Name creation vs. Replication in Petri Net Systems*

Fernando Rosa-Velardo and David de Frutos-Escrig

Dpto. de Sistemas Informáticos y Programación
Universidad Complutense de Madrid
{fernandorosa,defrutos}@sip.ucm.es

Abstract. We study the relationship between name creation and replication in a setting of infinite-state communicating automata. By name creation we mean the capacity of dynamically producing pure names, with no relation between them other than equality or inequality. By replication we understand the ability of systems of creating new parallel identical threads, that can synchronize with each other. We have developed our study in the framework of Petri nets, by considering several extensions of P/T nets. In particular, we prove that in this setting name creation and replication are equivalent, but only when a garbage collection mechanism is added for idle threads. However, when simultaneously considering both extensions the obtained model is, a bit surprisingly, Turing complete and therefore, more expressive than when considered separately.

1 Introduction

Extensive work has been developed in the last years in the field of multithreaded programs [21]. In general, once thread synchronization is allowed, reachability becomes undecidable [18], so that the effort is devoted to compute overapproximations of the set of reachable markings. For instance, [2] studies the case where threads are finite state, [3] considers a framework to statically analyze multithreaded systems with a fixed number of infinite-state communicating threads, and in [4] that work is extended to cope with dynamic creation of threads.

Dynamic name generation has also been thoroughly studied, mainly in the field of security and mobility [13]. Paradigmatic examples of nominal calculi are the π-calculus [16], and the Ambient Calculus [6]. Since the early versions of all these calculi are Turing complete, numerous efforts have been focused on the static analysis of undecidable properties [7, 17] and in the restriction of the original formalisms [5, 27] to get more manageable languages where some interesting properties become decidable.

In this paper we investigate the relationships between name creation and replication in this setting of infinite-state communicating automata, that in our

* Work partially supported by the Spanish projects DESAFIOS TIN2006-15660-C02-02, WEST TIN2006-15578-C02-01 and PROMESAS-CAM S-0505/TIC/0407.

case are Petri nets. By name creation we mean a mechanism to generate fresh names that can be communicated between components, with no relation between them other than equality or inequality, which have been called pure names in [13]. By replication we understand the capability of processes of creating an exact replica of themselves, though always starting its execution in a fixed initial state. Once created, these replicated processes evolve in an independent way, though possibly interacting with other processes.

We will prove that these two mechanism are equivalent, in the sense that they simulate each other. On the one hand, names can be used to represent the states of the different threads of a process that are running in parallel over a single copy of the (virtually) replicated process. On the other hand, different copies of the same component can be used to mimic the effect of adding pure names. In the proof of this equivalence it is essential to consider a garbage collection mechanism that removes idle threads, given that the corresponding garbage collection mechanism for names is implicit in the model (the set of used names is not an explicit part of the state). In fact, reachability is undecidable for the extension with name creation (equivalently with replication and garbage collection), but decidable if no such mechanism is considered.

However, though name creation and replication are equivalent, it turns out that they do not overlap, in the sense that a model dealing simultaneously with both surpasses the strength of any of these two features, reaching indeed Turing completeness. The intuitive explanation of why this is true is that once we have names, they can be used to distinguish between what at first were indistinguishable components, so that now threads can have a unique personality.

A formalism that encompasses both dynamic name generation and replication is TDL [9]. In TDL there are primitives both for name creation and thread creation. However, no garbage collection mechanism is considered, neither for names nor for threads. As a consequence, and quite surprisingly, reachability is decidable, though coverability is undecidable.

The remainder of the paper is structured as follows. Section 2 presents the basic model, that will be the starting point of our work. Section 3 extends the basic model with a mechanism for name creation and name communication, and gives a brief insight of the expressive power of the obtained model. Section 4 extends again the basic model, but now with a replication primitive, and proves the equivalence between the two extensions. In Section 5 we consider the model obtained when introducing the two features at the same time, and prove its Turing-completeness. Section 6 presents some results regarding boundedness. Finally, Section 7 presents our conclusions and some directions for further work.

2 The basic model

In order to concentrate on the two features of interest we will first consider a very basic model, based on Petri Nets, which could be considered not too useful in practice, but which will be an adequate starting point for later extensions. First, let us introduce some notations that we will use throughout the paper.

Given an arbitrary set A we will denote by $\mathcal{MS}(A)$ the set of finite multisets of A, that is, the set of mappings $m : A \to \mathbb{N}$. We denote by $S(m)$ the support of m, that is, the set $\{a \in A \mid m(a) > 0\}$, $|m| = \sum_{a \in S(m)} m(a)$, and by $+$ and $-$ the sum and difference operators for multisets, to distinguish them from $\cup$ and $\setminus$, the corresponding operators over sets. If $f : A \to B$ is an injective mapping and $m \in \mathcal{MS}(A)$, then we define $f(m) \in \mathcal{MS}(B)$ by $f(m)(b) = m(a)$ if $b = f(a)$ or $f(m)(b) = 0$, otherwise. We will consider a set $\mathcal{S}$ of service names, endowed with a function $arity : \mathcal{S} \to \{n \in \mathbb{N} \mid n \geq 2\}$ and we take the set of synchronizing labels $Sync = \{s(i) \mid s \in \mathcal{S},\ 1 \leq i \leq arity(s)\}$. If $arity(s) = 2$ then we will write $s?$ and $s!$ instead of $s(1)$ and $s(2)$, respectively, that can be interpreted as the offer and request of an ordinary service. We also use a set $\mathcal{A}$ of labels for autonomous transitions.

Definition 1. *A component net is a labelled Petri net $N = (P, T, F, \lambda)$ where:*

- *P and T are disjoint finite sets of places and transitions, respectively,*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net, and*
- *λ is a function from T to the set of labels $\mathcal{A} \cup Sync$.*

A marking M of N is a finite multiset of places of N, that is, $M \in \mathcal{MS}(P)$.

As usual, we denote by $t^\bullet$ and ${}^\bullet t$ the set of postconditions and preconditions of t, respectively, that is, $t^\bullet = \{p \mid (t, p) \in F\}$ and ${}^\bullet t = \{p \mid (p, t) \in F\}$.

Definition 2. *A basic net system is a set $\mathcal{N}$ of pairwise disjoint component nets. A marking of $\mathcal{N}$ is a set of markings of its components, one marking per component.*

For a net system $\mathcal{N}$ we will denote by P, T, F and λ the union of the corresponding elements in each net component, and we will denote simply by F the characteristic function of the set F. Analogously, we will consider markings as multisets of P. Now let us define the firing rules of the two kind of transitions.

Definition 3. *Let $\mathcal{N}$ be a basic net system and M a marking of $\mathcal{N}$. An autonomous transition, $t \in T$ with $\lambda(t) \in \mathcal{A}$, is enabled at marking M if ${}^\bullet t \subseteq M$. The reached state of $\mathcal{N}$ after the firing of t is $M' = M - {}^\bullet t + t^\bullet$.*

Therefore, autonomous transitions are exactly as ordinary transitions in P/T nets (see Fig. 1 left). This is not the case for synchronizing transitions.

