Complexity of LTL Model-Checking for Safe Object Nets

Michael Köhler-Bußmeier and Frank Heitmann
University of Hamburg, Department for Informatics
Vogt-Kölln-Straße 30, D-22527 Hamburg
koehler,heitmann@informatik.uni-hamburg.de

Abstract

In this paper we present recently obtained results from [14] concerning the complexity of LTL model checking of safe Elementary Object Nets (Eos) in a novel and more algorithmic oriented way.

Object nets are Petri nets which have Petri nets as tokens – an approach known as the nets-within-nets paradigm. Object nets are called elementary if the net system has a two levelled structure. Due to these two modelling levels object nets are very suited to model the mobility of e.g. active objects or agents. The well known p/t nets can be viewed as a special case of Eos.

For p/t nets the concept of safeness means that there is at most one token on each place. Since object nets have nested markings there are different possibilities to generalise this idea for Eos. In this paper we concentrate on the variant of Eos safeness that guarantees the finiteness of state spaces and show that for safe Eos the LTL model checking problem is PSpace-complete.

1 Model Checking for Object Nets

In the following we investigate the analysis of object nets using temporal logics. Object nets are Petri nets which have Petri nets as tokens – an approach which is called the *nets-within-nets* paradigm, proposed by Valk [22, 24] for a two levelled structure and generalised in [11, 12] for arbitrary nesting structures.[1] The Petri nets that are used as tokens are called net-tokens. Net-tokens are tokens with internal structure and inner activity. Due to these two modelling levels object nets are very suited to model the mobility of active objects or agents (cf. [9] and [10]).

In [12, 8] we studied decidability properties of unbounded object nets. In this paper we repeat very recent results from [14] concerning *bounded* object nets and present them in a novel way. In particular we repeat the formalism of (safe) elementary object nets and sharpen the definitions in some points. We then prove that checking if a safe Eos satisfies a property expressed in LTL is PSpace-complete. This was first proved in [14]. We recapitulate the proof here, but take a more algorithmic oriented point of view and thus make the proof more elegant and easier to follow in several places. Throughout this paper we stress the relevance of the presented modelling formalism for the modelling of mobile objects and agents.

The paper has the following structure: In Section 2 elementary object systems (Eos) are defined. In section 3 safe Eos are defined and in section 4 we discuss the complexity of LTL model checking these nets. The paper ends with a conclusion.

In the following we assume basic knowledge of Petri nets, see e.g. [20].

[1] Related approaches dealing with nesting and Petri nets are object nets [23], recursive nets [4], nested nets [17], PN^2 [5], MOB nets [15], hypernets [1], Mobile Systems [16], AHO systems [6], adaptive workflow nets [18], and Hornets [13].

2 Elementary Object Systems

An elementary object system (Eos) is composed of a system net, which is a p/t net $\widehat{N} = (\widehat{P}, \widehat{T}, \mathbf{pre}, \mathbf{post})$ and a set of object nets $\mathcal{N} = \{N_1, \ldots, N_n\}$, which are p/t nets given as $N = (P_N, T_N, \mathbf{pre}_N, \mathbf{post}_N)$. We assume that all sets of nodes (places and transitions) are pairwise disjoint. Moreover we have $\widehat{N} \notin \mathcal{N}$ and we assume the existence of the object net $\bullet \in \mathcal{N}$ which has no places and no transitions and is used to model anonymous, so called black tokens.

Figure 1 shows an example, which will be explained in detail in example 2.2 and 2.4 below. One may imagine the object nets (the nets which reside on places of the system net) to model mobile agents with internal structure and activity, while the system net models the environment in which the agents act and interact.

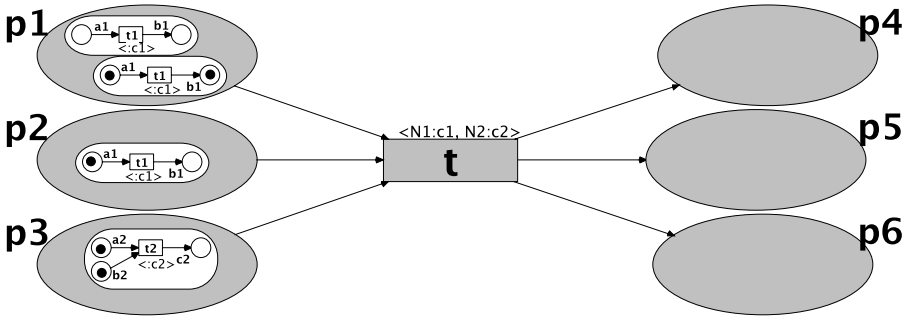

Figure 1: An object net

The system net places are typed by the mapping $d : \widehat{P} \to \mathcal{N}$ with the meaning, that the place $\widehat{p}$ of the system net contains net-tokens of the object net type N if $d(\widehat{p}) = N$.[2] No place of the system net is mapped to the system net itself, since $\widehat{N} \notin \mathcal{N}$.

Since the tokens of an Eos are instances of object nets a *marking* $\mu \in \mathcal{M}$ of an Eos OS is a *nested* multiset. A marking of an Eos OS is denoted $\mu = \sum_{k=1}^{|\mu|}(\widehat{p}_k, M_k)$ where $\widehat{p}_k$ is a place in the system net and M_k is the marking of the net-token of type $d(\widehat{p}_k)$. To emphasise the nesting, markings are also denoted as $\mu = \sum_{k=1}^{|\mu|} \widehat{p}_k[M_k]$. Tokens of the form $\widehat{p}[\mathbf{0}]$ and $d(\widehat{p}) = \bullet$ are abbreviated as $\widehat{p}[]$.

The set of all markings which are syntactically consistent with the typing d is denoted $\mathcal{M}$ (Here $d^{-1}(N) \subseteq \widehat{P}$ is the set of system net places of the type N):

$$\mathcal{M} := MS\Big(\bigcup_{N \in \mathcal{N}} \big(d^{-1}(N) \times MS(P_N) \big) \Big) \tag{1}$$

The transitions in an Eos are labelled with synchronisation channels. We assume a fixed set of channels C, including the channel ϵ which is used to describe the absence of any "real" channel. Each transition of the system net has one label for each object, defined by the labelling function function $\widehat{l} : \widehat{T} \to (N \to C)$. Each transition of an object net N has one single label,

[2] In the following the terms *(marked) object net* and *net-token* are used almost interchangeable. We use the term *net-token* whenever we like to emphasise the aspect that the marked object net is a token of the system net.

defined by the labelling function function $l_N : T_N \to C$.[3]

A *system event* is generated by transitions with matching labels. The labelling allows three cases of events:

1. System-autonomous firing: Whenever $\widehat{l}(\widehat{t})(N) = \epsilon$ holds for all $N \in \mathcal{N}$, the system net transition $\widehat{t}$ fires autonomously.
2. Synchronised firing: There is at least one object net that has to be synchronised, i.e. there is a N such that $\widehat{l}(\widehat{t})(N) \neq \epsilon$.
3. Object-autonomous firing: An object net N's transition t fires autonomously whenever $l_N(t) = \epsilon$.

