Modeling and Simulating Interaction Protocols Using Nested Petri Nets

Mirtha Lina Fernández Venero(✉) and Flávio Soares Corrêa da Silva

Department of Computer Science, University of São Paulo,
São Paulo 05508-090, Brazil
{mirtha,fcs}@ime.usp.br

Abstract. This paper is concerned with the problem of analyzing interaction protocols in a coordination platform called JamSession. We use nested Petri nets to provide a formal model for simulating the protocols and predicting conflicts on the system behavior.

1 Introduction

The notion of mobility has been increasingly used in areas such as communication protocols, multi-agent and intelligent systems, web and business applications, virtual environments, computers games, etc. This notion has introduced the need of designing location-dependent and context-aware software components whose complexity demands the unavoidable use of formal models (e.g. process calculi, Petri nets, Markov chains and automata-based techniques) for their development. In this article we use Petri nets (PNs) to provide a formal framework for simulating the interaction protocols of a coordination platform called JamSession. The platform was proposed in [3] for coordinating distributed, heterogeneous and mobile agents and resources. It uses a notion of location similar to the one provided in *Multilayered Multi-Agent Situated Systems* [1], where sites are related by pathways to form a directed graph. Agents inhabit these sites and can move from site to site to look for specific resources to accomplish their goals. Services are modeled using predicates attached to locations. Interaction protocols for coordinating agents are also linked to locations and are built from basic logic constructions. The language is simple but other notions such as norms and roles from *Electronic Institutions* [5] and the *Lightweight Coordination Calculus* (LCC) [11] can be modeled as well.

JamSession was recently used for coordinating inter-organizational workflows [4]. In that work, it was shown how hierarchical protocols can be verified using colored Petri nets (CPNs). This paper formalizes and extends that previous model for protocols involving recursive calls. Furthermore, we explain how to specify the dynamic behavior of concurrent interactions. Our aim is to provide a formal ground for the construction of a visualization tool for JamSession that supports the simulation and analysis of the agents movements. Here, we use

This work was supported by the São Paulo Research Foundation (FAPESP) under the grant 2010/52505-0.

nested Petri nets (NPNs), a class of high-level Petri nets where tokens can also be Petri nets [8]. As classical tokens, the net tokens can be added to or removed from places, but they can also fire their transitions, synchronizing them with other net tokens. The idea of using nets within nets has been effectively applied to multi-agent systems and mobile agents [2,7,10]. To model mobility, locations are encoded as places and the possible movements are encoded as transitions. Mobile agents are modeled as net tokens which can be moved from one place to another. Nevertheless, few methodologies have been proposed for modeling the rules that coordinate a sequence of agents movements [2]. In JamSession, these rules can be defined by means of interaction protocols. Therefore, in this article we provide a systematic approach for translating a JamSession specification into a NPN. For simplicity and due to the fact that in JamSession agents may be just passive entities, we represent the environment (agents, locations and feasible movements) as a color set and protocol calls as net tokens. However, the method can be easily adapted for dealing with agents nets. We model the system behavior as a Workflow Net [12] and define a property for its correctness which can be used for the early detection of interactions in conflict.

The paper has the following structure. Section 2 summarizes JamSession syntactical features and its computation rules. Section 3 presents an informal description of the translation of JamSession protocols into NPNs. The formal translation and the model for the dynamics of concurrent interactions are described in Sects. 4, 5 respectively. We draw some conclusions in Sect. 6.

2 The JamSession Platform

The coordination mechanism of JamSession is based on a directed graph where nodes represent locations that are inhabited by agents. The arcs of the graph characterize the admissible movements that agents can perform across locations. The agents provide services that are represented as first-order predicates. Each predicate is associated to a pair $[Agent, Location]$ and may also have *Input* and *Output* parameters. An agent stays in a location until it receives an order to move. Predicates and movements are combined in JamSession using interaction protocols which are linked to locations. A JamSession specification is a tuple $J = \langle Loc, Path, Ag, Var, D, Pred, Prot, \phi, \psi \rangle$ where

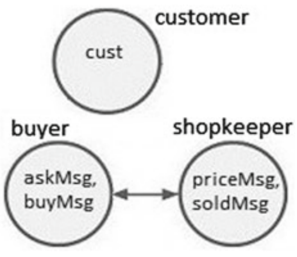
- $Loc \neq \emptyset$ is a set of locations and $Path \subseteq Loc \times Loc$ is a set of directed arcs between locations. The pair $(Loc, Path)$ is called the graph of locations;
- $Ag, Var, D, Pred, Prot$ are non-empty sets of agents, variables, domain values, predicate and protocol symbols respectively;
- $\phi : Pred \times Ag \times Loc \rightarrow (T_D \rightarrow \{\bot, \top\} \times T_D)$ characterizes the predicates definitions. Hereafter, T_C denotes the set of tuples over a set C;
- $\psi : Prot \times Loc \rightarrow \Sigma \times T_{Var} \times T_{Var}$ characterizes the protocols definitions and Σ is the language generated by the next rules, where $pd \in Pred$, $pt \in Prot$, $a \in Ag$, $l, l_1, l_2 \in Loc$, $V \in T_{Var}$ and $P \in T_{Var \cup D}$

$$\begin{aligned} Disj &:= Disj \vee Disj \mid \ Conj \\ Conj &:= Conj \wedge Conj \mid Entity \\ Entity &:= \bot \mid \top \mid \mathbf{move}(a, l_1, l_2) \mid [a, l]pd(P, V) \mid [l]pt(P, V) \end{aligned}$$

Given a predicate symbol *pd*, an agent a and a location l, the function $\phi(pd, a, l)$ takes a list of domain values as input and returns a list of output domain values and the result of the evaluation ($\perp$ or $\top$). Given a protocol symbol *pt* and a location l, the protocol definition $\psi(pt, l) = (F, V_i, V_o)$ is written as $[l]\ pt(V_i, V_o) ::= F$. The formula F has the structure of a disjunctive normal form in which literals may be move orders and predicate or protocol calls. Some of the variables occurring in F are considered as input (V_i) or output (V_o) variables in the protocol definition. The conjunction denotes the sequential evaluation of the atoms and the disjunction an alternative computation branch.

In JamSession, several protocols may be executed in parallel. The concurrent processes share the same configuration of the graph but they do not share variables. Besides, an agent can be used by just one predicate or move order at any given time. Predicates calls and move orders are suspended until the involved agent reaches the appropriate location. During the evaluation of a predicate, the agent is locked at the location. A move order is executed as an atomic operation.