Definition 4. *Let $\mathcal{N}$ be a basic net system and $s \in \mathcal{S}$ with $arity(s) = n$. The transitions in a tuple $\overline{t} = (t_1, \ldots, t_n)$ are said to be compatible for s if $\lambda(t_i) = s(i)$ for all $i \in \{1, \ldots, n\}$. We write ${}^\bullet\overline{t} = \sum_{i=1}^{n} {}^\bullet t_i$ and $\overline{t}^\bullet = \sum_{i=1}^{n} t_i^\bullet$.*

Definition 5. *Let $\mathcal{N}$ be a basic net system and M a marking of $\mathcal{N}$. A tuple of synchronizing transitions $\overline{t}$ is enabled at marking M if ${}^\bullet\overline{t} \subseteq M$. The reached state of $\mathcal{N}$ after the firing of $\overline{t}$ is $M' = M - {}^\bullet\overline{t} + \overline{t}^\bullet$.*

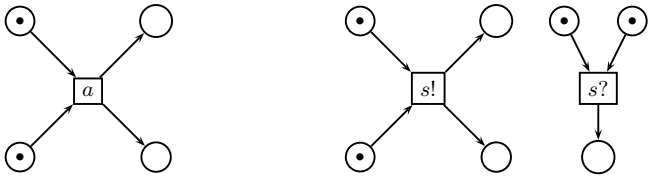

Fig. 1. Autonomous (left) and synchronizing (right) transitions

Thus, any synchronizing transition (Fig. 1 right) needs the presence of compatible enabled transitions, in which case they can be fired simultaneously. Notice that since the different nets of a basic net system do not have a name they must be addressed by communications in an anonymous way. Indeed, a net component is willing to synchronize with any nets that have enabled compatible transitions. Intuitively, it is willing to receive a service from anyone offering it. Therefore, there is no way in the present model to discriminate between different components. In fact, a process willing to synchronize can do it with any compatible counterpart, and its state after the execution will be the same whichever was that counterpart. In [12] we proved the analogous to the following result, but with a model with only two-way synchronizations.

Proposition 1. *Every Basic Net System can be simulated by a P/T net.*

The proof basically consists on having a different transition for every tuple of compatible synchronizing transitions, so that the firing of the tuple $\overline{t}$ is simulated by the firing of transition $\overline{t}$. In fact, there we proved this result for a model which also considered localities, so that each component is localized and can only synchronize with co-located components, thus proving that these locations do not introduce any further expressive power.

In the result above and some other times along the paper we assert that a model $\mathbf{M}'$ simulates another model $\mathbf{M}$. By that we mean here that for every system $\mathcal{N}$ in $\mathbf{M}$ there is $\mathcal{N}' = F(\mathcal{N})$ in $\mathbf{M}'$, where F is a computable function, such that the transition systems generated by the semantics of $\mathcal{N}$ and $\mathcal{N}'$ are isomorphic. Therefore, reachability in $\mathcal{N}$ and $\mathcal{N}'$ are equivalent. Moreover, that isomorphism also preserves the considered orders between markings, so that coverability and boundedness are also preserved. As a consequence, whenever reachability, coverability or boundedness are decidable in $\mathbf{M}'$ they are also decidable in $\mathbf{M}$.

Since reachability, coverability and boundedness are decidable for P/T nets, we have the following:

Corollary 1. *Reachability, coverability and boundedness are decidable for Basic Net Systems.*

3 Name creation

In this section we extend the previous model with the capability of name management. Names can be created, communicated between components and used to

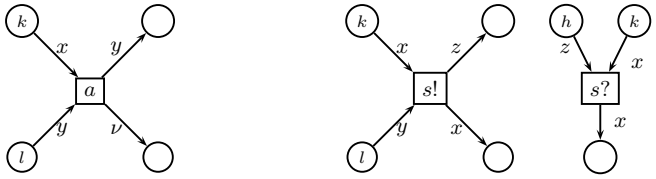

Fig. 2. Autonomous (left) and synchronizing (right) transitions

restrict synchronizations to happen only between components that know a particular name, as we have illustrated with several examples in [24]. We can use this mechanism to deal with authentication issues in our systems. We formalize the latter by replacing ordinary tokens by distinguishable tokens, thus adding a touch of colour to our nets. In order to handle these colours, we need matching variables labelling the arcs of the nets, taken from a set Var. Moreover, we add a primitive capable of creating fresh names, formalized by means of a special variable $\nu \in Var$.

Definition 6. *A ν-net component is a labelled coloured Petri Net $N = (P, T, F, \lambda)$, where P and T are finite disjoint sets of places and transitions of the net, respectively, $F : (P \times T) \cup (T \times P) \to Var$ is a partial function, $\lambda : T \to \mathcal{A} \cup Sync$ is a function labelling transitions, such that for every $t \in T$:*

1. *$\nu \notin pre(t)$,*
2. *If $\lambda(t) \in \mathcal{A}$ then $post(t) \setminus \{\nu\} \subseteq pre(t)$,*
3. *If $\lambda(t) \in Sync$ then $\nu \notin Var(t)$,*

where ${}^\bullet t = \{p \mid (p,t) \in Dom(F)\}$, $t^\bullet = \{p \mid (t,p) \in Dom(F)\}$, $pre(t) = \{F(p,t) \mid p \in {}^\bullet t\}$, $post(t) = \{F(t,p) \mid p \in t^\bullet\}$ and $Var(t) = pre(t) \cup post(t)$.

A ν-net component is a special kind of labelled coloured Petri Net with only one colour type for identifiers, taken from an infinite set Id, except for the special variable ν. Unlike for ordinary Coloured Petri Nets, where arbitrary expressions over some syntax can label arcs, we only allow variables, that are used to specify the flow of tokens from preconditions to postconditions. In particular, this means that only equality of identifiers can be checked by matching.[1] If t is an autonomous transition then it must be the case that every variable annotating a postcondition arc, except ν, also annotates some precondition arc. Then, the only way transitions can produce new names is by means of the variable ν attached to one of its outgoing arcs, which always produces a new name, not present in the current marking of the system.

Definition 7. *A marking of a ν-net $N = (P, T, F, \lambda)$ is a function $M : P \to \mathcal{MS}(Id)$. We denote by $S(M)$ the set of names in M, that is, $S(M) = \bigcup_{p \in P} S(M(p))$.*

Definition 8. *A ν-net system $\mathcal{N}$ is a set of disjoint ν-net components. A marking of $\mathcal{N}$ is a collection of markings of its components, one marking each.*

[1] In fact, analogous results could be obtained if we could also check inequality.