These three kinds of events can be reduced to the case of a synchronisation where a system net transition has exactly one synchronisation partner in each object net. This *normal form* is obtained by adding some idle transitions: For each object net $N \in \mathcal{N}$ we add the idle-transitions ϵ_N with $\mathbf{pre}_N(\epsilon_N) = \mathbf{post}_N(\epsilon_N) = \mathbf{0}$ to its transition set. For object-autonomous events we also add the set of idle transitions $\epsilon_{\widehat{P}} := \{\epsilon_{\widehat{p}} \mid \widehat{p} \in \widehat{P}\}$ with $\mathbf{pre}(\epsilon_{\widehat{p}}) = \mathbf{post}(\epsilon_{\widehat{p}}) = \widehat{p}$ to the set of system net transitions. We extend the labelling to idle transitions by $\widehat{l}(\epsilon_{\widehat{p}})(N) = l_N(\epsilon_N) = \epsilon$ for all $\widehat{p} \in \widehat{P}$ and $N \in \mathcal{N}$.

With these idle transitions the channel ϵ (which means "no synchronisation") is modelled as a synchronisation with an idle transition that has no effect.

The synchronisation labelling generates the set of system events Θ. An event is a pair – denoted $\widehat{\tau}[\theta]$ in the following. Here, $\widehat{\tau}$ is either a real transition $\widehat{t}$ or $\epsilon_{\widehat{p}}$ for some $\widehat{p}$; θ maps each object net to one of its transitions. An event has the meaning that the system net transition $\widehat{\tau}$ fires synchronously with all the object net transitions $\theta(N), N \in \mathcal{N}$.

A special case for the mapping θ is the idle map $\epsilon_{\mathcal{N}}$ which is defined $\epsilon_{\mathcal{N}}(N) = \epsilon_N$ for all $N \in \mathcal{N}$.

All events are generated from the labels: $\widehat{l}(\widehat{\tau})(N) = l_N(\theta(N))$ must hold for all $N \in \mathcal{N}$. Whenever $\widehat{\tau}$ is an idle transition $\epsilon_{\widehat{p}} \in \epsilon_{\widehat{P}}$ we also demand that $\theta(N)$ is the idle event ϵ_N except for exactly one object net N (which is the object-autonomous event), i.e. $|\{N \in \mathcal{N} : \theta(N) \neq \epsilon_N\}| = 1$ holds.

$$\Theta_l := \Big\{\widehat{\tau}[\theta] \mid \forall N \in \mathcal{N} : \widehat{l}(\widehat{t})(N) = l_N(\theta(N)) \wedge \widehat{\tau} \in \epsilon_{\widehat{P}} \Longrightarrow |\{N \in \mathcal{N} : \theta(N) \neq \epsilon_N\}| = 1\Big\} \tag{2}$$

Definition 2.1 (EOS)**.** *An elementary object system (*Eos*) is a tuple* $OS = (\widehat{N}, \mathcal{N}, d, \Theta_l)$ *such that:*

1. $\widehat{N}$ *is a p/t net, called the* system net*.*
2. $\mathcal{N}$ *is a finite set of disjoint p/t nets, called* object nets*.*
3. $d : \widehat{P} \to \mathcal{N}$ *is the typing of the system net places.*
4. Θ_l *is the set of events generated from the labelling* $l = (\widehat{l}, (l_N)_{N \in \mathcal{N}})$*.*

An Eos *with initial marking is a tuple* $OS = (\widehat{N}, \mathcal{N}, d, \Theta_l, \mu_0)$ *where* $\mu_0 \in \mathcal{M}$ *is the initial marking.*

[3] In the graphical representation the synchronisation labelling is defined by transition inscriptions in the form $\langle N_1 : \widehat{l}(\widehat{t})(N_1), \ldots\rangle$ in the system net and in the form $\langle : l_N(t)\rangle$ in the object nets (ϵ is omitted).

Example 2.2. *Figure 1 shows an* Eos *with the system net* $\widehat{N}$ *and the object nets* $\mathcal{N} = \{N_1, N_2\}$. *The system net is given as* $\widehat{N} = (\widehat{P}, \widehat{T}, \mathbf{pre}, \mathbf{post})$ *with* $\widehat{P} = \{p_1, \ldots, p_6\}$ *and* $\widehat{T} = \{t\}$. *The first object net is* $N_1 = (P_1, T_1, \mathbf{pre}_1, \mathbf{post}_1)$ *with* $P_1 = \{a_1, b_1\}$ *and* $T_1 = \{t_1\}$. *The second object net is* $N_2 = (P_2, T_2, \mathbf{pre}_2, \mathbf{post}_2)$ *with* $P_2 = \{a_2, b_2, c_2\}$ *and* $T_2 = \{t_2\}$. *The typing is* $d(p_1) = d(p_2) = d(p_4) = N_1$ *and* $d(p_3) = d(p_5) = d(p_6) = N_2$. *The labelling function of the system net* $\widehat{l}$ *is defined by* $\widehat{l}(t)(N_1) = c_1$ *and* $\widehat{l}(t)(N_2) = c_2$. *The labelling* l_{N_1} *of the first object net is defined by setting* $l_{N_1}(t_1) = c_1$. *Similarily,* l_{N_2} *is defined by* $l_{N_2}(t_2) = c_2$. *There is only one synchronous event:* $\Theta_l = \{t[N_1 \mapsto t_1, N_2 \mapsto t_2]\}$. *The initial marking has two net-tokens on* p_1, *one on* p_2, *and one on* p_3*:*

$$\mu = p_1[a_1 + b_1] + p_1[\mathbf{0}] + p_2[a_1] + p_3[a_2 + b_2]$$

Note that in figure 1 the structure of the three net-tokens is the same on p_1 *and* p_2 *but the net-tokens' markings are different. Note also that the object nets may be viewed as models of mobile agents with internal activity. The system net then models the environment.*[4] *In example 2.5 below a mobile agent, which resides in a slightly more sophisticated environment, is presented.* ◇