Example 1. We illustrate the functioning of JamSession by means of two protocols (*buyerP* and *shopkeeperP*) describing an interaction for a basic shopping. The graph of locations is shown in the right. The agent tokens *askMsg*, *buyMsg*, *priceMsg* and *soldMsg* represent messages to be exchanged between the protocols. The protocols corresponding to the roles are shown below. We have used *c*, *b* and *sh* to abbreviate the location names *customer*, *buyer*, *shopkeeper* respectively.



The *buyerP* protocol has a client B and an item X as input parameters and no output variable. The input data is verified by means of an agent *cust* at *c*. The *updateAsk* predicate stores the required data for the *askMsg* token and the message is moved to the *sh* location. After that, the *getPrice* predicate waits until *priceMsg* reaches the *b* location. When this occurs, the message *priceMsg* is sent back to the *sh*. After checking that X is affordable, the *buyMsg* token is updated and sent, and the *getConf* predicate waits for *soldMsg*. Once it is received, it is sent back and the purchase is confirmed to the client using the *chkConf* predicate. The behavior of *shopkeeperP* protocol is similar but it has a recursive call at the end, in order to wait for another buyer.

$[c]\ buyerP((B,X),()) ::= [cust,c]\ need((B,X),()) \wedge$
$\quad [askMsg,b]\ updateAsk((B,X),()) \wedge \mathbf{move}(askMsg,b,sh) \wedge$
$\quad [priceMsg,b]\ getPrice((V),(P)) \wedge \mathbf{move}(priceMsg,b,sh) \wedge [cust,b]\ afford((X,P),()) \wedge$
$\quad [buyMsg,b]\ updateBuy((X,B),()) \wedge \mathbf{move}(buyMsg,b,sh) \wedge$
$\quad [soldMsg,b]\ getConf((),(C)) \wedge \mathbf{move}(soldMsg,b,sh) \wedge [cust,c]\ chkConf((C),()).$

$[sh] shopkeeperP() ::= [askMsg,sh]\ getAsk((),(X,B)) \wedge \mathbf{move}(askMsg,sh,b) \wedge$
$\quad [priceMsg,sh]\ instock((X),(P)) \wedge \mathbf{move}(priceMsg,sh,b) \wedge$
$\quad [buyMsg,sh]\ getBuy((),(X,B)) \wedge \mathbf{move}(buyMsg,sh,b) \wedge$
$\quad [soldMsg,b]\ setConf((X,B,P),()) \wedge \mathbf{move}(soldMsg,sh,b) \wedge$
$\quad [soldMsg,sh]\ closeSale() \wedge [sh]\ shopkeeperP().$

A state in the computation of a JamSession formula consists of a formula, the distribution of agents over the graph (represented as a function $st : Ag \rightarrow Loc$) and a substitution θ holding the values of instantiated variables. The transition relation between the states ($\rightarrow$) is defined by the rules in Table 1. We write $F \xrightarrow{st,\theta,st',\theta'} F'$ instead of $(F, st, \theta) \rightarrow (F', st', \theta')$ to improve readability. The notation $st(a) \uparrow l$ indicates that the state of the graph has changed by the movement of a to l. Furthermore, we use $\theta \uparrow v = d$ to denote a new substitution obtained from θ where the variable v has been updated with the domain value d. As usual, the application of a substitution θ to a formula F is written as $F\theta$. The replacement of all occurrences of a variable x in F by the a value or variable v is denoted as $F[v/x]$. These notations are extended to tuples of variables and values in a straightforward way.

Table 1. JamSession computation rules

1) $\bot \vee F \xrightarrow{st,\theta,st,\theta} F\theta$	2) $\top \vee F \xrightarrow{st,\theta,st,\theta} \top$
3) $\bot \wedge F \xrightarrow{st,\theta,st,\theta} \bot$	4) $\top \wedge F \xrightarrow{st,\theta,st,\theta} F\theta$
5) $F_1 \diamond F_2 \xrightarrow{st,\theta,st,\theta} F \diamond F_2$	$if\ \diamond \in \{\vee, \wedge\}, F_1 \xrightarrow{st,\theta,st,\theta} F$
6) $\mathbf{move}(a, l_1, l_2) \xrightarrow{st,\theta,st,\theta} \bot$	$if\ st(a) = l_1, (l_1, l_2) \notin Path$
7) $\mathbf{move}(a, l_1, l_2) \xrightarrow{st,\theta,st(a)\uparrow l_2,\theta} \top$	$if\ st(a) = l_1, (l_1, l_2) \in Path$
8) $[a, l]\, pd(P, V) \xrightarrow{st,\theta,st,\theta\uparrow V=O} b$	$if\ st(a) = l, \phi(pd, a, l)(P\theta) = (b, O)$
9) $[l]\, pt(P, V) \xrightarrow{st,\theta,st,\theta} F_1$	$if\ \psi(pt, L) = (F, V_i, V_o), F_1 = F[P\theta/V_i][V/V_o]$

The first three rows of Table 1 describe the rules for conjunction and disjunction. The left-hand side of these operators must be reduced to a truth value before the right-hand side can be rewritten. This is enforced by the fifth rule. The sixth rule states that a move order fails in case the agent inhabits a location with no direct arc to the intended destination. On the contrary, a move order holds (rule 7) if l_1 is the current location of a and (l_1, l_2) is an arc of the graph. If a has not reached l_1, the move order is postponed until it can be evaluated. Predicate calls have a similar behavior with respect to agents and locations. If a is already situated at l, the function $\phi(pd, a, l)$ is evaluated for the input values, the formula is reduced, and the output variables are updated. Finally, a protocol call is unfolded by applying the function ψ to obtain its body definition. W.l.o.g we assume that each time a fresh copy is obtained from ψ (i.e. with a fresh set of variables). Furthermore, the input/output variables of the new formula are replaced by the parameters of the call. The substitution θ is initially empty and it is updated by the rules 6,7,8 and 10. It is applied to the remaining atoms of the formula using rules 1 and 4. We write $F \rightarrow^* F'$ if F reduces to F' in 0 or more steps. We write $F \xrightarrow{st,st_f} F'$ when no further step can be done from F'.

3 PN-Based Semantics for JamSession

In [4], it was shown how to model non-recursive protocols in JamSession using hierarchical CPNs [6]. CPNs are PNs in which each place has a type (color set) that describes the tokens it may store. The state of a CPN, called a marking, is a function relating each place to the multiset of tokens that inhabit it. Transitions represent actions or events that may change the marking of the net. An incoming (respectively outgoing) arc of a transition indicates that it may remove (respectively add) tokens from the corresponding place. The number and color of tokens to be removed or added is determined by the arc expressions which may contain variables. A transition is enabled in a marking if there is a binding of the variables which satisfies the expressions on the input arcs. In this case, the transition may fire, consuming and producing the input and output tokens respectively.

