is conflict-free if and only if all its inhibitor arcs refer to initial tokens or are non-conflicting.*

Lemma 1. *Let $(\mathcal{R}, T, F, I, \mathcal{C})$ be a conflict-free cfNet, M be a marking enabling two transitions $t_1 \neq t_2$ and $M \xrightarrow{t_1} M_1' \xrightarrow{t_2} M_2'$ and $M \xrightarrow{t_2} M_1'' \xrightarrow{t_1} M_2''$ be two possible runs of N. Then $M_2' = M_2''$.*

Theorem 2. *A conflict-free cfNet is terminating and confluent, i.e. for any initial marking M_0, there exists a unique final marking M reachable from M_0.*

Theorem 2 implies that in a conflict-free cfNet, the platform constraint (1) depends only on the initial marking M_0, given by a request R_0. Therefore, the problem of finding a resource allocation defined by equations (2) and (3) in a conflict-free cfNet is defined uniquely.

4.2 Conflict detection

Theorem 1 provides a criterion characterising conflict-free cfNets: all the inhibitor arcs must refer to initial tokens or be non-conflicting. In order to determine whether an inhibitor arc (r, t', t) is non-conflicting, we must check whether t and t' are mutually exclusive. Mutual exclusiveness of two transitions requires that there be no reachable marking enabling them simultaneously (Def. 13). However, checking the existence of such a reachable marking by direct exploration is complex: in the worst case, the number of possible markings is of the order of $2^{|T^a| \times |\mathcal{R}|}$, since each transition—including the initial request—can potentially generate a token in each of the places. Instead, we exploit the notion of marking well-formedness, which over-approximates reachability (Prop. 1). Given two conflicting transitions of a cfNet $(\mathcal{R}, T, F, I, \mathcal{C})$ (fixed for the remainder of this sub-section), we proceed in three steps:

1. We encode the existence of a well-formed marking enabling both transitions as a Boolean satisfiability problem and submit it to a SAT-solver.
2. If the problem is unsatisfiable, the two transitions are mutually exclusive. Otherwise, the satisfying valuation returned by the SAT-solver encodes a well-formed marking, reachability whereof can be efficiently checked.
3. If this marking is reachable, the two transitions are not mutually exclusive. Otherwise, we repeat step 1 with a refined encoding excluding this marking.

Boolean encoding of transition enabledness. With each place-colour pair (r,t) we associate a Boolean variable y_r^t, evaluating to $\mathtt{tt}$ iff the corresponding token is present in a given marking. For a transition $t \in T$, we define the following four predicates on markings ($\mathcal{T}$ stands for *tokens*, $\mathcal{I}$ stands for *inhibitors*):

$$\mathcal{T}_t^- \stackrel{def}{=} \bigwedge_{r \in R^-(t)} \Big(\bigvee_{t \in T^-(r)} y_r^t \Big), \qquad \text{// tokens are present in pre-places of } t \tag{4}$$

$$\mathcal{T}_t^+ \stackrel{def}{=} \bigwedge_{r \in R^+(t)} y_r^t, \qquad \text{// tokens are present in post-places of } t \tag{5}$$

$$\mathcal{I}_t^* \stackrel{def}{=} \bigwedge_{(r,*,t) \in I} \overline{y_r^*}, \qquad \text{// tokens are absent from initial inhibitors of } t \tag{6}$$

$$\mathcal{I}_t^\circ \stackrel{def}{=} \bigwedge_{(r,t',t) \in I,\, t' \neq *} \overline{y_r^{t'}}\,. \quad \text{// tokens are absent from non-initial inhibitors of } t \tag{7}$$

For a well-formed marking M, if the transition t has already been fired in the run leading to M, then $\mathcal{T}_t^+(M)$ evaluates to $\mathtt{tt}$. Thus, t *is enabled* under M iff $\mathcal{E}_t(M) \stackrel{def}{=} \mathcal{T}_t^- \wedge \overline{\mathcal{T}_t^+} \wedge \mathcal{I}_t^* \wedge \mathcal{I}_t^\circ = \mathtt{tt}$.