We name special properties of Eos: An Eos is *p/t-like* iff it has only places for black tokens: $d(\widehat{P}) = \{\bullet\}$. An Eos is a *generalised state machine* (GSM) iff for all $\widehat{t}$ there is either exactly one place in the preset and one in the postset typed with the same object net N or there are no such places:

$$\forall N \in \mathcal{N} : \forall \widehat{t} \in \widehat{T} : \left|\{\widehat{p} \in {}^\bullet\widehat{t} \mid d(\widehat{p}) = N\}\right| = \left|\{\widehat{p} \in \widehat{t}^\bullet \mid d(\widehat{p}) = N\}\right| \leq 1 \tag{3}$$

and the inital marking has at most one net-token of each type:

$$\forall N \in \mathcal{N} : \sum\nolimits_{\widehat{p} \in \widehat{P}, d(\widehat{p}) = N} \Pi^1(\mu_0)(\widehat{p}) \leq 1 \tag{4}$$

Obviously each p/t-like Eos is a GSM since $d(\widehat{p}) = \bullet$ for all $\widehat{p}$.

2.1 Projections and Firing Rule

Firing a system event $\widehat{\tau}[\theta] \in \Theta_l$ removes net-tokens together with their individual internal markings. A nested multiset $\lambda \in \mathcal{M}$ that is part of the current marking μ, i.e. $\lambda \leq \mu$, is replaced by a nested multiset ρ.

Let $\mu = \sum_{k=1}^{|\mu|} (\widehat{p}_k, M_k)$ be a marking of an Eos. To formalize the firing rule we need to introduce projections. The projection Π^1 on the first component abstracts away the substructure of all net-tokens:

$$\Pi^1\left(\sum\nolimits_{k=1}^{|\mu|} \widehat{p}_k[M_k]\right) := \sum\nolimits_{k=1}^{|\mu|} \widehat{p}_k \tag{5}$$

The projection Π^2_N on the second component is the abstract marking of all net-tokens of the type $N \in \mathcal{N}$ ignoring their local distribution within the system net:

$$\Pi^2_N\left(\sum\nolimits_{k=1}^{|\mu|} \widehat{p}_k[M_k]\right) := \sum\nolimits_{k=1}^{|\mu|} \mathbf{1}_N(\widehat{p}_k) \cdot M_k \tag{6}$$

[4] Since we only have one system net transition, the system net in figure 1 should be viewed as an excerpt of a larger system net only.

where the indicator function $\mathbf{1}_N : \widehat{P} \to \{0,1\}$ is given by $\mathbf{1}_N(\widehat{p}) = 1$ iff $d(\widehat{p}) = N$. Note that $\Pi^2_N(\mu)$ results in an marking of the object net N.

The enabling condition is expressed by the *enabling predicate* ϕ_{OS} (or just ϕ whenever OS is clear from the context):

$$\begin{array}{ll} \phi(\widehat{\tau}[\theta], \lambda, \rho) & \iff \Pi^1(\lambda) = \mathbf{pre}(\widehat{\tau}) \wedge \Pi^1(\rho) = \mathbf{post}(\widehat{\tau}) \wedge \\ & \forall N \in \mathcal{N} : \Pi^2_N(\lambda) \geq \mathbf{pre}_N(\theta(N)) \wedge \\ & \forall N \in \mathcal{N} : \Pi^2_N(\rho) = \Pi^2_N(\lambda) - \mathbf{pre}_N(\theta(N)) + \mathbf{post}_N(\theta(N)) \end{array} \tag{7}$$

With $\widehat{M} = \Pi^1(\lambda)$ and $\widehat{M'} = \Pi^1(\rho)$ as well as $M_N = \Pi^2_N(\lambda)$ and $M'_N = \Pi^2_N(\rho)$ for all $N \in \mathcal{N}$ the predicate ϕ has the following meaning:

1. The first conjunct expresses that the system net multiset $\widehat{M}$ corresponds to the pre-condition of the system net transition $\widehat{\tau}$, i.e. $\widehat{M} = \mathbf{pre}(\widehat{\tau})$.

2. In turn, a multiset $\widehat{M'}$ is produced, that corresponds to the post-set of $\widehat{\tau}$.

3. An object net transition t_N is enabled if the combination M_N of the markings of net-tokens of type N enables it, i.e. $M_N \geq \mathbf{pre}_N(\theta(N))$.

4. The firing of $\widehat{\tau}[\theta]$ must also obey the object marking distribution condition

$$M'_N = M_N - \mathbf{pre}_N(\theta(N)) + \mathbf{post}_N(\theta(N))$$

   where $\mathbf{post}_N(\theta(N)) - \mathbf{pre}_N(\theta(N))$ is the effect of the object net's transition on the net-tokens.

Note that (1) and (2) assures that only net-tokens relevant for the firing are included in λ and ρ. Conditions (3) and (4) allows for additonal tokens in the net-tokens.

For system-autonomous events $\widehat{\tau}[\epsilon_{\mathcal{N}}]$ the enabling predicate ϕ can be simplified further. We have $\mathbf{pre}_N(\epsilon_N) = \mathbf{post}_N(\epsilon_N) = \mathbf{0}$. This ensures $\Pi^2_N(\lambda) = \Pi^2_N(\rho)$, i.e. the sum of markings in the copies of a net-token is preserved w.r.t. each type N. This condition ensures the existence of linear invariance properties (cf. [12], Prop. 7).

Analogously, for an object-autonomous event we have an idle-transition $\widehat{\tau} = \epsilon_{\widehat{p}}$ for the system net and the first and the second conjunct is: $\Pi^1(\lambda) = \mathbf{pre}(\widehat{t}) = \widehat{p} = \mathbf{post}(\widehat{t}) = \Pi^1(\rho)$. So, there is an addend $\lambda = \widehat{p}[M]$ in μ with $d(\widehat{p}) = N$ and M enables $t_N := \theta(N)$.

Definition 2.3 (Firing Rule)**.** *Let OS be an* Eos *and $\mu, \mu' \in \mathcal{M}$ markings. The event $\widehat{\tau}[\theta]$ is enabled in μ for the mode $(\lambda, \rho) \in \mathcal{M}^2$ iff $\lambda \leq \mu \wedge \phi(\widehat{\tau}[\theta], \lambda, \rho)$ holds.*

An event $\widehat{\tau}[\theta]$ that is enabled in μ for the mode (λ, ρ) can fire: $\mu \xrightarrow[OS]{\widehat{\tau}[\theta](\lambda,\rho)} \mu'$. The resulting successor marking is defined as $\mu' = \mu - \lambda + \rho$.