In fact, ν-net systems with a single component correspond to the minimal OO-nets defined in [14]. As for any kind of coloured Petri nets, transitions are fired relative to a mode $\sigma : Var(t) \to Id$, that chooses among the tokens that lie in the precondition places. We will denote modes by $\sigma, \sigma', \sigma_1, \sigma_2, \ldots$

Definition 9. *Let $\mathcal{N}$ be a ν-net system, $t \in T$ with $\lambda(t) \in \mathcal{A}$ and M a marking of $\mathcal{N}$. A transition t is enabled with mode σ if $\sigma(\nu) \notin S(M)$ and for all $p \in {}^\bullet t$, $\sigma(F(p,t)) \in M(p)$. The reached state of $\mathcal{N}$ after the firing of t with mode σ is the marking M' given by $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\} \quad \forall p \in P$.*

Autonomous transitions work mainly as the ordinary transitions in coloured nets (see Fig. 2 left). The only novelty is the presence of the variable ν, which is specially treated in the firing rule: The condition $\sigma(\nu) \notin S(M)$ causes the creation of fresh (equal) identifiers in all the places reached by arcs labelled by that special variable.

For a tuple of synchronizing transitions $\overline{t} = (t_1, \ldots, t_n)$ we denote by $post(\overline{t}) = \bigcup_{i=1}^{n} post(t_i)$, $pre(\overline{t}) = \bigcup_{i=1}^{n} pre(t_i)$ and $Var(\overline{t}) = post(\overline{t}) \cup pre(\overline{t})$.

Definition 10. *Let $\overline{t} = (t_1, \ldots, t_n)$ be a tuple of synchronizing transitions of a ν-net system. We say the transitions in $\overline{t}$ are compatible if:*

1. *$\lambda(t_i) = s(i)$ for some $s \in \mathcal{S}$ with $arity(s) = n$,*
2. *$post(\overline{t}) \setminus \{\nu\} \subseteq pre(\overline{t})$*

The compatibility conditions are still merely syntactical: All the transitions in the tuple must meet together the same constraint imposed to autonomous transitions (see Fig. 2 right). A mode for $\overline{t}$ is a map $\sigma : Var(\overline{t}) \to Id$.

Definition 11. *Let $\mathcal{N}$ be a ν-net system and M a marking of $\mathcal{N}$. We say that the tuple of compatible transitions $\overline{t} = (t_1, \ldots, t_n)$ is enabled with mode σ if for all $p \in {}^\bullet\overline{t}$, $\sum_{i=1}^{n} \{\sigma(F(p,t_i))\} \subseteq M(p)$. The reached state of $\mathcal{N}$ after the firing of $\overline{t}$ with mode σ is the marking M', given by*

$$M'(p) = M(p) - \sum_{i=1}^{n} \{\sigma(F(p,t_i))\} + \sum_{i=1}^{n} \{\sigma(F(t_i,p))\} \quad \forall p \in P$$

By means of synchronization we achieve both name communication and restriction of communications by name matching. If $\overline{t} = (t_1, t_2)$, the former is obtained by using a variable in $post(t_1) \setminus pre(t_1)$ and in $pre(t_2)$ (or vice versa), as the variable z in Fig. 2 right. The latter is obtained by using the same label both in $pre(t_1)$ and $pre(t_2)$, which forces the matching between the corresponding tokens, as the variable x in Fig. 2 right.

In [24] we proved several interesting (un)decidability results for a model we call Mobile Synchronizing Petri Nets, that are essentially these ν-net systems, but again with some syntactic sugar supporting a flat kind of mobility and allowing only two-way synchronizations. In particular, though reachability turns out to be undecidable when adding names, as proved in [14] for minimal OO-nets,

coverability remains decidable, meaning that the expressive power of this model lies somewhere in between ordinary P/T nets and Turing machines. The latter was proved by taking into account the abstract nature of created names. More precisely, we proved that they are well structured systems [11] when we consider the order defined by $M_1 \sqsubseteq_\alpha M_2 \Leftrightarrow$ if there is some M' such that $M_1 \equiv_\alpha M'$ (they are equal up to renaming of identifier tokens) and $M'(p) \subseteq M_2(p) \ \ \forall p \in P$. All these results are trivially transferred to the model considered in this paper.

4 Replication

So far, components were not an essential feature of any of the models just considered. In fact, any basic net system or ν-net system could be flattened to an equivalent one with a single component. However, we have preferred to maintain components in order to have an incremental presentation, and because they are the basis of the agent based architecture of mobile systems that motivated our study. In this section we introduce a replication primitive, that creates an identical copy of the net component that executes it, marked in some fixed way. This primitive makes very desirable the structuring of the system by means of components. Replication, together with synchronization, can be used to implement a spawning primitive, that creates a different component taken from a finite set.

Definition 12. *A Replicated Net (RN) is a labelled Petri net $N = (P, T, F, \lambda)$ where:*

- *P and T are finite disjoint sets of places and transitions, respectively,*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs of the net,*
- *λ is a function from T to the set of labels $\mathcal{A} \cup Sync \cup \mathcal{MS}(P)$.*

A marking M of N is a finite multiset of places of N.

The only syntactical difference with basic nets is that now some transitions may be labelled by a multiset of places, that is, by a marking. That multiset corresponds to the initial marking of the replicated net that is created when firing such a transition.

We represent the presence in the system of several copies of a net by considering several markings of that net, so that now each $N \in \mathcal{N}$ does not represent a single net component, but one of the possible types of nets that can appear in the system. Therefore, unlike for ordinary basic nets, a marking is not just a collection of individual markings, one marking per component, but a multiset of markings, and not necessarily one marking per component, but any natural

Definition 13. *An RN system $\mathcal{N}$ is a set of pairwise disjoint RNs. A marking $\mathcal{M}$ of $\mathcal{N}$ is a finite multiset of markings of its components.*

Alternatively, we could expand the state of a replicated net system, getting an equivalent system with several copies of each net in the initial system, each marked by an ordinary marking. Then, the firing of a replication transition would

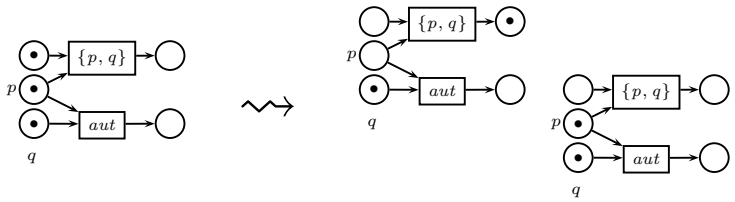

Fig. 3. RN system

add a new component to the system, thus modifying its architecture. However, in order to clearly justify our positive results about the decidability of some properties it is crucial to stress the fact that the set of types of the components of the system remains the same, as explicitly captured by our definitions.

Given an RN system $\mathcal{N}$, marked by $\mathcal{M}$, we will denote by $\mathcal{M}(N)$ the marking of the subsystem of $\mathcal{N}$ composed only of copies of N, so that in order to completely specify a marking $\mathcal{M}$ it is enough to specify each $\mathcal{M}(N)$ for every $N \in \mathcal{N}$. The definitions of enabled transition and firing of transitions are analogous to those for basic net systems. Next, we will present the definitions corresponding to autonomous transitions.