In this section we present an informal description of the translation of JamSession protocols into colored and nested PNs. As in [4], three basic color sets are used: Ag, $AgTok = Ag \times Loc$ and $Bool = \{\bot, \top\}$. The state of the graph of locations is represented by means of a special place of $AgTok$ type, denoted as SGL. In addition, the CPN associated to a JamSession formula has two special places of $Bool$ type: one with no incoming arc (*source node*) and the other one with no outgoing arc (*sink node*). These places are denoted as In and Out respectively. The CPNs corresponding to a move order and a predicate call have a single transition relating In, Out and SGL (see Fig. 1 a and b). In case of a predicate call, after firing t_p, the content of SGL remains the same and a $Bool$ token is produced at Out. The latter is represented by a variable (x) that indicates any token value belonging to the type. In the net corresponding to a move order, the firing of t_m produces a (possibly new) token at SGL and a $Bool$ token at Out. The color of these tokens depends on the existence of an arc between the involved locations. To this end, we use the function s to represent adjacency relation of the graph and the conditional operator ($\_?\_ : \_$).

The structure of the net for either $A \wedge B$ or $A \vee B$ is depicted in Fig. 1c. The substitution transitions A and B represent the nets for the operands. The input (respectively output) place of the substitution transition is fused with the input (respectively output) place of the associated CPN. The output token of the CPN for A enables an intermediary transition $t_\diamond$ that controls the activation of the CPN for B. For the CPN of the conjunction, the outgoing arcs of $t_\diamond$ are labeled by the expressions (1) $x = \top ? \top : \emptyset$ and (2) $x = \bot ? \bot : \emptyset$. Here, $\emptyset$ indicates that no token should be added to the output place[1]. For the CPN of a disjunction, the expressions are defined as (1) $x = \bot ? \top : \emptyset$ and (2) $x = \top ? \top : \emptyset$. Note that this is not the usual (non-deterministic) PN representation of an alternative. This is because JamSession disjunction evaluates the right-hand side only if the left-hand side was previously reduced to $\bot$.

If the protocol definitions are not recursive, then a hierarchical CPN can be used to represent a formula. However, a more powerful formalism is required

[1] This arc inscription is allowed e.g. by CPN Tools.

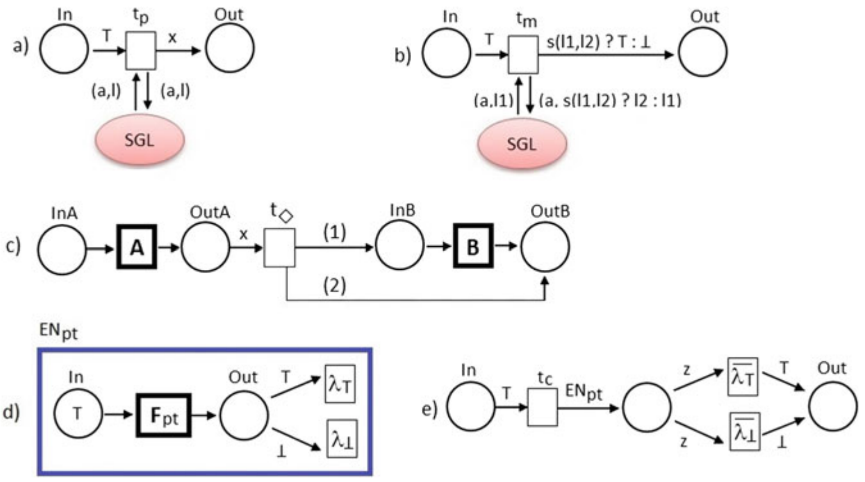


Fig. 1. PNs for JamSession constructions: a) $[a, l]\ pd(P, V)$; b) $move(a, l1, l2)$; c) $A \diamond B$; d) protocol definition and e) protocol call.

for protocols that are recursively defined. In this paper we use NPNs [9], an extension of CPNs in which tokens can be also nets. These net tokens may be added or removed as ordinary ones. In addition, they are allowed to change the marking by firing their own internal transitions. More precisely, a NPN is formed by several CPNs $(SN, EN_1, \ldots, EN_n)$, one of them called *system net* (SN) and the rest *element nets*. Each EN_i is considered as a type whose values are marked nets of the form (EN_i, M). The firing of a transition t, in SN or a marked net, may be performed according to the classical PN rules. In addition, a net token may synchronize the firing with another net token at the same place (*horizontal synchronization step*) or with the parent net (*vertical synchronization step*). The synchronization is performed by means of two disjoint sets of labels, respectively Lab_h and Lab_v, which are attached to transitions. It is assumed that for each label $l \in Lab_v$ there is a complementary label $\bar{l} \in Lab_v$.

In our model, we associate an element net to each protocol definition. The net (say EN_{pt}) is built by adding two sink transitions at the *Out* place of the protocol formula, as shown in Fig. 1d. These transitions represent the two possible results of a protocol call and they are labeled for vertical synchronization. A protocol call is modeled as depicted in Fig. 1e. After the *In* node, the net has a transition which creates a net token of EN_{pt} type (say nt) at an intermediary place (pc). The initial marking of nt has a token $\top$ at the source place and the remaining places are empty. The child net may perform several steps, corresponding to the reduction sequence of the protocol call. Once nt reaches a final marking (i.e. a *Bool* token at its *Out* place) the execution of the protocol call terminates and one of the sink transitions gets enabled. The complementary transition at the parent net will be also enabled by the binding $z = nt$. Hence, a vertical synchronization occurs, the transitions fire, nt is removed from pc and a $Bool$ token is added at the Out place of the parent net.

4 Formal Translation of Jamsession Protocols into NPNs

In the later we provide the formal translation of Jamsession protocols into NPNs. To this end, firstly we adapt the definition of NPN from [9] for sharing some places of the system net. Besides, we restrict ourselves to autonomous steps and the vertical steps that remove the net tokens involved. As usual in CPNs, we have a set of finite basic types and a set of basic constants belonging to these types. The element nets represent types and constants. We assume that the arc expressions are multisets over the constants and typed variables. However, we will omit the braces for multisets of a single element.