We write $\mu \xrightarrow[OS]{\widehat{\tau}[\theta]} \mu'$ whenever $\mu \xrightarrow[OS]{\widehat{\tau}[\theta](\lambda,\rho)} \mu'$ for some mode (λ, ρ).

Example 2.4. *Consider the* Eos *of figure 1 again. The current marking μ of the* Eos *enables $t[N_1 \mapsto t_1, N_2 \mapsto t_2]$ in the mode (λ, ρ) where*

$$\begin{array}{rcl} \lambda & = & p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2] \\ \rho & = & p_4[a_1 + b_1 + b_1] + p_5[\mathbf{0}] + p_6[c_2] \end{array}$$

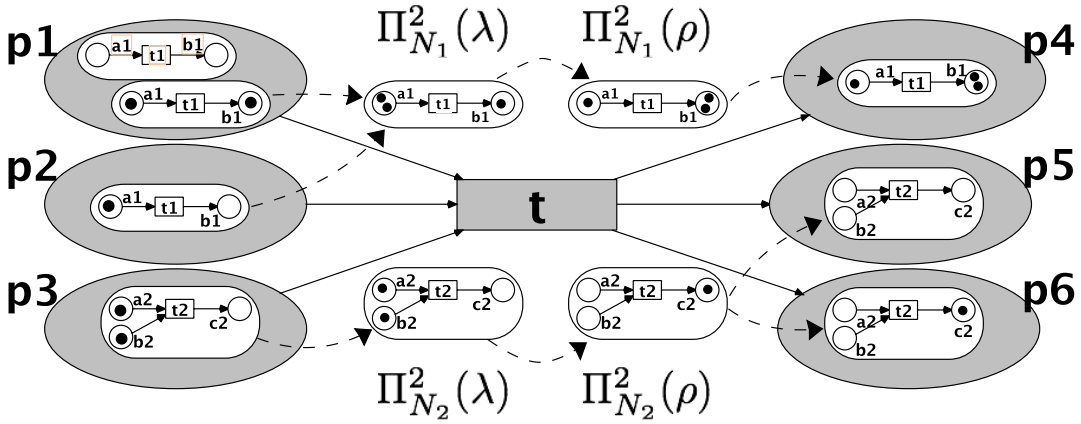

Figure 2: The EOS of figure 1 illustrating the projections $\Pi^2_N(\lambda)$ and $\Pi^2_N(\rho)$

We have the current marking:

$$\mu \quad = \quad p_1[\mathbf{0}] + \underbrace{p_1[a_1 + b_1] + p_2[a_1] + p_3[a_2 + b_2]}_{\lambda} = p_1[\mathbf{0}] + \lambda$$

The net-tokens' markings are added by the projections Π^2_N, resulting in the markings $\Pi^2_N(\lambda)$. The sub-synchronisation generate $\Pi^2_N(\rho)$. (The results are shown above and below the transition t.) After the synchronisation we obtain the successor marking on p_4, p_5, and p_6 as shown in figure 2:

$$\begin{aligned} \mu' &= (\mu - \lambda) + \rho = p_1[\mathbf{0}] + \rho \\ &= p_1[\mathbf{0}] + p_4[a_1 + b_1 + b_1] + p_5[\mathbf{0}] + p_6[c_2] \end{aligned}$$

Note that a different successor marking is possible: $\rho' = p_4[a_1 + b_1 + b_1] + p_5[c_2] + p_6[\mathbf{0}]$ would also work, because we may distribute one object net's tokens arbitrarily between all object nets of this same type. Also note that (λ, ρ) is not the only mode that enables the above event. If we take the other (empty) object net on place p_1, we would still be able to fire. The transition t_1 in the object net is still enabled due to the single token from the object net on p_2.

Again interpreting the object nets as mobile agents and the system net as the environment, the firing of transition t synchronously with transition t_1 of object net N_1 and t_2 of object net N_2, may be viewed as an interaction of the two agents represented by the different object nets. A object-autonomous event may then be viewed as an independent action of an agent, while a system-autonomous event may either be viewed as something that happens in the environment or as a movement of an agent. A synchronous event may be an interaction between several agents as above, or it might be an interaction of one or several agents with the environment. ⋄

Example 2.5. *We now consider a mobile agent which moves from its home to the office where it can execute two different tasks in parallel before he leaves for the city. The example system shown in figure 3 can be expressed in rewriting logic [19]. The following specification is made in* Maude *syntax. All operators for the system level are prefixed by "sn" and "on" for the object net.*

```
mod ONS is
```

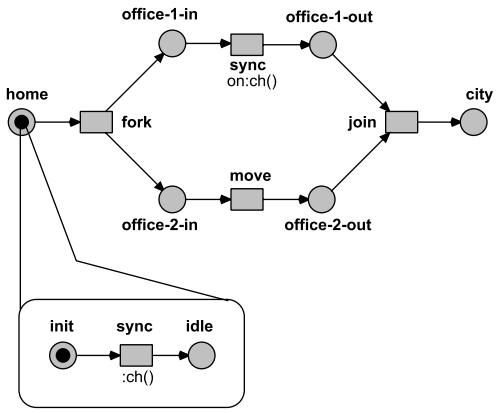

Figure 3: A simple object net system

```
  ***  Nested Markings ***
sort Token-sys .

sorts Token-sn  Marking-sn .
subsort Token-sn <   Marking-sn .
op m-nil : -> Marking-sn .
op __ : Marking-sn  Marking-sn -> Marking-sn [assoc comm id: m-nil] .

sorts Token-on  Marking-on .
subsort Token-on <   Marking-on .
op m-nil : -> Marking-on .
op __ : Marking-on  Marking-on -> Marking-on [assoc comm id: m-nil] .
var M M1 M2 : Marking-on .

  *** Net Tokens ***
sort ON .
op on : Marking-on -> ON .

sort SN .
op sn : Marking-sn -> SN .

  *** Places and Channels  ***
op  s0 : SN -> Token-sys .

ops sn-home  sn-office-in-1  sn-office-in-2
     sn-office-out-1  sn-office-out-2  sn-city
     sn-uptown  sn-downtown :  ON ->  Token-sn .

ops  on-init  on-at-office  on-idle : ->  Token-on .
ops  ch?! ch!? : -> Token-on .

  *** Initial Marking  ***
op Init-Marking : -> Token-sys  .
eq Init-Marking =  s0(sn(sn-home(on(on-init)))) .
```

```
    *** Transitions ***
  rl  [sn-fork]: sn-home(on(M1 M2)) =>
    sn-office-in-1(on(M1)) sn-office-in-2(on(M2))  .

  crl  [sn-sync]: sn-office-in-1(on(M1)) => sn-office-out-1(on(M2))
       if ch?!  M1 => ch!?  M2 .

  rl  [on-sync]:  ch?! on-init =>  ch!? on-idle .

  rl  [sn-move]: sn-office-in-2(on(M)) => sn-office-out-2(on(M)) .

  rl  [sn-join]: sn-office-out-1(on(M1)) sn-office-out-2(on(M2))
       => sn-city(on(M1 M2)) .

  endm
```