Definition 14. *Given an RN system $\mathcal{N}$ with $\mathcal{N} = \{N_1, \ldots, N_n\}$, $N_i = (P_i, T_i, F_i, \lambda_i)$, and $\mathcal{M}$ a marking of $\mathcal{N}$, $t \in T_i$ with $\lambda(t) \in \mathcal{A}$ is M-enabled if $M \in \mathcal{M}(N_i)$ and ${}^\bullet t \subseteq M$. The reached marking after the M-firing of t is $\mathcal{M}'$, where*

- $\mathcal{M}'(N_i) = \mathcal{M}(N_i) - \{M\} + \{M'\}$, *where* $M' = M - {}^\bullet t + t^\bullet$,
- $\mathcal{M}'(N_j) = \mathcal{M}(N_j)$ *for every* j *with* $i \neq j$.

In the previous definition, the marking of the component firing the transition is replaced in $\mathcal{M}(N)$ by the resulting marking of that firing, where N is the type of that component.

The definition of firing of a tuple of synchronizing transitions is analogous to that in the previous sections, but taking into account that now there may be several components of the same type and that the synchronizing transitions may or may not belong to the same components. To complete the presentation, we define the firing of replication transitions.

Definition 15. *Given an RN system $\mathcal{N}$ with $\mathcal{N} = \{N_1, \ldots, N_n\}$, $N_i = (P_i, T_i, F_i, \lambda_i)$, and $\mathcal{M}$ a marking of $\mathcal{N}$, $t \in T_i$ with $\lambda(t) \in \mathcal{MS}(P_i)$ is M-enabled if $M \in \mathcal{M}(N_i)$ and ${}^\bullet t \subseteq M$. The reached marking after the M-firing of t is $\mathcal{M}'$, where*

- $\mathcal{M}'(N_i) = \mathcal{M}(N_i) - \{M\} + \{M', \lambda(t)\}$, *where* $M' = M - {}^\bullet t + t^\bullet$,
- $\mathcal{M}'(N_j) = \mathcal{M}(N_j)$ *for every* j *with* $i \neq j$.

Therefore, the net firing the replication transition is changed as if it were an autonomous transition; besides, a new net of the same type is created, initially marked by $\lambda(t)$. Fig. 3 illustrates an RN system, initially with one component that can either fire an autonomous transition (labelled by *aut*) or create a replica that can only fire that autonomous transition, and chooses to do the latter.

So far, net components can only be created, but never removed, even though no tokens lie in them, thus making impossible the firing of any of its transitions (supposing, without loss of generality that every transition has at least one precondition). In such a case, we proved in [26] that both reachability and coverability are decidable. The former is proved by taking into account that markings must remember the number of created components, even if some of them are deadlocked. Since this assumption is not very realistic, next we introduce a garbage collection mechanism, that allows us to remove from markings those nets that do not currently have any token, which form a subset of deadlocked components, if we assume that every transition has at least one precondition. We do it simply by disregarding the empty marking as a possible marking of a net, as formalized in the next definition.

Definition 16. *Given two markings $\mathcal{M}$ and $\mathcal{M}'$ of an RN system $\mathcal{N}$ we will write $\mathcal{M} \equiv \mathcal{M}'$ if for all $N \in \mathcal{N}$, $\mathcal{M}(N) \equiv_N \mathcal{M}'(N)$, where $\equiv_N$ is the least equivalence relation on multisets of markings of N such that $M \equiv_N M + \{\emptyset\}$.*

If we identify markings up to $\equiv$, every time a net component becomes empty, we will ignore it. We will write g-RN to denote RN systems with garbage collection. Once again, in [26] we proved the decidability of coverability for g-RN systems by means of a simulation of them using ν-net components.

Proposition 2. *Every g-RN system can be simulated by a ν-net system.*

The proof of the previous result consists on simulating the behaviour of every copy of the same component within the same net, using a different identifier to simulate the ordinary black tokens in each of the copies, thus distinguishing between the different copies. In particular, garbage collection of empty components is obtained for free by means of the disappearance of the identifier that represents it. We guarantee that different components do not incorrectly interact with each other labelling every arc of the autonomous transitions with the same variable, so that they only manipulate tokens of the same type (representing one single net component). Moreover, creation of net components is mapped to creation of new tokens that reflect the initial marking of the new component.

Notice that if we removed every deadlocked component, instead of only the empty ones, this simulation would no longer be correct, unless we could remove from ν-net markings useless identifiers (those that cannot enable any transition). This fact illustrates that in fact the garbage collection for identifiers obtained for free due to their disappearance is not an optimal garbage collection mechanism, but a conservative one.

Therefore, every decidable problem for ν-net systems that is preserved by this simulation is also decidable for g-RN systems, so that coverability is decidable for the latter. Now we prove that the reciprocal is also true, so that undecidability results are also transferred from ν-net systems to g-RN systems.

Proposition 3. *Every ν-net system can be simulated by a g-RN system.*

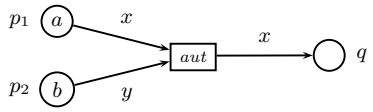

Fig. 4. ν-net without name creation

Proof (sketch). Given a ν-net system N, we have to simulate it using a g-RN system N^*. Without loss of generality, we assume that the ν-net system is just a ν-net without synchronizing transitions, since every ν-net system can be easily flattened to a single equivalent component.

The key idea is to use an RN component for each different name, all of the same type. This component type has the same places as N, so that a black token in a place p corresponding to the component that is simulating the identifier a stands for a token a in the place p of the original net. Fig. 5 shows the simulation of the net in Fig. 4. There, the net in the left simulates the occurrence of a in Fig. 4, and the one in the right does the same for b.

This is a straightforward way to represent the markings of N in N^*. However, the simulation of firings is quite more intricate. Given a transition of N, if every arc that is adjacent to it is labelled with the same variable, then the firing of that transition only involves one token name. If, for instance, that token is a then the RN component representing a can fire an autonomous transition that mimics that firing, moving tokens from the preconditions of t to its postconditions. However, if there is more than one variable adjacent to t then that is not possible. Consider the net in Fig. 4, with two different variables x and y next to its sole transition. It may still be the case that it is fired with x and y instantiated to the same value, so that we need an autonomous transition, labelled by $x = y$ in Fig. 5, that mimics the original transition, as in the previous case. However, t can also be fired taking two different tokens, say a and b, for x and y, respectively. In that case two net components, one representing a and one representing b, should interact by means of a pair of synchronizing transitions to simulate the firing: The one representing a should remove a token from p_1 and put it in q (action $(x \neq y)_x$ in Fig. 5), while the one representing b should just remove a token from its place p_2 (action $(x \neq y)_y$ in Fig. 5). However, since both components must be of the same type, they both must be ready to perform these three actions. That is why in the simulation shown in Fig. 5 we have three transitions, one autonomous transition and two synchronizing transitions.