Definition 1. *A NPN is a tuple $N = (\Sigma, P_s, L, (EN_0, EN_1, \ldots, EN_n))$ s.t. Σ is a finite set of non-empty basic types, P_s is a finite set of shared places and L is a set of labels s.t. for each $l \in L$, there is a complementary label $\overline{l} \in L$ s.t. $\overline{\overline{l}} = l$ and for all $l_1, l_2 \in Lab_v$, $l_1 \neq l_2$ implies $\overline{l_1} \neq \overline{l_2}$. Furthermore, for all $i = 0 \ldots n$, $EN_i = (P, C, I, T, \Lambda, A, W)$ (called net component) is a colored Petri Net where*

- *P is a finite set of places s.t. $P_s \subset P$ if $i = 0$ and $P \cap P_s = \emptyset$ if $i > 0$,*
- *$C : P \to \Sigma \cup \{\{EN_1\}, \ldots, \{EN_n\}\}$ is a type function s.t. for all $p \in P_s$, $C(p) \in \Sigma$,*
- *I is the initial function defined from P into closed expressions over Σ,*
- *T is a finite set of transitions s.t. $P \cap T = \emptyset$,*
- *Λ is a partial function from T to L,*
- *$A \subseteq ((P_s \cup P) \times T) \cup (T \times (P_s \cup P))$ is a set of arcs,*
- *W is an arc expression function defined from A to expressions s.t.*
  - *there are no net constants in input arc expressions;*
  - *every variable has at most one occurrence in each input arc expression;*
  - *given two arcs (p_1, t) and (p_2, t), $Var(W(p_1, t)) \cap Var(W(p_2, t)) = \emptyset$;*
  - *for each net variable $x \in Var(W(t, q))$ there should be one input arc of t s.t. $x \in Var(W(p, t))$; and*
  - *if $\Lambda(t)$ is defined and $x \in Var(W(p, t)) \cap Var(W(t, q))$ then $C(x) \in \Sigma$.*

The net components share a set of places of basic types belonging to EN_0. The remaining places and transitions of the net components are pairwise disjoint. A *marking* of an element net is inductively defined as follows.

- A marking of EN_i over N, $1 \leq i \leq n$, is a function M, mapping each place p in EN_i to a finite multiset over Σ. The pair (EN_i, M) is called a marked net component or a net token of EN_i.
- Let $\bar{\Sigma}$ be a set of marked net components. Then a function M, mapping each place in a net component EN_i to a finite multiset over $\bar{\Sigma} \cup \Sigma$, is also marking of EN_i over N.

Let $\bar{\Sigma}$ denote the set of net tokens of NPN N. A marking of N is a function M, mapping each place in the net component EN_0 to a finite multiset over $\bar{\Sigma} \cup \Sigma$. Any marking must respect the type definition of the place. Hence, for all $p \in P$, if $C(p) \in \Sigma$, then $M(p)$ is a multiset over $C(p)$; otherwise $M(p)$ is a multiset of net tokens of $C(p)$. The initial marking of any net component is the marking obtained from the initialization expressions. The constant EN_i represents the marked net (EN_i, I_i). The initial marking of N is denoted as I_0. By definition, all places with net type are initially empty.

Given a transition t in a net component EN_i, we write $W(t)$ for the set $\{W(a) | a = (p, t) \in A\}$. A *binding* for t is a function b assigning to each variable $v \in W(t)$ a value from $\bar{\Sigma} \cup \Sigma$ (of the corresponding type). It is extended in a straightforward way to set of expressions. A transition t may fire in a marking M if it is *enabled* w.r.t. a binding b, i.e. for all $a = (p, t) \in A$, $b(W(a)) \subseteq M(p)$. If so, after the firing, it is obtained a new marking M' s.t. for any place p, $M'(p) = (M(p) - b(W(p, t))) \cup b(W(t, p))$. This is denoted as $M[t\rangle M'$. The set $\{b(x) \notin \Sigma \mid x \in W(p, t)\}$ are the net tokens involved in the firing of t.

An *autonomous step* is the firing of an unlabeled transition in SN or in a net token, according to the above rule. A *vertical step* is the firing of a transition t, labeled as $l = \Lambda(t)$ and the firing of a transition labeled as $\bar{l}$ in all net tokens involved in the firing of t. Due to the restrictions on the arc expressions, any vertical step removes the involved net tokens. We say that M' is *directly reachable* from M, denoted as $M[\rangle M'$, if there is an autonomous or vertical step s.t. $M[t\rangle M'$. A marking M is called *dead* if there is no directly reachable marking from it. It is called *reachable* if there is a sequence of zero or more steps $I_0[\rangle M_1[\rangle \ldots [\rangle M_k$ s.t. $M_k = M$. This is denoted as $I_0[*\rangle M$. A NPN terminates if there is no infinite sequence of steps starting from I_0.

The next definition provides the formal translation of a JamSession formula Finto a NPN. As we mentioned in the previous section, the element nets are obtained from the protocols definitions and the system net is the net associated to F. Case I.1.a of the definition deals with the translation of $\top$ and $\bot$. Cases I.1.b, I.1.c, I.2, I.3 and II correspond to the nets in Fig. 1a, b, c, d and e respectively. The initial marking of SN has a token $\top$ at the source and the SGL place with the initial state of the graph of locations.

Definition 2. *Let $J = \langle Loc, Path, Ag, Var, D, Pred, \phi, Prot, \psi\rangle$ be a JamSession specification, F be a JamSession formula and st an initial configuration of the graph. The NPN associated to J and F is $N = (\{Bool, Ag, AgTok\}, \{SGL\}, \{\lambda_\top, \bar{\lambda}_\top, \lambda_\bot, \bar{\lambda}_\bot\}, (EN_F, EN_{pt_1}, \ldots, EN_{pt_k}))$ where*

I- $EN_F = (\{SGL\} \cup P, C, I, T, \Lambda, A, W)$ is s.t. $C(SGL) = AgTok$, $I(SGL)$ is the multiset obtained from st and

1. If $F \in \{\top, \bot\}$ or $F = move(a, l_1, l_2)$ or $F = [a, l]\ pd(\ldots)$ then $P = \{In, Out\}$, $C(In) = C(Out) = Bool$, $I(In) = \top$ and $\Lambda = \emptyset$. Besides,

a. If $F \in \{\top, \bot\}$ then $T = \{t_F\}$, $A = \{a_1 = (In, t_F), a_2 = (t_F, Out)\}$, $W(a_1) = \top$ and $W(a_2) = F$.

b. If $F = [a,l]\ pd(\ldots)$ then $T = \{t_p\}$, $A = \{a_1 = (In, t_p), a_2 = (t_p, Out)$, $a_3 = (SGL, t_p), a_4 = (t_p, SGL)\}$, $W(a_1) = \top$, $W(a_2) = x$ is a Bool variable, $W(a_3) = W(a_4) = (a,l)$.

c. If $F = move(a, l_1, l_2)$ then $T = \{t_m\}$, $A = \{a_1 = (In, t_m), a_2 = (t_m, Out), a_3 = (SGL, t_m), a_4 = (t_m, SGL)\}$, $W(a_1) = \top$, $W(a_2) = s(l_1, l_2)?\top : \bot$, $W(a_3) = (a, l_1)$ and $W(a_4) = (a, s(l_1, l_2)?l_2 : l_1)$.