This specification allows a rewriting sequence (i.e. firing sequence) resulting in the successor marking `s0(sn(sn-city(on(on-idle))))`*:*

```
Maude> rew Init-Marking .
  rewrites: 6 in 0ms cpu (5ms real) (~ rewrites/second)
  result Token-sys: s0(sn(sn-city(on(on-idle))))
Maude>
```

MAUDE *provides a generic LTL model checker [2]. The user only has to provide the basic properties. For our example we define two properties:* `office2out` *and* `terminated`*. The first proposition is true whenever the place* sn-office-out-2 *is marked and the second when the place* sn-city *is marked.*

```
var X : Marking-sn .
var M : Marking-on .
ops office2out  terminated  :  -> Prop .
eq (s0(sn(X sn-office-out-2(on(M))))  |=  office2out) = true .
eq (s0(sn(sn-city(on(M))))  |=  terminated) = true .

ops phi1 phi2 : -> Prop .
eq phi1 = <> office2out .
eq phi2 = <> terminated .
```

In this specification we have the two LTL formulae ϕ_1 *and* ϕ_2*. The formula* `phi1 = <> office2out` *is the representation of* $\phi_1 = \Diamond$*office2out, which expresses that the mobile agent will sometimes be at the place* `sn-office-out-2`*. The satisfiability of the logical atoms is given as equations:*

```
eq (s0(sn(X sn-office-out-2(on(M)))) |= office2out) = true
```

Model-checking the state spaces reveals the correctness of ϕ_1*:*

```
  Maude> rew Init-Marking |= phi1 .
  rewrite in CHECK : Init-Marking |= phi1 .
  rewrites: 16 in 20ms cpu (32ms real) (800 rewrites/second)
  result Bool: true
```

The formula `phi2 = <> terminated` *(for* $\phi_2 = \Diamond$*terminated) describes, together with the equation*

`eq (s0(sn(sn-city(on(M)))) |= terminated) = true`,

that the place `sn-city` *will eventually be occupied by the mobile agent. It is maybe not obvious at first sight that there is one trace which refutes the property: When firing the transition* `fork` *the token on* `on-init` *might not be assigned to the object net on place* `sn-office-1-in` *and therefore the synchronisation is disable. The model checking processes reveals this counter example:*

```
Maude> rew Init-Marking |=  phi2 .
rewrite in CHECK : Init-Marking |= phi2 .
rewrites: 23 in 10ms cpu (28ms real) (2300 rewrites/second)
result ModelCheckResult: counterexample(
 {s0(sn(sn-city(on(on-idle)))),'sn-fork}
 {s0(sn(sn-office-in-1(on(m-nil))
        sn-office-out-2(on(on-init)))),'sn-move},
 {s0(sn(sn-office-in-1(on(m-nil))
        sn-office-out-2(on(on-init)))),deadlock})
```

This example illustrates the potential of LTL model checking in the context of mobile agents modelled with object nets. ◇

3 Boundedness and Safe Eos

Boundedness is the problem to decide whether there are only finitely many reachable markings. A p/t net is called n-safe with $n \in \mathbb{N}$ if in every reachable marking there are at most n tokens on each place: $\forall m \in RS(\mathbf{m}_0) : \forall p \in P : m(p) \leq n$. A net that is n-safe for some n is also called bounded. The following property is well-known for p/t nets.

Lemma 3.1. *The set of reachable markings of a p/t net N is finite iff N is n-safe for some n.*

Proof. If N is n-safe then $|RS(\mathbf{m}_0)| \leq (n+1)^{|P|}$. If the set of reachable markings is finite then N is n-safe for $n := \max\{\mathbf{m}(p) \mid p \in P, \mathbf{m} \in RS(\mathbf{m}_0)\}$. □

A p/t net is called *safe* if it is 1-safe. Therefore, each reachable marking of a safe net is a set and we have $|RS(\mathbf{m}_0)| \leq 2^{|P|}$, i.e. the number of subsets of P.

In the following we consider bounded object nets. Reachability is decidable whenever the object nets are bounded. An Eos is *N-bounded* if $\Pi^2_N(\mu)$ is bounded in each reachable marking μ. An Eos is *semi-bounded* if it is bounded for all object nets $N \in \mathcal{N}$. The reachability problem is decidable for a semi-bounded Eos ([8], Thm. 3.4).

Note that it is possible that an Eos is N-bounded but N (considered as a p/t net in isolation) is not bounded for the initial marking $\Pi^2_N(\mu_0)$. Any unbounded object net where each transition is synchronised with a dead system net transition is an example.

In [14] we identified four different variants of safeness – named safe(1), safe(2), safe(3), and safe(4) – where safe($i+1$) implies safe(i), but not vice versa. On p/t-like Eos all these variants coincide.

Definition 3.2. *Let OS be an* Eos.

- *OS is safe(1) iff all reachable markings are sets.*

$$\forall \mu \in RS(OS) : \forall \widehat{p}[M] : \mu(\widehat{p}[M]) \leq 1$$

- *OS is safe(2) iff for all reachable markings there is most one token on each system net place:*
$$\forall \mu \in RS(OS): \quad \forall \widehat{p} \in \widehat{P} : \Pi^1(\mu)(\widehat{p}) \leq 1$$
- *OS is safe(3) iff for all reachable markings there is most one token on each system net place and each net-token is safe:*
$$\begin{aligned} \forall \mu \in RS(OS): \quad & \forall \widehat{p} \in \widehat{P} : \Pi^1(\mu)(\widehat{p}) \leq 1 \wedge \\ & \forall N \in \mathcal{N} : \forall p \in P_N : \forall \widehat{p}[M] \leq \mu : M(p) \leq 1 \end{aligned}$$
- *OS is safe(4) iff for all reachable markings there is most one token on each place (w.r.t. projections):*
$$\begin{aligned} \forall \mu \in RS(OS): \quad & \forall \widehat{p} \in \widehat{P} : \Pi^1(\mu)(\widehat{p}) \leq 1 \wedge \\ & \forall N \in \mathcal{N} : \forall p \in P_N : \Pi^2_N(\mu)(p) \leq 1 \end{aligned}$$

We simply say that OS is a safe Eos *if any of the above properties hold.*

In the following we concentrate on the variant of safe(3) Eos, since it turned out that reachability for safe(1) or safe(2) Eos is undecidable and an algorithm for the reachability problem for safe(3) Eos will also work with safe(4) Eos. For generalised state machines safe(3) is even equivalent to safe(4) (cf. [14]).