Regarding name creation, we map it to component creation. Thus, every transition with outgoing arcs labelled by ν are simulated by replicating transitions. The initial marking of the new component is that with a token in every postcondition linked by a ν-arc (see Fig. 6). Notice that we have to remove in the simulation the arcs labelled by ν, since the replicating transition automatically marks those places when fired. This construction works if we assume that transitions that create names do not deal with more than two token names, which in fact can be done without loss of generality (otherwise, RN components should allow synchronizing transitions to create new components).

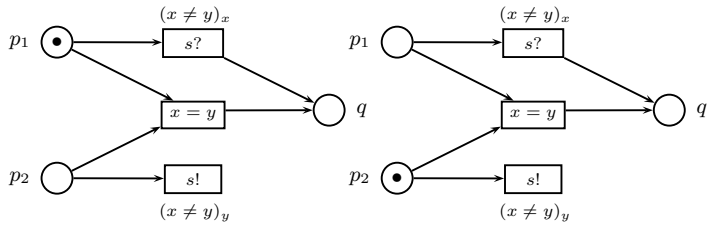

Fig. 5. Simulation of the ν-net in Figure 4 by means of g-RN systems

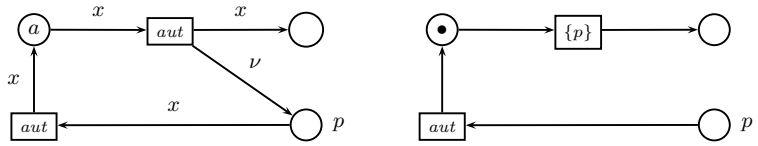

Fig. 6. ν-net with name creation and its RN simulation

The previous construction can be generalized to transitions with an arbitrary number of variables. In the case of two variables, we have implicitly used the two partitions of the set $\{x, y\}$, namely $\{\{x, y\}\}$ and $\{\{x\}, \{y\}\}$. In general, for an arbitrary transition t, we would need to consider any possible partition of the set $Var(t)$. For each element in the partition, if it has n elements then we add n transitions, and label them with labels $s(1), \ldots, s(n)$ for some new $s \in \mathcal{S}$ with arity n, to make them compatible.

Corollary 2. *The reachability problem is undecidable and the coverability problem is decidable for g-RN systems.*

5 Name creation + Replication

Now we consider a model in which we combine the two features in the two previous sections, that is, systems composed of nets that can both create fresh identifiers and replicate themselves. Formally, ν-RN systems are ν-net systems for which we allow an extra transition label type, so that we can label transitions with markings of components, as in the previous section, but now these markings are composed of named tokens.

Quite surprisingly, though the expressive powers obtained by adding either name creation or only replication are identical, as we have proved in the previous section, it turns out that they are somehow orthogonal, so that when combined we reach Turing-completeness, as we will prove next. Informally, we could say that both features generate bidimensional infinite state systems. In the case of ν-net components we may have an unbounded number of different tokens, each of which can appear in markings an unbounded number of times. Analogously, in the case of RN systems, we may have an unbounded number of components, each of them with a possible unbounded number of tokens. However, these two dimensional spaces are not the same, although somehow equivalent to each other,

according to Prop. 2 and Prop. 3. But when we merge the three sources of infinity, only the common dimension overlaps, or following the geometric metaphor, we have two perpendicular planes, that no longer generate a bidimensional space, but the whole tridimensional space. We will see that this space is too rich, thus producing a Turing-complete formalism.

Proposition 4. *Any Turing machine can be simulated by a ν-RN System.*

Proof (sketch). Given a Turing machine, we have to simulate it using a ν-RN System. The main problem to simulate Turing machines is the representation of the potentially one-way infinite set of cells of the tape. We represent the tape by means of a doubly-linked list. Each node of the list (or cell of the tape) will be represented by a different copy of the same process, depicted in Fig. 7. This process has a memory with two kinds of data: First, it must know whether its value is 0 or 1. For that purpose, it has two places named 0 and 1, that are marked in mutual exclusion. Second, and more important, the cell must hold information about which are the previous and next cell whenever it becomes part of the tape of the machine, for what we will use identifiers. For that purpose, components that represent cells have three places (among others), *me*, *prev* and *next*, that contain three different identifiers: the name of the cell in *me*, that of the previous cell in *prev* and that of the next cell in *next*.

At any time, the tape has a final cell, that does know its own name and that of its previous cell (it has been initialized firing its *init*? transition), but not that of the next cell, which in fact does not still exist. In order to expand the tape a new cell can be created by firing the replicating transition $\{i, 0\}$. The new cell can generate its own name, after which it is willing to synchronize with its parent net (the cell at the end of the tape). Then, the synchronization of the transitions *init*! and *init*? causes the exchange of the respective names of the cells, putting them in the corresponding *next* and *prev* places, so that the net in the tape now knows the name of its next cell, and the new cell becomes the last of the tape.

Finally, the cells should also have synchronizing transitions that output the name of the previous and the next cell (transitions *left*! and *right*!, respectively), two others that output its current value (transitions $r(0)$! and $r(1)$!) and finally four more that change it as indicated by the program of the machine (those labelled by $w(0)$? and $w(1)$?). All these transitions are doubly-linked with the place *me*, by arrows labelled by *auth* (those labelled by $w(0)$? and $w(1)$? should also have those arrows, but we have omitted them in the picture for readability).

The part of the construction described so far is the same for any Turing Machine. The rest of the simulation only needs an additional net component for the control, with a place *now* containing the identifier of the cell where the head is pointing. All the synchronizations with the program mentioned in the previous paragraph are done forcing the matching between the value in the place *now* of the program and the value in the place *me* of the cell, by means of the variable *auth*, so that the head can only read and modify the proper cell. Of course, it should also have one place per state of the machine, marked in mutual exclusion,

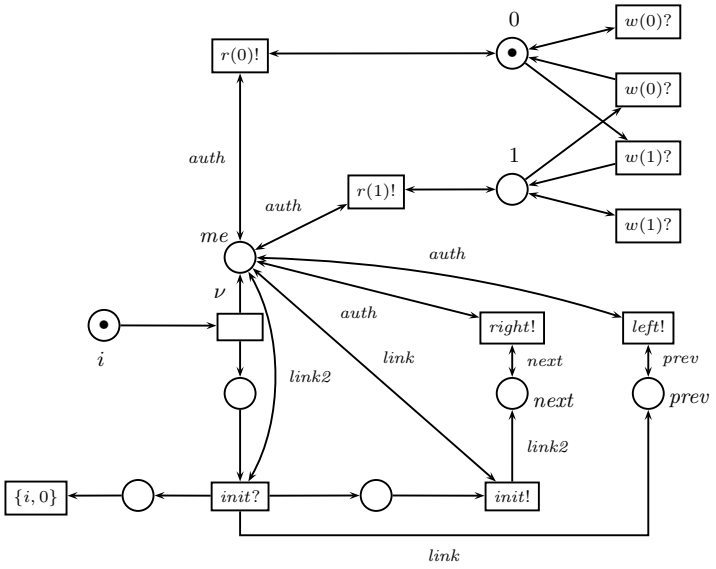

Fig. 7. Turing cell

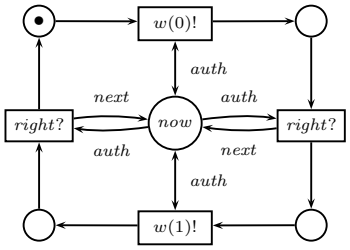

Fig. 8. Turing machine that computes 0101 . . .

each of them with the corresponding actions attached. As an example, Fig. 8 shows the simulation of the head of a Turing machine that computes the infinite sequence 0101 . . .