2. If $F = F_1 \diamond F_2$ with $\diamond \in \{\vee, \wedge\}$, let N_1 and N_2 the nets constructed for F_1 and F_2. Then $P = P_1 \cup P_2$, $C = C_1 \cup C_2$, $I = I_1$, $T = T_1 \cup T_2 \cup \{t_\diamond\}$, $\Lambda = \Lambda_1 \cup \Lambda_2$, $A = A_1 \cup A_2 \cup \{a_1 = (Out_1, t_\diamond), a_2 = (t_\diamond, In_2), a_3 = (t_\diamond, Out_2)\}$ and $W = W_1 \cup W_2 \cup \{W(a_1) = x\} \cup W_\diamond$. If $F = F_1 \wedge F_2$, then $W_\diamond = \{W(a_2) = x = \top?\top : \emptyset, W(a_3) = x = \bot?\bot : \emptyset\}$; otherwise $F = F_1 \vee F_2$ and $W_\diamond = \{W(a_2) = x = \bot?\top : \emptyset, W(a_3) = x = \top?\top : \emptyset\}$.

3. If $F = [l]\ pt(\ldots)$ then $P = \{In, Out, p_c\}$, $C(p_c) = \{EN_{pt}\}$, $T = \{t_c, t_\top, t_\bot\}$, $\Lambda(t_\top) = \bar{\lambda}_\top$, $\Lambda(t_\bot) = \bar{\lambda}_\bot$, $A = \{a_1 = (In, t_c), a_2 = (t_c, p_c)$, $a_3 = (p_c, t_\top)$, $a_4 = (p_c, t_\bot), a_5 = (t_\top, Out), a_6 = (t_\bot, Out)\}$, $W(a_1) = \top$, $W(a_2) = EN_{pt}$, $W(a_3) = W(a_4) = z$ is a variable of EN_{pt} type, $W(a_5) = \top$ and $W(a_6) = \bot$.

II- There is one component net EN_{pt_i} for each protocol definition $\psi(pt, l) = (F_1, V_i, V_o)$. The net $EN_{pt} = (P, C, I, T, \Lambda, A, W)$ is constructed from the net $N_1 = (P_1, C_1, I_1, T_1, \Lambda_1, A_1, W_1)$ corresponding to F_1. This way, we have $P = P_1 - \{SGL\}$, $C = C_1 - \{C_1(SGL)\}$, $I = I_1$, $T = T_1 \cup \{tr_\top, tr_\bot\}$, $\Lambda = \Lambda_1 \cup \{\Lambda(tr_\top) = \lambda_\top, \Lambda(tr_\bot) = \lambda_\bot\}$, $A = A_1 \cup \{a_1 = (Out, tr_\top), a_2 = (Out, tr_\bot)\}$ and $W = W_1 \cup \{W(a_1) = \top, W(a_2) = \bot\}$.

In Appendix A, Proposition 1, we prove that this translation preserves the semantics of Table 1, i.e, any reduction sequence of F can be simulated by a firing sequence of N. If the computation of F is finite the firing sequence is also finite and ends with the same state of the graph. If F leads to an infinite execution then the net has also an infinite firing sequence. Furthermore, if we label each reduction step and each autonomous step of the net with the involved operation $([a,l]pd, move(a, l_1, l_2), \vee, \wedge, [l]pt)$, then we can show that the resulting sequences of labels are the same. The translation models all possible reduction sequences of the formula, abstracting away from input/output parameters and even the initial configuration of the graph. Therefore, the behavior of the net may include firing sequences corresponding to infeasible execution sequences. But these sequences may become feasible when F becomes part of an interaction or some predicate definition changes.

5 The Dynamic Behavior of Concurrent Protocols

Workflow definitions provide an effective method for specifying the execution flow of a set of tasks. They can be modeled by PNs where the tasks are represented by transitions and the places represent causal dependencies. These nets are called *Workflow Nets* (WF-nets) [12] and they have a unique source place i and a unique sink place o. Furthermore, every other place or transition is on a path from i to o. The initial and final markings of the net have a single token at i and o resp and are denoted in the same way.

The dynamic behavior of a JamSession interaction can be specified by means of a NPN where the system net models the execution flow of a set of concurrent formulas. The net SN can be obtained from a WF-net (say WN) by replacing each transition corresponding to a task (say T) with the JamSession net associated to a formula (say NF). Let In and Out be the source and the sink of NF respectively. Then, the next rules can be used for the replacement of T by NF:

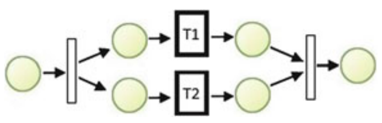
1. Add transitions it and ot and arcs (it, In) and (Out, ot) labeled as $\top$ and z respectively, where z is a$Bool$ variable.
2. Replace each arc (p, T) or (T, p) by (p, it) or (ot, p) respectively.

As an alternative, the formula may be defined as an element net instead of embedding it in the WF-net. In this case the rules are:

1. Add a sink transition t to NF with a label for vertical synchronization, e.g. λ. Furthermore, add an arc (Out, t) with a $Bool$ variable as the label and define the resulting net as an element net, say EN_F.
2. Add a place pF of EN_F type to the WF-net and two transitions it and ot s.t. ot is labeled as $\bar{\lambda}$. Furthermore, add the arcs (it, pF) and (pF, ot) labeled as EN_F and z respectively, where z is a variable of EN_F type.
3. Replace each arc (p, T) or (T, p) by (p, it) or (ot, p) respectively.

In both cases, the last rule must preserve the arc labels. The latter replacement is more suitable for interactions that require the parallel composition of multiple instances of the same formula. In such a case, the expression of the arcs (it, pF) and (pF, ot) should be defined with a number of constants and variables according to number of required instances.