In contrast to p/t nets not all notions of safeness are equivalent to finiteness of the state space: The state space of safe(1) or safe(2) Eos is infinite in the general case. Finiteness only holds for safe(3) and safe(4) Eos:

Theorem 3.3 ([14])**.** *If an* Eos *is safe(3) or safe(4) then its set of reachable markings is finite.*

By Theorem 3.3 we know that (strongly) safe Eos have finite state spaces, but compared to the state spaces of safe p/t nets they may become quite large: For 1-safe p/t nets it is known that whenever there are n places, then the number of reachable states is bound by $O(2^n)$. In the proof of Thm. 3.3 however, we have seen that whenever the number of places in the system net and in all object nets is bound by n, then the number of reachable states is in $O(2^{n^2})$ – a quite drastic increase. This combinatorical explosion makes it in general very hard to represent the state space explicitly. In the following we characterise this kind of blow-up in complexity theoretical terms.

4 Complexity of LTL Model Checking

It is a known fact that most interesting questions about the behaviour of classical 1-safe p/t nets like liveness, deadlock-freedom, and reachability are Pspace-hard (see [3]). This follows from the fact, first observed in [7], that a 1-safe p/t net of size $\mathcal{O}(n^2)$ can simulate a linear bounded automaton starting on an empty tape of size n. Since the net can furthermore be constructed in polynomial time, hardness results concerning linear bounded automata carry over to 1-safe p/t nets. From there they directly carry over to safe Eos, since 1-safe p/t nets can be seen as a special kind of safe(4) Eos, which are also safe(3), safe(2), and safe(1). Thus it is for instance Pspace-hard to decide reachability and liveness for safe Eos.

The more interesting question is therefore, if polynomial space suffices and, if so, devise algorithms for these problems.

In [14] we showed that many problems on safe(3) and safe(4) Eos can be decided in polynomial space. More explicitly we showed that Theorem 4.1 below holds. Among other problems

reachability can be expressed as an LTL formula and is thus PSPACE-complete. Liveness can not be expressed as an LTL formula and we will address the problem to decide liveness in the outlook. Our construction in [14] is an adjustment of the construction given in [3] for 1-safe p/t nets. In [14] we explicitly pointed out the similarities. Here we take a more algorithmic oriented point of view, but we want to emphasize that the following result heavily relies on ideas from [3] and also [25].

In the following we will uses LTL formulae to express a property like reachability with a formula ϕ. A net is then said to *satisfy* ϕ if all its runs satisfy ϕ. A run of a net is just a sequence of markings. To be able to express properties of a net with LTL formulas the set of atomic propositions should be closely related to the markings. To achive this let $\widehat{p_1}, \ldots, \widehat{p_n}$ be the system net's places, let $N_1, \ldots, N_n$ be the object nets with $d(\widehat{p_i}) = N_i$ for all i, and let $p_{N_i,1}, \ldots, p_{N_i,m_i}$ be the places of the object net N_i.[5] We now define

$$prop := \left\{ \begin{array}{l} \widehat{p_1}[], \widehat{p_1}[p_{N_1,1}], \ldots, \widehat{p_1}[p_{N_1,m_1}], \\ \ldots, \\ \widehat{p_n}[], \widehat{p_n}[p_{N_n,1}], \ldots, \widehat{p_n}[p_{N_n,m_n}] \end{array} \right\}$$

as the set of atomic propositions for the LTL formulae. A marking is a subset of prop and a computation (resp. a run) is a sequence of subsets of *prop*. Here $\widehat{p_1}[p_{N_1,1}] + \widehat{p_2}[]$ describes that the object net N_1 resides on the system net place $\widehat{p_1}$ and that an empty object net of type N_2 resides on the place $\widehat{p_2}$. The place $p_{N_1,1}$ of the object net N_1 is marked.[6]

In the following we want to show the following theorem:

Theorem 4.1. *Given a safe(3) or safe(4)* EOS *N and an LTL formula ϕ checking whether N satisfies ϕ can be done in polynomial space in the size of N and ϕ, that is, there is a polynomial p independent of N and ϕ such that the algorithm uses $O(p(|N| + |\phi|))$ space.*

Since reachability can be expressed as an LTL formula we have the following corollary:

Corrolary 4.2. *The reachability problem for safe(3) and safe(4)* EOS *is* PSPACE*-complete.*

We need the following important result:

Theorem 4.3. *Given an LTL formula ϕ, one can build a finite automaton A_ϕ and a Büchi automaton B_ϕ such that $L(A_\phi) \cup L_\omega(B_\phi)$ is exactly the set of computations satisfying the formula ϕ.*

The proof of Theorem 4.3 exceeds the scope of this paper (see [3] and [25] for details). Here it suffices to know the following two facts: First, the states of A_ϕ are sets of subformulae of ϕ and the states of B_ϕ are pairs of sets of subformulae of ϕ, thus they both may have exponentially many states in $|\phi|$ and must not be completly constructed if we want to prove Theorem 4.1 above. Yet note that a single state has polynomial size in $|\phi|$. Second, given two states q_1 and q_2 of A_ϕ or B_ϕ and a marking μ (which is a symbol of the alphabet of A_ϕ resp. B_ϕ), it is possible to decide in polynomial space (in $|\phi|$) if a transition from q_1 to q_2 with label μ exists. In what follows we will not need the automata A_ϕ and B_ϕ, but the automata $A_{\neg\phi}$ and

[5] Note that in general some places of the system net might be typed with the same object net. In that case $N_i = N_j$ for some i and j and also $p_{N_i,k} = p_{N_j,k}$ for all $1 \leq k \leq m_i = m_j$.