Notice that the simulation in the proof of the previous result does not make any use of garbage collection, so that Turing completeness is achieved even without it. In fact, we are implicitly considering a kind of garbage collection for identifiers, in the sense that they could disappear from markings after the firing of some transition (as happens for instance to identifier b in Fig. 4 after firing *aut*), thus becoming reusable. We conjecture that if we avoided this automatic garbage collection for names, by including the full set of created names to the state, we would get a situation analogous to that in [9], so that reachability is also decidable. Instead, we can prove the following undecidability result.

Corollary 3. *Coverability and reachability are undecidable for ν-RN systems.*

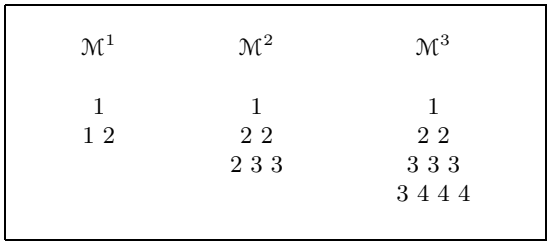

Fig. 9. The natural order of ν-RNs is not a wqo

Proof. It is clear that reachability remains undecidable since it also was undecidable in the less general model of ν-net systems. Coverability is undecidable for ν-RN systems, or we could use the previous simulation to decide the halting problem in Turing machines, just by asking whether the marking with a token in the place representing the final state can be covered.

It is worth pointing out that though the natural orders for both ν-net systems and RN systems are well-quasi orders (wqo) [11], as a consequence of the previous undecidability result, the natural order for ν-RN systems, that extends both, cannot be wqo.

Definition 17. *Given two markings $\mathcal{M}_1$ and $\mathcal{M}_2$ of a ν-RN system $\mathcal{N}$ we define the order $\sqsubseteq$ given by $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ if there are two injections $h_1 : \mathcal{M}_1 \to \mathcal{M}_2$ and $h_2 : S(\mathcal{M}_1) \to S(\mathcal{M}_2)$ such that for every $M \in \mathcal{M}_1$, $h_1(M)(p) \subseteq h_2(M(p))$.*

Therefore, h_1 has the role of mapping components of $\mathcal{M}_1$ to components of $\mathcal{M}_2$, while h_2 has the role of mapping identifiers in $\mathcal{M}_1$ to identifiers in $\mathcal{M}_2$. Indeed, the defined order is not a well-quasi order. To see that, it is enough to consider the simple case of systems with a single net type, with just one place, so that every marking of such a system consists on a multiset of multisets of identifiers. In this case, the previous definition is simplified to $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ if there are two injections $h_1 : \mathcal{M}_1 \to \mathcal{M}_2$ and $h_2 : S(\mathcal{M}_1) \to S(\mathcal{M}_2)$ such that for all $M \in \mathcal{M}_1$, $h_1(M) \subseteq h_2(M)$. Let us take $Id = \mathbb{N}$ and consider the sequence of markings depicted in Fig. 9, $\mathcal{M}^i = \{M^i_1, \ldots, M^i_i, M^i_{i+1}\}$, for $i = 1, 2, \ldots$, where $M^i_k = \{k, \underset{i}{\ldots}, k\}$ for $k = 1, \ldots, i$ and $M^i_{i+1} = \{i, i+1, \underset{i}{\ldots}, i+1\}$, for which there are no indices $i < j$ such that $\mathcal{M}^i \sqsubseteq \mathcal{M}^j$. This is so because for any $i < j$, by cardinality, the only possible choice is to take h_1 and h_2 as $h_1(M^i_k) = M^j_k$ for $k = 1, \ldots, i$, and h_2 the identity, which does not work because $i \in M^i_{i+1}$ but $i \notin M^j_k$ for all $k \geq i + 1$.

6 Boundedness results

Unlike ordinary P/T nets, that have a single infinite dimension, our ν-net systems have two infinite dimensions, since identifiers can appear an unbounded number of times, and there may also be an unbounded amount of different

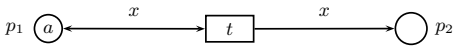

Fig. 10. Width-bounded but not depth-bounded ν-net

Fig. 11. Depth-bounded but not width-bounded ν-net

identifiers. Therefore, we can consider three different notions of boundedness, depending on the dimensions we are considering. In this chapter we will consider ν-net systems with a single component since, as we have said before, every ν-net system can be flattened to an equivalent ν-net.

Definition 18. *Let $N = (P, T, F, M_0)$ be a ν-net.*

1. *N is bounded if there is $n \in \mathbb{N}$ such that for every reachable M and every $p \in P, |M(p)| \leq n$.*
2. *N is depth-bounded if there is $n \in \mathbb{N}$ such that for every reachable M, $a \in Id$ and $p \in P$, $M(p)(a) \leq n$.*
3. *N is width-bounded if there is $n \in \mathbb{N}$ such that for every reachable M, $|S(M)| \leq n$.*

Proposition 5. *A ν-net is bounded if and only if it is depth-bounded and width-bounded.*

As we have said before, in [24] we proved that ν-net systems[2] with order $\sqsubseteq_\alpha$ are well structured systems. Moreover, Prop. 4.5 and Lemma 4.7 prove that, in fact, they are *strictly* well structured (well structured systems with strict monotonicity). In [11] it is proved that for this kind of systems the boundedness notion induced by the considered order is decidable.

Corollary 4. *Boundedness is decidable for ν-nets.*

Therefore, according to the previous result, we can decide if a ν-net is bounded, in which case it is both depth and width-bounded according to Prop. 5. However, if it is not bounded, it could still be the case that it is width-bounded (see Fig. 10) or depth-bounded (see Fig. 11), though not both simultaneously.

Now let us see that width-boundedness is also decidable for ν-nets, for which we can reuse the proof for P/T nets.