Lemma 2. *If a transition t is enabled with a marking M, then holds the equality $\mathcal{E}_t(M) = \mathtt{tt}$. Conversely, if $\mathcal{E}_t(M) = \mathtt{tt}$ and M is well-formed then t is enabled.*

Lemma 2 provides a characterisation of transition enabledness under well-formed markings. The following results provide a similar characterisation of the well-formedness of a marking.

Transition t *may have been enabled* in the run leading to a marking M only if $\mathcal{B}_t(M) \stackrel{def}{=} \mathcal{T}_t^- \wedge \mathcal{I}_t^* = \mathtt{tt}$. Notice that the stronger predicate $\mathcal{T}_t^- \wedge \mathcal{I}_t^* \wedge \mathcal{I}_t^\circ$ does not characterise the desired property, since some of the tokens inhibiting t may have been generated after t has been fired (clearly, this cannot be the case for the initial tokens). Notice also that once $\mathcal{B}_t(M)$ holds for some marking M, it will hold for all markings reachable from M.

The well-formedness of a marking essentially means that, for every non-initial token, there is a transition that could have generated it. Thus, a marking M is well-formed iff $\mathcal{W}(M) \stackrel{def}{=} \bigwedge_{r \in R} \bigwedge_{t \in T^-(r)} \big(y_r^t \Rightarrow \mathcal{B}_t \wedge \mathcal{T}_t^+\big) = \mathtt{tt}$.

Thus, the fact that two transitions, t_1 and t_2 can be both enabled with the same well-formed marking is encoded by the predicate $\mathcal{E}_{t_1} \wedge \mathcal{E}_{t_2} \wedge \mathcal{W}$. If this predicate is not satisfiable then transitions t_1 and t_2 are mutually exclusive. Recall from Sect. 3.1 that well-formedness is an over-approximation of reachability. Therefore, the converse does not hold: given a marking M satisfying $\mathcal{E}_{t_1} \wedge \mathcal{E}_{t_2} \wedge \mathcal{W}$, one has to check whether M is reachable.

Marking reachability. Let $M \subseteq \mathcal{R} \times T^a$ be a well-formed marking of a cfNet $(\mathcal{R}, T, F, I, \mathcal{C})$. We associate to the marking M the corresponding *causality graph*, which is a directed hyper-graph $\mathcal{G}_M \stackrel{def}{=} (V, E)$ with vertices $V = T_M$ and the set $E \subseteq T_M \times 2^{T_M}$ of edges representing the "must be fired before" relation among the corresponding transitions: an edge (t, T) with $T \in 2^{T_M}$ means that, *for the transition t to be fired, at least one transition in T must be fired before t.*

We put $E = E_1 \cup E_2$ with E_1 and E_2 defined by (8) and (9) below.

For a place $r \in \mathcal{R}$, denote $T_r \stackrel{def}{=} \{t \in T_M \mid (r,t) \in M\}$ the set of colours of the tokens in r present in the marking M. We put

$$E_1 \stackrel{def}{=} \left\{ (t, T_r) \in T_M \times 2^{T_M} \;\middle|\; t \in T_M, r \in R^-(t) \right\}. \tag{8}$$

By Def. 3, for any transition t to be fired, it is necessary that in each place $r \in R^-(t)$ there be at least one token. Hence, at least one of the transitions generating such tokens must be fired before t.

Furthermore, if firing a transition t' generates a token that inhibits some transition t, then firing of t cannot happen after that of t'. Thus, we put

$$E_2 \stackrel{def}{=} \left\{ (t', \{t\}) \in T_M \times 2^{T_M} \;\middle|\; \exists r \in \mathcal{R} : (r, t', t) \in I \right\}. \tag{9}$$

Notice that, if such $(r, t', t) \in I$ actually exists, necessarily $(r, t') \in M$, since $t' \in T_M$. Thus, we do not have to state this condition explicitly in (9).