[6] Note that with this interpretation some subsets of *prop* are useless. For example the set $\{\widehat{p_1}[], \widehat{p_1}[p_{N_1,1}]\}$ describes the same marking as $\{\widehat{p_1}[p_{N_1,1}]\}$, namely that the net N_1 resides on the place $\widehat{p_1}$ and that its first place is marked. This is only relevant for the given formula ϕ. In general, in a conjunction of the form $\widehat{p_i}[] \wedge \widehat{p_i}[p_{N_i,k}] \wedge \ldots$ the conjunct $\widehat{p_i}[]$ can be deleted without loss. This is possible in time $O(|\phi|)$. If the formula is correctly given in the first place, no further problems arise.

Algorithm 1 Checking A for nonemptiness

Var: q of type state of $A_{\neg\phi}$
Var: μ of type state of A_N
1: $(q, \mu) \leftarrow (q_0, \mu_0)$;
2: **while** (q, μ) is not a final state of A **do**
3: guess a state q' of $A_{\neg\phi}$ with $q \xrightarrow[A_{\neg\phi}]{\mu} q'$
4: guess a marking μ' and an event $\widehat{\tau}[\theta]$ with $\mu \xrightarrow[N]{\widehat{\tau}[\theta]} \mu'$
5: $(q, \mu) \leftarrow (q', \mu')$
6: **end while**
7: **return true**

$B_{\neg\phi}$ instead. Below it will become clear why, but at first we will need two further automata, A_N and B_N, obtained from the safe Eos N, whose set of states correspond to the reachable markings of the net N. The initial state of A_N and B_N is the initial marking and the transition relation δ_N (again the same for A_N and B_N) contains the triple (μ_1, μ_1, μ_2) of markings if μ_2 can be reached from μ_1 via some event $\widehat{\tau}[\theta] \in \Theta$. Thus A_N and B_N correspond closely to the reachability graph and indeed only differ from it in the edges' labels. The set of final states of A_N is the set of deadlocked reachable markings of N and the set of final states of B_N contains all reachable markings of N, thus $L(A_N)$ is the set of all finite and $L_\omega(B_N)$ is the set of all infinite runs of N. Note that A_N and B_N may have exponentially many states in $|N|$ and we again can not construct them completly.

We now define two automata A and B with the usual construction to be the product automata of $A_{\neg\phi}$ and A_N, resp. of $B_{\neg\phi}$ and B_N. Then $L(A) = L(A_{\neg\phi}) \cap L(A_N)$ and $L_\omega(B) = L_\omega(B_{\neg\phi}) \cap L_\omega(B_N)$ holds[7] and $L(A) \cup L_\omega(B)$ is the set of runs of N that do *not* satisfy ϕ. The question whether N satisfies ϕ, i.e. if all of N's runs satisfy ϕ, is thus reduced to the question if $L(A)$ and $L_\omega(B)$ are both empty.

To prove Theorem 4.1 we now devise a *nondeterministic* algorithm for the *nonemptiness* problem of A and B. By Savitch's Theorem [21] a deterministic algorithm follows, and by reversing its answer we obtain the sought algorithm.

Apart from using nondeterminism we also make use of another trick: Since A and B may have exponentially many states in $|N|$ and $|\phi|$, we construct them *on the fly*. We start with the initial state as the current state (q, μ), repeatedly guess a next state (q', μ'), check that (q', μ') is indeed reachable from (q, μ) via some transition and, if so, update the current state. In the case of A, we know that $L(A)$ is nonempty, if we reach a final state. We thus return *true* in this case. Algorihm 1 is outlined below and is only a minor modification from the algorithm given in [3].

The case of B is only slightly more complicated. Since $L_\omega(B)$ is nonemtpy if and only if a final state is visited infinitly often, i.e. if there is a reachable final state q such that there is a loop from q to itself, we proceed as in Algorithm 1 but if we reach a final state, we guess that this final state will be revisited or not. This guess is then checked in a similar way as Algorithm 1 checks for any final state. See [3] for details.

Why are these algorithms polynomial space algorithms (in $|N|$ and $|\phi|$)? At first we note that only a constant number of states need to be saved and that the states of $A_{\neg\phi}$ (resp. $B_{\neg\phi}$)

[7] In general it is not the case that the product of two Büchi automata accepts the intersection of the languages, but it holds here.

and A_N (B_N) have polynomial size in $|\phi|$ and $|N|$.[8] Since A and B have been constructed with the product automata construction we actually need to check if a transition is possible in $A_{\neg\phi}$ and A_N to check if one is possible in A (and similar for B). For $A_{\neg\phi}$ (and $B_{\neg\phi}$) this follows from the construction of $A_{\neg\phi}$ which we have not shown here. (See for example [25].) In the algorithm one only needs to store a state of $A_{\neg\phi}$ and a symbol of the alphabet (which is a marking of N) of $A_{\neg\phi}$. Then one can check similar to Algorithm 1 if a final state can be reached. This can be done in polynomial space. See [25] for more details.

To check if a transition in A_N or B_N is possible, we need to check if, given two markings $\mu, \mu' \in \mathcal{M}$ and an event $\widehat{\tau}[\theta] \in \Theta$, the relation $\mu \xrightarrow[N]{\widehat{\tau}[\theta]} \mu'$ holds.

For this we first check that $\widehat{\tau}[\theta]$ is indeed an event. Using the synchronisation labeling this can easily be done in space polynomial in $|N|$, indeed we only need constant space for this. Now we nondeterministicallys guess[9] a mode $(\lambda, \rho) \in \mathcal{M}^2$, which is in $O(|N|^2)$ and check deterministically if the event $\widehat{\tau}[\theta]$ is enabled in μ for this mode, that is we check that $\lambda \leq \mu$ and $\phi(\widehat{\tau}[\theta], \lambda, \rho)$ holds. This is both possible in polynomial space, where the former is obvious and for the later we only need to compare partial markings of λ and ρ with the presets and postsets of certain nets. At last we check that $\mu' = \mu - \lambda + \rho$ holds, which again is possible in polynomial space. We thus have an algorithm to nondeterministically decide $\mu \xrightarrow[N]{\widehat{\tau}[\theta]} \mu'$ in polynomial space. By Savitch's Theorem [21] a deterministic algorithm follows.

Finally we need to be able to check if a state in A (resp. B) is a final state, that is we need to check for final states in $A_{\neg\phi}$, $B_{\neg\phi}$, A_N and B_N. For $A_{\neg\phi}$ and $B_{\neg\phi}$ one by their definition only needs to do some syntactic checks on the states, which are actually sets of subformulas of ϕ. This is easily possible in polynomial space. To check for final states in B_N is also simple, since the set of final states of B_N equals its set of states. At last to check if a state of A_N is a final state, we need to check if a marking μ of N, which is a state of A_N, is a deadlock, that is, we have to check that no event is enabled in μ. Above we have devised an algorithm which given μ guesses μ' and an event and then checks if μ' is reachable from μ with this event, that is the algorithm checks that μ is not a deadlock. Since this algorithm is deterministic we can just reverse the answer and have an algorithm to check if μ is a deadlock.