Proposition 6. *Let N be a ν-net with initial marking M_0. N is width-unbounded if and only if there are two markings M_1 and M_2 such that:*

- *M_1 is reachable from M_0 and M_2 is reachable from M_1*
- *$M_1 \sqsubseteq_\alpha M_2$*
- *$|S(M_1)| < |S(M_2)|$*

[2] MSPN systems with abstract identifiers, as called there

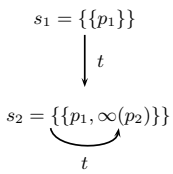

Fig. 12. Karp-Miller tree of the ν-net in Fig. 10

As a consequence, in order to prove that a net is width-unbounded it is enough to find two witness markings like the ones appearing in the result above. These witnesses can be found by constructing a modified version of the Karp-Miller tree that considers markings that are identified up to renaming of identifiers and some of the identifiers can appear infinitely-many times, and by cutting out branches in which the amount of different identifiers strictly grows.

Corollary 5. *The problem of deciding whether a ν-net is width-bounded is decidable.*

We have said that though the coverability problem for ν-net systems is decidable, reachability is undecidable. Of course, if the net is bounded then its space state is finite and, therefore, reachability for this class of ν-nets is decidable. In fact, this is also true for the class of width-bounded ν-nets, even though they can generate infinite state spaces. This is so because it is possible to simulate every width-bounded ν-net by means of a ν-net that does not use the ν-variable, that is, that does not create any identifier, and nets in this subclass are equivalent to P/T nets [23].

Proposition 7. *Every width-bounded ν-net can be simulated by a ν-net without name creation.*

The simulation is based on the fact that, for identifiers that can appear a bounded number of times, we can control when they are removed by explicitly having the nodes of the Karp-Miller tree of the net as places of the simulating net, marked (in mutual exclusion) whenever the ω-marking they represent cover the current marking. By knowing when an identifier is removed, they can be reused, so that at that point we can put them in a repository R of identifiers that can be used whenever a new one was created in the original net. Moreover, we only need to reuse identifiers that can appear a bounded number of times, since they are the only ones that may disappear in all possible executions (even though identifiers that appear an unbounded number of times could also eventually disappear).

As an example, let us consider the ν-net in Fig. 10 (which does not create any identifier already), whose Karp-Miller tree is depicted in Fig. 12. Since we are identifying markings up to renaming of identifiers, we can represent them as the multiset of multisets of places in which identifiers appear, one multiset of places per identifier. Then, the ν-net that results of the simulation sketched above is shown in Fig. 13. Another example is shown in Fig. 14, whose Karp-Miller tree is depicted in Fig. 15, and the simulating ν-net is that in Fig. 16. In it we use

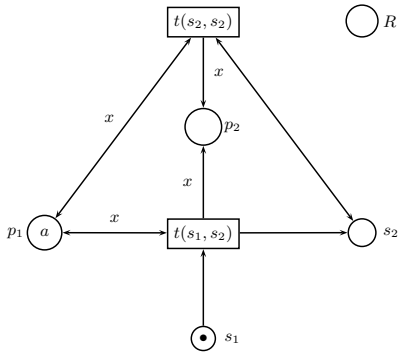

Fig. 13. ν-net without name creation that simulates the ν-net in Fig. 10

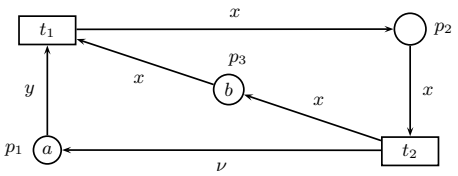

Fig. 14. Another ν-net

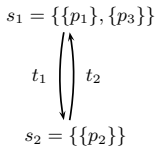

Fig. 15. Karp-Miller tree of the ν-net in Fig. 14

a variable ν', different from ν, that mimics the effect of ν by taking the "new" identifier from the repository.

Corollary 6. *Reachability is decidable for width-bounded ν-nets.*

Proof. Given a marking M of the ν-net N, we construct its Karp-Miller tree, which is finite, and the ν-net without name creation N^* that simulates it. Let n be a natural such that $|S(\overline{M})| \leq n$ for all $\overline{M} \in S$. If $|S(M)| > n$ then M is trivially not reachable. Let us suppose that $|S(M)| \leq n$. In that case, M is reachable in N if and only if there is $\overline{M} \in S$ and a bijection $h : I \to S(M)$ such that:

- $M \sqsubseteq_\alpha \overline{M}$
- M^* is reachable in N^*, where
  - $M^*(p)(a) = M(p)(h(a))$ for all $p \in P$,
  - $M^*(\overline{M}) = 1$ and $M^*(s) = 0$ for all $s \neq \overline{M}$.

Since S is finite and there are finitely-many bijections between I and $S(M)$, reachability in N reduces to reachability in N^*, which is decidable [24].

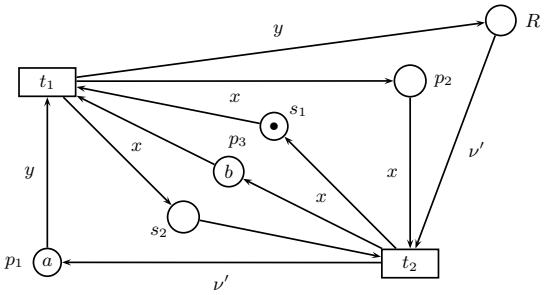

Fig. 16. ν-net without name creation simulating the ν-net in Fig. 14

As final remark, all the boundedness results we have seen in this section regarding identifiers can be translated to the setting with replication, thanks to Prop. 3 and Prop. 2. Therefore, the analogous version of width-boundedness, that we can call component-boundedness is decidable for g-RN systems, and reachability is also decidable for component-bounded g-RN systems.

7 Conclusions and Future Work

We have investigated the consequences of adding two different primitives to Petri Net Systems. The first one is a pure name creation primitive. The model that results from adding it was already studied in [14] and [24], where we proved that its expressive power lies in between that of ordinary Petri Nets and Turing machines. The second primitive, which was presented in [26], deals with component replication. We have proved that these two extensions can simulate each other and thus have the same power. However, when we simultaneously extend the basic model in the two directions, by incorporating both the name creation and the replication primitive, the obtained model turns out to be Turing complete.

There are several directions in which we plan to continue our research. First, we would like to know the relation between our g-RN systems (or equivalently, our ν-net systems) and some well established Petri net formalisms, whose expressive power is also in between Turing machines and Petri nets, such as Petri nets with transfer or reset arcs. We already know that both can be weakly (that is, preserving reachability) simulated using ν-nets, but we do not know if a more faithful simulation that preserves the full behaviour of the net is also possible.

Another interesting issue would be finding some restrictions to the use of the ν variable or that of replicating transitions, so that a model with constrained features but including both primitives would no longer be Turing complete, thus having similar results to those presented in Sect. 6 in the complete model. For instance, it is clear that this is the case if the number of times we can use the replication transitions is globally bounded, or equivalently, whenever we can only create a finite number of names. Certainly, the simulation of Turing machines we have presented would not work if only a bounded quantity of different names

could appear at any reachable marking, even if an arbitrary number of them can be created, which suggests that coverability remains decidable in that case.