Definition 15. *Let $G = (V, E)$ with $E \subseteq V \times 2^V$ be a hyper-graph. A* path *in G is a sequence $(e_i)_{i=0}^n$, with $e_i = (v_i, S_i) \in E$, such that $v_{i+1} \in S_i$, for all $i < n$. When $n \in \mathbb{N}$, we say that the path is* finite*, otherwise, when $n = \infty$, it is* infinite*. We say that the path* starts with the edge e_0.

Definition 16. *A hyper-graph $G = (V, E)$ with $E \subseteq V \times 2^V$* has a cycle $C \subseteq V$, *if there exists a set of finite paths $\{(e_i^j)_{i=0}^{n_j}\}_{j \in J}$, with $e_i^j = (v_i^j, S_i^j) \in E$, such that $C = \{v_i^j \mid j \in J, i \in [0, n_j]\}$ and, for all $j \in J$ and $i \in [0, n_j]$, we have $S_i^j \subseteq C$. Otherwise, G is said to be* free from cycles.

Theorem 3. *Let M be a well-formed marking and $\mathcal{G}_M$ its causality graph. The marking M is reachable iff $\mathcal{G}_M$ is free from cycles.*

Cycle-freedom of a hyper-graph can be checked in linear time [7].

Encoding refinement. Let $M \subseteq \mathcal{R} \times T^a$ be a well-formed marking of a cfNet $\mathcal{N}$, enabling two conflicting transitions t_1 and t_2. If M is reachable, $\mathcal{N}$ has a conflict. If it is not reachable, the encoding has to be refined to exclude M. Let Φ be the predicate used at the previous step of the process (initially $\Phi = \mathcal{E}_{t_1} \wedge \mathcal{E}_{t_2} \wedge \mathcal{W}$). We refine this predicate by taking $\Phi \wedge \overline{\Phi_M}$, where $\Phi_M = \bigwedge_{(r,t) \in M} y_r^t \wedge \bigwedge_{(r,t) \notin M} \overline{y_r^t}$ is the characteristic predicate of the marking M.

4.3 Priority

As shown in Sect. 4.1, conflict-free cfNets are confluent: the same platform constraint is obtained by any run of the cfNet, for a given initial marking. In other words, enabled transitions can be fired in arbitrary order. This is not the case for cfNets with conflicts: firing one of two conflicting transitions disables the other one, generating different platform constraints. Thus, for reachable conflicts, the choice of which of the two conflicting transitions should be fired, has to be resolved externally to the cfNet. This can be achieved by introducing priority among the conflicting transitions.

For a cfNet $N = (\mathcal{R}, T, F, I, \mathcal{C})$, a priority relation is a partial order $> \subseteq T \times T$ on its set of transitions. For two transitions t_1 and t_2, a priority $t_1 > t_2$ means that when both transitions are enabled, t_1 must be fired before t_2. Priorities can be defined statically or dynamically, depending, for example, on the availability of the resources corresponding to the post-places of the two transitions.

Example 5 (Virtual resources with inhibitors—continued). In Ex. 4 we have presented a cfNet modelling a "virtual" resource p (see Fig. 4) representing two processing cores p_1 and p_2. With the initial request p there can be two possible runs of the cfNet as shown in Fig. 4b and 4c.

The fact that a virtual resource is used to represent the two cores implies that they are functionally equivalent. However, we can consider a scenario, where the two cores differ in their non-functional properties. For instance, suppose that p_1 has better energy efficiency than p_2. Imposing the priority $t_1 > t_2$ for the cfNet in Fig. 4a, would ensure that p_1 is allocated rather than p_2.