Example 2. The interaction of Example 1 can be modeled using the WF-net for the parallel composition of two tasks (shown on the right). In Fig. 2, the tasks have been replaced (using the two approaches above) by the nets corresponding to each protocol call. The net tokens are represented as black dots with an arrow pointing to the marked net. The starts stand for a net of a predicate call or a move order.



A workflow is correct if its WF-net is *sound* [12]. Three conditions are required to satisfy this property. First, from the initial marking, it is always possible to reach the final state. Second, the final marking is the only marking reachable with a token at o. Finally, every task must be performed for at least one execution of the workflow. We use this property to define the correctness of a JamSession interaction. Note that the net resulting from the above rules is also a WF-net and preserves all the nodes from WN. Therefore, we may assume that any marking of WN is also a marking of N (the remaining places of SN are empty, except SGL). In a sound interaction there should be no conflict in the use of agents, i.e, if the evaluation of a predicate or mover order is required then

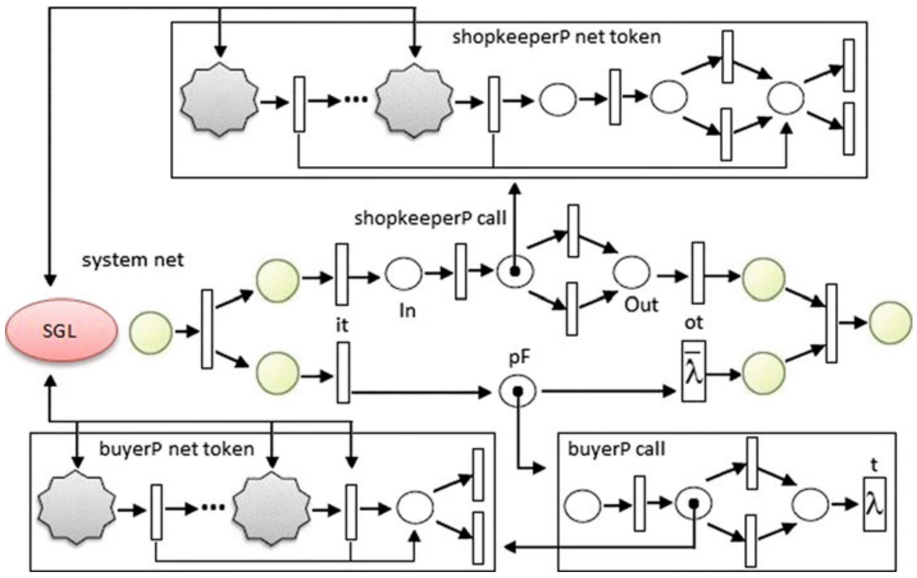


Fig. 2. A system net for the JamSession interaction of Example 1

it will be eventually completed. Furthermore, the interaction should terminate by reducing all formulas along a workflow path to a truth value. The soundness of WN ensures that if there exists a dead state other than o, then it is due to a transition in the net of a formula, in particular a predicate or move transition.

Definition 3. *Let $SF = \{F_1, \ldots, F_n\}$ be a set of formulas over a JamSession specification J. Let WN be a sound WF-net over the tasks $T_1, \ldots, T_n$ where for all $1 \leq i \leq n$, T_i is associated to F_i. Furthermore, let N be the NPN obtained from J, SF and WN. The interaction N is sound for an initial marking I_0 if and only if N terminates and for any marking M, $I_0[*\rangle M$ implies $M[*\rangle o$.*

The NPNs in which the vertical synchronization consumes the child nets are called NPNs *with autonomous elements*. For these nets, a finite coverability tree can be effectively constructed [9]. The leaves of this tree allow to decide termination and investigate properties of infinite sequences and dead markings. The nets defined in Sect. 4 are NPNs with autonomous elements in which the net tokens may share a set of basic places belonging to SN. This extension has little influence on the construction of the coverability tree. Therefore, the soundness property for a JamSession interaction can be decided by inspecting the leaves of this tree. See Appendix A, Proposition 2 for further details.

6 Conclusions

The NPN approach provided a suitable framework for modeling and simulating the interaction protocols in JamSession. The translation presented in this work is well-suited for automation and can be extended to other constructions. For simplicity, we encoded agents and locations as colored tokens. However, the place SGL can be unfolded into several places corresponding to locations and the agents behavior can be represented as element nets. The model is easily adapted to allow predicates dealing with several agents that may be synchronized to perform a common task. The environment (e.g. the topology of the graph and its initial configuration) and the protocols can be modified without affecting the system structure. Therefore, we believe it can be helpful for analyzing multi-agent interactions involving recursion, e.g. in related initiatives such as LCC. The main disadvantage of this approach is the lack of automated tools for NPNs. Nevertheless, model checking tools can help in verifying termination, reachability and the soundness property defined in this paper. Preliminary results on this direction can be found in [13].

Acknowledgments. The authors are grateful to the anonymous reviewers for their comments on an earlier version of this paper.

A Proofs

In this section we prove that, given a JamSession formula F and an initial configuration of the graph, the NPN obtained from Definition 2 has a firing sequence that simulates the reduction sequence of F (Proposition 1). Furthermore, we show that the soundness property defined for a JamSession interaction is decidable (Proposition 2). In the later we say that a marking M_f of a NPN associated to a JamSession formula is final if there is a single token c at the sink place and all other places, but SGL, are empty. This is denoted as M_f^c. We will assume that a marking may contain empty places not belonging to the net.

Lemma 1. *Let J be a JamSession specification, F be a JamSession formula, st an initial configuration of the graph of locations and N be the NPN associated to J and F. If $F \xrightarrow{st,st_f} c$ with $c \in Bool$ then there is M_f s.t. $I_0[*\rangle M_f^c$ and $M_f^c(SGL) = st_f$.*

Proof. The property trivially holds if $F \in \{\top, \bot\}$: the only transition in net (t_F) is enabled in I_0 and, after it fires, the final marking M_f^F is obtained. Hence, we have $I_0[t_F\rangle M_f^c$ with $F = c$ and $I_0(SGL) = M_f^c(SGL) = st = st_f$. If $F = move(a, l_1, l_2)$ then $F \xrightarrow{st,st_f} c$ either by rule 6 or 7. In both cases we have $st(a) = l_1$ and, by definition of N, I_0 has a token (a, l_1) at SGL. Since $I_0(In) = \top$, the transition t_m is enabled and an autonomous step occurs. After that, the In place is empty, the Out place has a token which coincides with c and SGL is updated according to the rule applied. Hence, $I_0[t_m\rangle M_f^c$ and $M_f^c(SGL) = st_f$ holds. When $F = [a, l]\ pd(\ldots)$ then $F \xrightarrow{st,st_f} c$ by rule 8 and hence $st(a) = l$. Therefore, $(a, l) \in I_0(SGL)$, the transition t_p is enabled and an autonomous step occurs. After that, the In place is empty, SGL remains unchanged and the

Out place has a token c using the binding $x = c$. Thus, we have $I_0[t_p\rangle M_f^c$ and $I_0(SGL) = M_f^c(SGL) = st = st_f$.