It would also be desirable to establish a complete hierarchy of models, including those appearing in [23], and other related models such as [8, 9, 15]. This hierarchy would be based on the quantity of information about the past that a configuration of a system remembers [22]. For instance, the difference between replication with and without garbage collection is that in the former case we force the configuration to remember the number of replicated componentes, which we do not do in the latter.

We are also interested in the practical implementation of the results presented in this paper. In [25] we presented a prototype of a tool for the verification of MSPN systems, which mainly corresponds to our ν-nets here. We plan to extend it to include the replication operator and coverability, boundedness and reachability algorithms, covering the cases in which these properties are decidable.

Finally, let us comment that we have found many interesting connections between the concepts discussed in this paper and the (un)decidability of properties that we have obtained, and those used and obtained in the field of security protocols, like those studied in [10]. In particular, the study of the restrictions needed to preserve the decidability of coverability, could be related with some other recent studies in the field of security protocols [19, 20], while we could also try to use the efficient algorithms [1] that combine forward and backward reachability, to decide security properties of protocols.

References

[1] P.A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. *Forward Reachability Analysis of Timed Petri Nets.* Int. Conf. on Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, FORMATS'04. LNCS vol. 3253, pp. 343-362. Springet, 2004.

[2] T. Ball, S. Chaki, and S.K. Rajamani. *Parameterized Verification of Multithreaded Software Libraries.* 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'01. LNCS vol. 2031, pp. 158–173. Springer, 2001.

[3] A. Bouajjani, J. Esparza, and T. Touili. *A generic approach to the static analysis of concurrent programs with procedures.* 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'03. ACM SIGPLAN 38(1):62-73. ACM, 2003.

[4] A. Bouajjani, J. Esparza, and T. Touili. *Reachability Analysis of Synchronized PA Systems.* 6th Int. Workshop on Verification of Infinite-State Systems, INFINITY'04. ENTCS vol. 138(2), pp. 153-178. Elsevier, 2005.

[5] N. Busi, and G. Zavattaro. *Deciding Reachability in Mobile Ambients.* 14th European Symposium on Programming Languages and Systems, ETAPS'05. LNCS vol. 3444, pp. 248-262. Springer, 2005.

[6] L. Cardelli, and A.D. Gordon. *Mobile Ambients.* 1st Int. Conf. on Foundations of Software Sciences and Computation Structures, FOSSACS'98. LNCS vol. 1387, pp. 140-155. Springer, 1998.

[7] L. Cardelli, G. Ghelli, and A.D. Gordon. *Types for the Ambient Calculus.* Information and Computation 177(2): 160-194 (2002).

[8] G. Delzanno. *An overview of MSR(C): A CLP-based Framework for the Symbolic Verification of Parameterized Concurrent Systems.* 11th Int. Workshop on Functional and Logic Programming, WFLP'02. ENTCS vol. 76. Elsevier, 2002.

[9] G. Delzanno. *Constraint-based Automatic Verification of Abstract Models of Multitreaded Programs.* To appear in the Journal of Theory and Practice of Logic Programming, 2006.

[10] N.A. Durgin, P.D. Lincoln, J.C. Mitchell, and A. Scedrov. *Undecidability of bounded security protocols.* Proc. Workshop on Formal Methods and Security Protocols (FMSP'99).

[11] A.Finkel, and P.Schnoebelen. *Well-Structured Transition Systems Everywhere!* Theoretical Computer Science 256(1-2):63-92 (2001).

[12] D. Frutos-Escrig, O. Marroquín-Alonso and F. Rosa-Velardo. *Ubiquitous Systems and Petri Nets.* Ubiquitous Web Systems and Intelligence, LNCS vol.3841. Springer, 2005.

[13] A. Gordon. *Notes on Nominal Calculi for Security and Mobility.* Foundations of Security Analysis and Design, FOSAD'00. LNCS vol. 2171, pp. 262-330. Springer, 2001.

[14] O. Kummer. *Undecidability in object-oriented Petri nets.* Petri Net Newsletter, 59:18-23, 2000.

[15] R. Lazic. *Decidability of Reachability for Polymorphic Systems with Arrays: A Complete Classification.* ENTCS 138(3): 3-19 (2005).

[16] R. Milner, J. Parrow, and D. Walker. *A Calculus of Mobile Processes, I.* Information and Computation 100(1): 1-40 (1992).

[17] F. Nielson, R.R. Hansen, and H.R. Nielson. *Abstract interpretation of mobile ambients.* Sci. Comput. Program. 47(2-3): 145-175 (2003).

[18] G. Ramalingam. *Context-sensitive synchronization-sensitive analysis is undecidable.* ACM Trans. Program. Lang. Syst. 22(2): 416-430 (2000).

[19] R.Ramanujam, and S.P.Suresh. *Decidability of context-explicit security protocols.* Journal of Computer Security, vol 13, number 1, 2005, pages 135-165.

[20] R.Ramanujam, and S.P.Suresh. *Tagging makes secrecy decidable with unbounded nonces as well.* 23rd Int. Conf. Foundations of Software Technology and Theoretical Computer Science, FSTTCS'03, LNCS vol. 2914, pp.363-374. Springer, 2003.

[21] M. Rinard. *Analysis of multithreaded programs.* 8th Static Analysis Symposium, SAS'01. LNCS 2126, pp. 1–19. Springer, 2001.

[22] F. Rosa Velardo, C. Segura Díaz and David de Frutos Escrig. *Tagged systems: a framework for the specification of history dependent properties.* Fourth Spanish Conference on Programming and Computer Languages, PROLE'04. ENTCS vol. 137, Issue 1, 2005.

[23] F. Rosa-Velardo, D. Frutos-Escrig, and O. Marroquín-Alonso. *Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems.* 1st Int. Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, MTCoord'05. ENTCS vol.150(1). Elsevier, 2006.

[24] F. Rosa-Velardo, D. Frutos-Escrig, and O. Marroquín-Alonso. *On the expressiveness of Mobile Synchronizing Petri Nets.* 3rd Int. Workshop on Security Issues in Concurrency, SecCo'05. ENTCS (to appear).

[25] F. Rosa-Velardo. *Coding Mobile Synchronizing Petri Nets into Rewriting Logic.* 7th Int. Workshop on Rule-based Programming, RULE'06. ENTCS (to appear).

[26] F. Rosa-Velardo, D. Frutos-Escrig, O. Marroquín-Alonso. *Replicated Ubiquitous Nets.* Ubiquitous Web Systems and Intelligence, LNCS vol. 3983. Springer, 2006.

[27] P. Zimmer. *On the Expressiveness of Pure Mobile Ambients.* 7th Int. Workshop on Expressiveness in Concurrency, EXPRESS'00, ENTCS vol. 39(1). Elsevier, 2003.