Consider now the second scenario, where two applications are running on this platform, both requiring a processing core, but unaware of the platform architecture. In the first cycle, one of the applications requests p and p_1 is allocated as discussed above. In the next cycle, the second application also requests p. Since, p_1 is not available (indicated by its cost function being undefined for all values), we inverse the priority, setting $t_1 < t_2$. Thus, t_2 will be fired leading to an allocation of p_2 to the second application.

Finally, notice that, if none of p_1 and p_2 is available, the choice of priority is irrelevant, since the constraint problems generated from both markings in Fig. 4b and 4c will be unsatisfiable.

5 The Kalray architecture case study

Figure 7 shows the complete cfNet modelling a single cluster of the Kalray architecture described in Sect. 2. For the sake of clarity, we group the resources in several boxes: one for each group of processors, one for each memory side and one for each bus. In order to further unclutter the figure, the smaller rectangles on the sides of each box allow us to group the arcs of the cfNet. For instance, the thicker red arc in the figure represents four arcs of the cfNet, going from places p_1 and p_2 to transitions t_{61} and t_{62}.

Each memory side box repeats the virtual resource pattern (Ex. 3). The arcs between a processor box, a memory box and a bus box reproduce the memory and bus example (Ex. 1).

The architectural constraints are modelled as follows: 1) transition t_1 ensures mutual exclusion among processors p_1, p_2, p_3 and p_4; 2) transition t_2 ensures the mutual exclusion among memory sides, and t_3, t_4—among memory banks of each side respectively; 3) transitions t_{51}, t_{52}, t_{61}, t_{62}, t_{71}, t_{72}, t_{81} and t_{82} ensure that only one of the processors from one group can have access to one memory side using a dedicated bus.

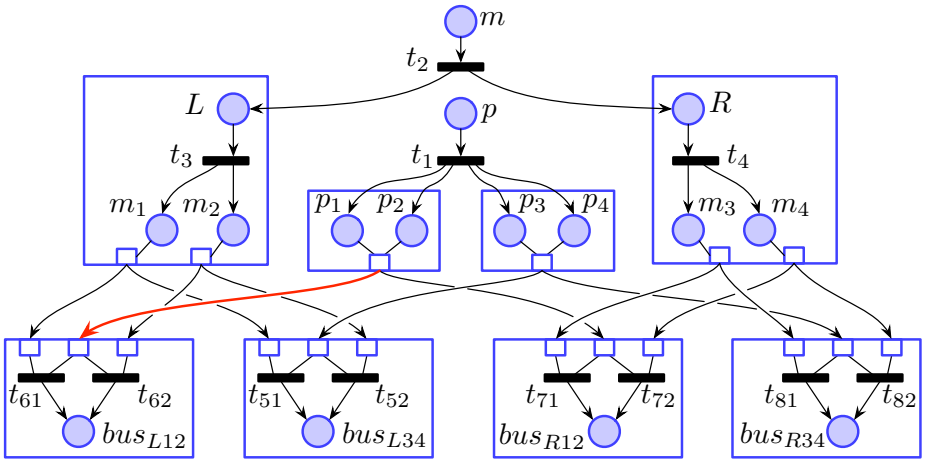


Fig. 7: A cfNet modelling a Kalray cluster

It is possible to request either a virtual processing core p or a specific one out of p_1, p_2, p_3 or p_4. Similarly, it is possible to request virtual memory m, a specific memory side L or R, or a specific memory bank m_1, m_2, m_3 or m_4.

The resource allocation constraints for the existing transitions implementing the dispatching policy are as follows (S_4 is the symmetric group on $\{1,2,3,4\}$):