For the remaining cases, we use induction on the length of the sequence $F \xrightarrow{st,st_f} c$. If $F = F_1 \wedge F_2$, then by rules 3–5, we have that $F_1 \xrightarrow{st,st_f^1} c_1$ with $c_1 \in \{\top, \bot\}$. Note that I_0 can be considered as an initial marking for the net obtained from F_1, say N_1. By induction, there is a final marking $M_{f,1}$ s.t. $I_0[*\rangle M_{f,1}^{c_1}$ and $M_f^c(SGL) = st_f^1$. This marking for N_1 also enables the transition $t_\wedge$ which, after firing, removes c_1 from Out_1. If $c_1 = \bot$ then $F \xrightarrow{st,st_f^1} c_1$ using rule 3 and $t_\wedge$ adds $c_1 = c$ to Out_2 which is also the sink place of N. Hence, we have $I_0[*\rangle M_{f,1}^{c_1}[t_\wedge\rangle M_f^c$ and $M_f^c(SGL) = st_f^1 = st_f$. On the contrary, if $c_1 = \top$ then $F \rightarrow^* \top \wedge F_2 \xrightarrow{st_f^1,\theta,st_f^1,\theta} F_2\theta$ by rule 4. Besides, c_1 is added at In_2 by $t_\wedge$, leading to a marking M'. The net N_2 for F_2 coincides with net for $F_2\theta$ (say N_2') and M' is an initial marking for N_2'. Using induction, we have that if $F_2\theta \rightarrow^! c$ then there $M'[*\rangle M_f^c$ and $M_f^c(SGL) = st_f$. Since M_f^c is also a final marking for N, we obtain $I_0[*\rangle M_{f,1}^{c_1}[t_\wedge\rangle M'[*\rangle M_f^c$. The proof is analogous in case $F = F_1 \vee F_2$.

When $F = [l]\ pt(\ldots)$ we have $F \xrightarrow{st,\theta,st,\theta} F_1$ by rule 9. Let N_1 be the net associated to F_1. By the induction hypothesis, if $F_1 \xrightarrow{st,st_f} c$ then there is $M_{f,1}$ s.t. $I_1 = M_{11}[\rangle M_{21}[*\rangle M_{k1} = M_{f,1}^c$ and $M_{f,1}^c(SGL) = st_f$. Note that, in N, the transition t_c is enabled and we have the autonomous step $I_0[t_c\rangle M_1$ where $M_1(In) = M_1(Out) = \emptyset$ and $M_1(p_c) = (EN_{pt}, I_{pt})$. By Definition 2, the element net EN_{pt} has the same structure as N_1, except for *SGL* and the two sink transitions at the end. Hence, the marking $I_{pt} \cup I_0(SGL)$ coincides with I_1. Furthermore, for every marking M_{i1} in the sequence $I_1[*\rangle M_{f,1}^c$ we obtain a marking M_i in N by defining $M_i(In) = M_i(Out) = \emptyset$, $M_i(SGL) = M_{i1}(SGL)$ and $M_i(p_c) = (EN_{pt}, M_{i1} - M_{i1}(SGL))$ with $1 \leq i \leq k$. Thus, we obtain a sequence $M_1[\rangle M_2[*\rangle M_k$ of autonomous steps in N. The marking M_k enables the transition tr_c in the net token at *pc*. At the same time, the transition t_c in N gets enabled. Therefore, by a vertical step, the net token is removed from *pc* and a token c is added at *Out* reaching desired final marking. All in all, we obtain the sequence $I_0[\rangle M_1[*\rangle M_k[\rangle M_f^c$ s.t. $M_f^c(SGL) = st_f$. □

Lemma 2. *Let J be a JamSession specification, F be a JamSession formula, st an initial configuration of the graph of locations and N be the NPN associated to J and F. If $F \xrightarrow{st,st'} F'$ with $F' \notin Bool$ then there is a dead marking M s.t. $I_0[*\rangle M$, $M(SGL) = st'$ and $M(Out) = \emptyset$.*

Proof. If $F = F'$ then either $F = move(a, l_1, l_2)$ or $F = [a, l]\ pd(\ldots)$ none of the rules can be applied. This is due to the fact that $st(a) \neq l_1$ and hence in the initial marking I_0, there is no token (a, l_1) at *SGL*. Therefore, the only transition in net is not enabled, the initial marking is dead and we obtained $I_0[*\rangle I_0 = M$, $M(Out) = \emptyset$ and $M(SGL) = st = st'$.

We proceed using induction on the length of the sequence $F \xrightarrow{st,st'} F'$ and the size of the formula. If $F = F_1 \diamond F_2$ with $\diamond \in \{\vee, \wedge\}$ then we have the next two cases. Let N_1 and N_2 be the nets obtained from F_1 and F_2 respectively. If $F_1 \xrightarrow{st, st_1} F_1'$ with $F_1' \notin \mathit{Bool}$ then $F' = F_1' \diamond F_2$ and $st_1 = st'$. Using induction we have that, for N_1 there is a dead marking M_1' s.t. $I_0[*\rangle M_1'$ and $M_1'(SGL) = st_1$. Since Out_1 (the sink place of N_1) is empty, the transition $t_\diamond$ is not enabled and the marking is also dead for N. Otherwise, $F_1 \xrightarrow{st, st_1} c$ and $F \to^* c \diamond F_2 \xrightarrow{st_1, \theta, st_1, \theta} F_2\theta \xrightarrow{st_1, st'} F'$. Then, by Lemma 1, $I_0[*\rangle M^c_{f,1}$ for N_1. The same firing sequence can be considered for N leading to the firing of the transition $t_\diamond$. The marking obtained is an initial marking for N_2 (which coincides with the net for $F_2\theta$). Let denote this marking as I_1. Now, using induction, we have that $I_1[*\rangle M$, M is dead, $M(SGL) = st'$ and $M(Out_2) = \emptyset$. The required result holds since $I_0[*\rangle M^\top_{f,1}[t_\diamond\rangle I_1[*\rangle M$ and M is also a dead for N.