Putting it all together we have proven Theorem 4.1.

5 Conclusion and Outlook

In this paper we discussed the concept of safeness for Elementary Object Nets (Eos). We have shown that reachability is Pspace-complete for safe(3) and safe(4) Eos and that in fact every property that can be expressed in LTL is decidable in Pspace for safe(3) and safe(4) Eos. We also stressed the usefullness of Eos as a modelling tool for mobile objects or agents.

Not all interesting properties can be expressed in LTL, though, and liveness is a particularly interesting example. Liveness can be expressed in CTL. However, CTL formulas need to be treated differently than the LTL formulas discussed above. In [3] it is shown, that the model-checking problem for CTL and 1-safe p/t nets is in Pspace, too. It seems to us that these results, with slight modifications, carry over to safe(3) and safe(4) Eos, too. This and the related case for safe(1) and safe(2) Eos will be further investigated in a forthcoming paper.

Appart from this a closer look at the LTL case might be interesting. The model checking problem for many problems might indeed be far easier to solve than the general case presented

[8] To be more concrete: The states of A_ϕ and B_ϕ have size quadratic in $|\phi|$ and the states of A_N and B_N are quadratic in $|N|$.

[9] Note that we have already guessed μ' and $\widehat{\tau}[\theta]$, which also only have polynomial size in the size of the net.

here. Also the case for safe(4) Eos – even if Pspace-complete, too – might be algorithmically easier to solve than the case for safe(3) Eos from a practitioners point of view. This might be especially interesting with applications and therefore the need of an efficient implementation of the LTL model checking algorithm above in mind.

References

[1] Marek A. Bednarczyk, Luca Bernardinello, Wieslaw Pawlowski, and Lucia Pomello. Modelling mobility with Petri hypernets. In Josè Luiz Fiadeiro, Peter D. Mosses, and Fernando Orejas, editors, *Recent Trends in Algebraic Development Techniques (WADT 2004)*, volume 3423 of *Lecture Notes in Computer Science*, pages 28–44. Springer-Verlag, 2004.

[2] Steven Eker, José Meseguer, and Ambarish Sridharanarayanan. The Maude LTL model checker. In *Electronic Notes in Theoretical Computer Science*, volume 71. Elsevier, 2002.

[3] Javier Esparza. Decidability and complexity of petri net problems – an introduction. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.

[4] Serge Haddad and Denis Poitrenaud. Theoretical aspects of recursive Petri nets. In S. Donatelli and J. Kleijn, editors, *Application and Theory of Petri Nets*, volume 1639 of *Lecture Notes in Computer Science*, pages 228–247. Springer-Verlag, 1999.

[5] Kunihiko Hiraishi. PN^2: An elementary model for design and analysis of multi-agent systems. In Farhad Arbab and Carolyn L. Talcott, editors, *Coordination Models and Languages, COORDINATION 2002*, volume 2315 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag, 2002.

[6] Kathrin Hoffmann, Hartmut Ehrig, and Till Mossakowski. High-level nets with nets and rules as tokens. In *Application and Theory of Petri Nets and Other Models of Concurrency*, volume 3536 of *Lecture Notes in Computer Science*, pages 268 – 288. Springer-Verlag, 2005.

[7] N. D. Jones, L. H. Landweber, and Y. E. Lien. Complexity of some problems in petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[8] Michael Köhler. Reachable markings of object Petri nets. *Fundamenta Informaticae*, 79(3-4):401 – 413, 2007.

[9] Michael Köhler, Daniel Moldt, and Heiko Rölke. Modeling the behaviour of Petri net agents. In J. M. Colom and M. Koutny, editors, *Application and Theory of Petri Nets*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.

[10] Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling mobility and mobile agents using nets within nets. In W. v. d. Aalst and E. Best, editors, *Application and Theory of Petri Nets*, volume 2679 of *Lecture Notes in Computer Science*, pages 121–140. Springer-Verlag, 2003.

[11] Michael Köhler and Heiko Rölke. Concurrency for mobile object-net systems. *Fundamenta Informaticae*, 54(2-3), 2003.

[12] Michael Köhler and Heiko Rölke. Properties of Object Petri Nets. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets*, volume 3099 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 2004.

[13] Michael Köhler-Bußmeier. Hornets: Nets within nets combined with net algebra. In Karsten Wolf and Giuliana Franceschinis, editors, *Application and Theory of Petri Nets*, volume 5606 of *Lecture Notes in Computer Science*, pages 243–262. Springer-Verlag, 2009.

[14] Michael Köhler-Bußmeier and Frank Heitmann. Safeness for object nets. *Fundamenta Informaticae*, 2010. To appear.

[15] Olaf Kummer, Roxana Dietze, and Manfred Kudlek. Decidability problems of a basic class of object nets. *Fundamenta Informaticae*, 79(3-4):295–302, 2008.

[16] Charles Lakos. A Petri net view of mobility. In *Formal Techniques for Networked and Distributed Systems (FORTE 2005)*, volume 3731 of *Lecture Notes in Computer Science*, pages 174–188. Springer-Verlag, 2005.

[17] Irina A. Lomazova. Nested Petri nets – a formalism for specification of multi-agent distributed systems. *Fundamenta Informaticae*, 43(1-4):195–214, 2000.

[18] Irina A. Lomazova, Kees M. van Hee, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova, and Marc Voorhoeve. Nested nets for adaptive systems. In *Application and Theory of Petri Nets and Other Models of Concurrency*, Lecture Notes in Computer Science, pages 241–260. Springer-Verlag, 2006.

[19] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[20] Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[21] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. on Computer and System Sciences*, 4:177–192, 1970.

[22] Rüdiger Valk. Modelling concurrency by task/flow EN systems. In *3rd Workshop on Concurrency and Compositionality*, number 191 in GMD-Studien, St. Augustin, Bonn, 1991. Gesellschaft für Mathematik und Datenverarbeitung.

[23] Rüdiger Valk. Petri nets as token objects: An introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, 1998.

[24] Rüdiger Valk. Object Petri nets: Using the nets-within-nets paradigm. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Advanced Course on Petri Nets 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 819–848. Springer-Verlag, 2003.

[25] Moshe Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.