$$
\begin{aligned}
c_{t_1}(x_p, x_{p_1}, x_{p_2}, x_{p_3}, x_{p_4}) &= \bigvee_{\sigma \in S_4} (x_{p_{\sigma(1)}} = x_p \ \wedge \ x_{p_{\sigma(2)}} = x_{p_{\sigma(3)}} = x_{p_{\sigma(4)}} = 0),\\
c_{t_2}(x_m, x_L, x_R) &= (x_L = x_m \wedge x_R = 0) \vee (x_R = x_m \wedge x_L = 0),\\
c_{t_3}(x_L, x_{m_1}, x_{m_2}) &= (x_{m_1} = x_L \wedge x_{m_2} = 0) \vee (x_{m_2} = x_L \wedge x_{m_1} = 0),\\
c_{t_{51}}(x_{m_1}, x_{p_3}, x_{p_4}, x_{bus_{L34}}) &= \big(x_{bus_{L34}} > 0 \wedge x_{m_1} > 0 \wedge\\
&\qquad ((x_{p_3} = 0 \wedge x_{p_4} > 0) \vee (x_{p_4} = 0 \wedge x_{p_3} > 0))\big)\\
&\qquad \vee \big(x_{bus_{L34}} = 0 \wedge ((x_{p_3} = 0 \wedge x_{p_4} = 0) \vee x_{m_1} = 0)\big),
\end{aligned}
$$

c_{t_4} is similar to c_{t_3}, whereas $c_{t_{52}}$, $c_{t_{61}}$, $c_{t_{62}}$, $c_{t_{71}}$, $c_{t_{72}}$, $c_{t_{81}}$, $c_{t_{82}}$ are similar to $c_{t_{51}}$.

The initial cost functions for the resources are as follows: 1) for all $i \in [1,4]$, $cost_{p_i}(d) = 0$, if $d \in \{0,1\}$ and undefined otherwise; 2) for all $i \in [1,4]$, $cost_{m_i}(d) = 0$, if $0 \leq d \leq 128$ and undefined otherwise; 3) for $i \in \{L,R\}$ and $j \in \{12,34\}$, $cost_{bus_{ij}}(d) = 0$, if $d \in \{0,1\}$ and undefined otherwise; 4) the cost function for p, m, L, R are defined everywhere as constant 0.

Here, the focus is on resource availability, rather than the actual cost. Hence, all cost functions are 0 when defined. Notice also, that we require the whole bus once a memory is requested, regardless of the amount of data to be passed.

6 Related work

The idea of using Petri nets for resource management is not novel. A number of works explore Resource Allocation Systems (RAS) in the context of Flexible Manufacturing Systems (FMS) [9] and introduce different subclasses of Petri nets to account for various allocation requirements [14,28]. In [21] the authors investigate how the methods used in FMS can be extended to software applications with concurrent processes competing for shared resources. They propose a new subclass of Petri nets, PC^2R, where the resources of different types, their availability and control flow of each process are represented as a unique system. A small change in the process flow or resources set will require a change of this model. In contrast, we are applying the separation of concerns principle by providing distinct models for systems of interdependent resources with their available capacities and for resource requests from abstract applications.

Numerous works study constraints and dependencies, such as temporal, causality and resource constraints, in various contexts by considering the underlying dependency graphs, e.g. [1,4,18,25]. Due to their syntactic structure inspired from Petri nets, cfNets generalise these approaches.

The term "constraint-flow net" can be confused with "network of constraints", a notion initially proposed in [22]. In fact, these two formalisms are very different. Constraint-flow nets consist of places and transitions, representing, respectively, resources and dependencies among them. Each transition has an associated constraint schema, whereof the arity is the total number of incoming and outgoing arcs of the transitions. These constraint schemata are used to generate constraints imposed on the possible resource allocations by the execution platform modelled by the cfNet. On the other hand, networks of constraints have been proposed in [22] to palliate the combinatorial explosion, when a given relation among n variables (constraint) is represented by an n-dimensional $(0, 1)$-matrix. Instead, such n-ary constraint can be optimally approximated by a network of n binary ones, thereby considerably reducing the size of the representation.