Finally, if $F = [l]\ pt(\ldots)$ we have $F \xrightarrow{st, \theta, st, \theta} F_1$ by rule 9. By the induction hypothesis, the net N_1 corresponding to F_1 has a firing sequence s.t. $I_1[*\rangle M_1$, M_1 is a dead marking, $I_1(SGL) = st$, $M_1(SGL) = st'$ and the sink place of N_1 is empty. For the net N we have $I_0[t_c\rangle M'$ where $M'(In) = M'(Out) = \emptyset$ and $M'(p_c) = (EN_{pt}, I_{pt})$. Since EN_{pt} has the same set of places as N_1 except for SGL, the sequence $I_1[*\rangle M_1$ can be considered as the inner sequence of the net token at p_c. Hence, we obtain a sequence $I_0[*\rangle M'[*\rangle M$ of autonomous steps in N s.t. $M(SGL) = st'$. However, since M_1 is dead and the *Out* place of the net token is empty, the transitions for vertical synchronization will never fire. Therefore M is also dead in N. □

Proposition 1. *Let J be a JamSession specification, $F \in \Sigma$ be a JamSession formula, st an initial configuration of the graph of locations and N be the NPN associated to J and F. Then, there is a firing sequence of N simulating the reduction sequence of F.*

Proof. When the reduction sequence of F is finite, the result follows from Lemmas 1 and 2. It remains to show that, if there is an infinite sequence of reductions starting from F and st then there is also an infinite firing sequence with N. Note that, all rules of Table 1 reduce the size of the formula w.r.t the number of operations and entities, but the last one. Hence, if there is an infinite reduction sequence from F, there is also an infinite reduction sequence from a protocol call which is a subterm of F. Therefore, we may assume that $F = F_1 \diamond [l]pt(...) \diamond F_2 \to^* [l]pt(...) \diamond F_2$ and $[l]pt(...)$ leads to an infinite reduction sequence. Since $F_1 \to^! c$, we use Lemma 1 to obtain a firing sequence of N till the creation of the net token corresponding to $[l]pt(...)$. Using induction on the marking structure we obtain an infinite firing sequence corresponding to the reduction sequence of $[l]pt(...)$. From that firing sequence, we construct an infinite sequence of autonomous steps for N which completes the proof. □

Proposition 2. *Soundness is decidable for JamSession interactions.*

Proof. The coverability tree for a NPN with autonomous elements is constructed in [9] as follows. The nodes of the tree are labeled with markings of N. The root of the tree is labeled as I_0 and any internal node labeled by M has a child node labeled M' for each M' s.t. $M[\rangle M'$. The leaves of the tree are classified as final (dead markings), covering (markings leading to infinite cycles) and iterative (markings leading to infinite recursion). A node labeled as M' is called covering if it has an ancestor labeled as M s.t. $M \preceq M'$, where $\preceq$ is a quasi-ordering based on the tree structure of the markings. A node labeled as M' is called iterative if it has an ancestor labeled as M s.t. both markings are obtained from the firing of a transition t that generates the same net token, and the last token is nested in the first one. The net is terminating if all leaves are final.

The extension introduced in Definition 1 does not affect the tree structure of the markings. This is due to the fact that the shared places belong to SN and no net token is created for this net component. Therefore, for these nets, the quasi-ordering $\preceq$ and the covering nodes can be defined as in [9]. Nevertheless, a further condition is required in order to ensure that an iterative node leads to an infinite recursive sequence. Since the transition t may have shared places as input, we should also demand that M' covers the marking of the shared places in M, i.e. $M(P_s) \preceq M'(P_s)$. This relation can be effectively computed for places of basic type (or even for multi-level nets). Therefore, the coverability tree is finite. In order to decide the soundness of Definition 3 it is enough to check that all leaves of the tree are labeled by markings with a single token at o and the remaining places empty, except SGL. These markings are dead because o is a sink and there is no transition in N having SGL as the only input place. □

References

1. Bandini, S., Manzoni, S., Vizzari, G.: Multi-agent approach to localization problems: the case of multilayered multi-agent situated system. Web Intell. Agent. Syst. **2**(3), 155–166 (2004)
2. Chang, L., He, X., Shatz, S.M.: A methodology for modeling multi-agent systems using nested Petri nets. Int. J. Softw. Eng. Knowl. Eng. **22**(7), 891–925 (2012)
3. Corrêa da Silva, F.S.: Knowledge-based interaction protocols for intelligent interactive environments. Knowl. Inf. Syst. **30**, 1–24 (2012)
4. Corrêa da Silva, F.S., Venero, M.L.F., David, D.M., Saleemb, M., Chung, P.W.H.: Interaction protocols for cross-organisational workflows. Knowl. Based Syst. **37**, 121–136 (2013)
5. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Garcia, P., Arcos, J.L.: On the formal specification of electronic institutions. In: Sierra, C., Dignum, F.P.M. (eds.) AgentLink 2000. LNCS (LNAI), vol. 1991, pp. 126–147. Springer, Heidelberg (2001)
6. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer, Heidelberg (1992)
7. Köhler, M., Moldt, D., Rölke, H.: Modelling mobility and mobile agents using nets within nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 121–139. Springer, Heidelberg (2003)
8. Lomazova, I.A., Schnoebelen, P.: Some decidability results for nested Petri Nets. In: Bjorner, D., Broy, M., Zamulin, A.V. (eds.) PSI 1999. LNCS, vol. 1755, pp. 208–220. Springer, Heidelberg (2000)
9. Lomazova, I.A.: Recursive nested Petri nets: analysis of semantic properties and expessibility. Program. Comput. Softw. **27**(4), 183–193 (2001)

10. Lomazova, I.A.: Modeling dynamic objects in distributed systems with nested Petri nets. Fundam. Informaticae **51**(1–2), 121–133 (2002)
11. Robertson, D.: Multi-agent coordination as distributed logic programming. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 416–430. Springer, Heidelberg (2004)
12. van der Aalst, W.M.P.: Interorganizational workflows: an approach based on message sequence charts and Petri nets. Syst. Anal. Model. Simul. **34**(3), 335–367 (1999)
13. Fernández Venero, M.L., Corrêa da Silva, F.S.: On the use of SPIN for studying the behavior of nested Petri Nets. In: Iyoda, J., de Moura, L. (eds.) SBMF 2013. LNCS, vol. 8195, pp. 83–98. Springer, Heidelberg (2013)