Constraint-flow nets bear some similarity with event structures [23], in particular the more recent generalisations involving conflict [30] and dynamicity [3]. Event structures encode causality and conflict relations among concurrent events. The generalisation proposed in [30] matches the expressive power of arbitrary Petri nets. This suggests that it should be capable of encoding the flow part of cfNets: in our context, an event structure configuration would represent the set of transitions having been fired. However, since places—representing resources in cfNets—do not appear explicitly in event structures, it is not immediately clear how the initial requests and constraint schemata should be encoded. We leave a more detailed study of the correspondence between the various forms of event structures and cfNets for future work.

Constraint-flow nets are specifically tailored to provide a natural way of incorporating constraint schemata that allow expressing quantitative dependencies among amounts of allocated resources. To the best of our knowledge, such combinations of constraint schemata with an underlying graph structure have not been studied in existing literature.

7 Conclusion

In this paper, we have introduced constraint-flow nets (cfNets) that allow modelling resource dependencies, thereby bridging a gap in the current state-of-the-art approaches to the specification of requests for resources and resource allocation to applications: resource allocation commonly relies on the assumption that the requested resources are completely specified, whereas specification languages operate with high-level abstractions of resources, e.g. "memory" or "thread", leaving the resource manager to endow these with precise semantics.

The cfNet model provides a means to formally specify the structure of resources provided by the platform and the dependencies among them. This specification serves as an abstraction layer, which allows designers to focus on the resources immediately relevant to the application functionality, while taking care of low-level structure and dependencies inherent to the specific target platform. Thus, our approach simplifies application design and greatly enhances portability. This is particularly useful for platforms with complex resource architectures, such as the Massively Parallel Processor Arrays (MPPA) (e.g. Kalray), or cloud platforms, where the resource architecture can change dynamically at run time.

In this paper, we have defined the cfNet model and provided its semantics, defining a constraint problem for a given initial resource request. The cfNet model comprises inhibitor arcs. On one hand, these increase the expressiveness of the model and simplify certain constraint problems by reducing the number of variables involved. On the other hand, inhibitors can generate conflicts introducing ambiguity in the constraint problem definition. We have provided a sufficient condition—which can be easily checked syntactically—for the cfNet to be conflict-free. For cfNets that do not satisfy this condition, we have provided an efficient method for determining whether a given inhibitor induces a conflict. As shown by the virtual resources example, reachable conflicts can appear, for example, when the use of different instances of the same similar resource type are preferable in different situations. Such conflicts can be dynamically resolved by defining priorities between conflicting transitions.

In future work, we are planning to further improve the conflict detection algorithm: the encoding refinement presented in this paper excludes only one marking; by exploiting the causality hyper-graph, more unreachable markings could be excluded in one step. Similarly, the structure of a cfNet could be exploited for building concurrent or distributed resource allocators. We also consider implementing the cfNet model in the JavaBIP [5,6] component coordination framework in order to evaluate the practical performance, using various SMT and constraint solvers.

Acknowledgements This paper has received a large number of very constructive comments. Although—mostly due to space and time limitations—we did not manage to address all of them, we are very grateful to the anonymous reviewers for their suggestions that we hope to implement in our future work.

References

1. M. K. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem determination using dependency graphs and run-time behavior models. In A. Sahai and F. Wu, editors, *Utility Computing: Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004)*, volume 3278 of *Lecture Notes in Computer Science*, pages 171–182. Springer Berlin Heidelberg, 2004.
2. C. Ansótegui, M. Bofill, M. Palahí, J. Suy, and M. Villaret. Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem. In *Symposium on Abstraction, Reformulation, and Approximation*, 2011.
3. Y. Arbach, D. Karcher, K. Peters, and U. Nestmann. Dynamic causality in event structures. In *International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2015)*, pages 83–97. Springer, 2015.
4. D. A. Berson, R. Gupta, and M. L. Soffa. GURRR: A global unified resource requirements representation. *SIGPLAN Not.*, 30(3):23–34, Mar. 1995.
5. S. Bliudze, A. Mavridou, R. Szymanek, and A. Zolotukhina. Coordination of software components with BIP: Application to OSGi. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, MiSE 2014, pages 25–30, New York, NY, USA, 2014. ACM.
6. S. Bliudze, A. Mavridou, R. Szymanek, and A. Zolotukhina. Exogenous coordination of concurrent software components with JavaBIP. *Software: Practice and Experience*, 2017. Early view: `http://dx.doi.org/10.1002/spe.2495`.
7. S. Bliudze, A. Simalatsar, and A. Zolotukhina. Modelling resource dependencies. Technical Report 218599, EPFL, May 2016. Available online: `https://infoscience.epfl.ch/record/218599`.
8. A. A. Chien, H. Casanova, Y.-S. Kee, and R. Huang. The virtual grid description language: vgDL. Technical Report CS2005-0817, Department of Computer Science and Engineering, University of California, San Diego, 2004.
9. J. M. Colom. The resource allocation problem in flexible manufacturing systems. In W. M. P. Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference*, volume 2679 of *Lecture Notes in Computer Science*, pages 23–35. Springer Berlin Heidelberg, June 2003.
10. Y. Cui and K. Nahrstedt. QoS-aware dependency management for component-based systems. In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pages 127–138, 2001.
11. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82. Springer Berlin Heidelberg, 1998.
12. B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 97:1–97:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association.
13. C. Ensel and A. Keller. An approach for managing service dependencies with XML and the resource description framework. *Journal of Network and Systems Management*, 10(2):147–170, June 2002.
14. J. Ezpeleta, J. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11:173–184, 04/1995 1995.

15. G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B. de Dinechin. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems*, pages 1–51, 2015.
16. Kalray. Kalray MPPA-256. `http://www.kalray.eu/IMG/pdf/FLYER_MPPA_MANYCORE.pdf`, Mar. 2015.
17. Y.-S. Kee, D. Logothetis, R. Y. Huang, H. Casanova, and A. A. Chien. Efficient resource description and high quality selection for virtual grids. In *CCGRID*, pages 598–606. IEEE Computer Society, 2005.
18. A. A. Kountouris and C. Wolinski. Hierarchical conditional dependency graphs for conditional resource sharing. In *Euromicro Conference, 1998. Proceedings. 24th*, volume 1, pages 313–316 vol.1, Aug 1998.
19. K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.
20. O. Lassila and R. R. Swick. Resource description frame-work (RDF) model and syntax specification. Technical Report REC-rdf-syntax-19990222, World Wide Web Consortium (W3C), Feb. 1999.
21. J.-P. López-Grao and J.-M. Colom. A Petri net perspective on the resource allocation problem in software engineering. In K. Jensen, S. Donatelli, and J. Kleijn, editors, *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900 of *Lecture Notes in Computer Science*, pages 181–200. Springer Berlin Heidelberg, 2012.
22. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
23. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.
24. R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, Sept. 1999.
25. P. Senkul and I. H. Toroslu. An architecture for workflow scheduling under resource allocation constraints. *Information Systems*, 30(5):399 – 422, 2005.
26. P. Tendulkar, P. Poplavko, I. Galanommatis, and O. Maler. Many-core scheduling of data parallel applications using SMT solvers. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 615–622. IEEE, 2014.
27. P. Tendulkar, P. Poplavko, J. Maselbas, I. Galanommatis, and O. Maler. A run-time environment for real-time streaming applications on clustered multi-cores. Technical report, Verimag, 2015.
28. F. Tricas, F. Garcia-Valles, J. M. Colom, and J. Ezpeleta. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 271–277, April 2005.
29. H. N. Van, F. D. Tran, and J. M. Menaud. SLA-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*, volume 1, pages 357–362, Oct 2009.
30. R. van Glabbeek and G. Plotkin. Event structures for resolvable conflict. In *29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, pages 550–561. Springer, 2004.
31. J. Vanderham, F. Dijkstra, F. Travostino, H. Andree, and C. Delaat. Using RDF to describe networks. *Future Generation Computer Systems*, 22(8):862–867, Oct. 2006.