Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents[1]

[1, 2] *R.A. Nesterov <rnesterov@hse.ru, r.nesterov@campus.unimib.it>*
[1] *A.A. Mitsyuk <amitsyuk@hse.ru>*
[1] *I.A. Lomazova <ilomazova@hse.ru>*
[1] *National Research University Higher School of Economics,*
20, Myasnitskaya st., Moscow, 101000, Russia
[2] *Dipartimento di Informatica, Sistemistica e Communicazione,*
Università degli Studi di Milano-Bicocca,
Viale Sarca 336 – Edificio U14, I-20126 Milano, Italia

Abstract. In this paper, we present an approach to model and simulate models of multi-agent systems (MAS) using Petri nets. A MAS is modeled as a set of workflow nets. The agent-to-agent interactions are described by means of an interface. It is a logical formula over atomic interaction constraints specifying the order of inner agent actions. Our study considers positive and negative interaction rules. In this work, we study interfaces describing acyclic agent interactions. We propose an algorithm for simulating the MAS with respect to a given interface. The algorithm is implemented as a ProM 6 plug-in that allows one to generate a set of event logs. We suggest our approach to be used for evaluating process discovery techniques against the quality of obtained models since this research area is on the rise. The proposed approach can be used for process discovery algorithms concerning internal agent interactions of the MAS.

Keywords: Petri nets; multi-agent systems; interaction; interface; simulation; event logs.

DOI: 10.15514/ISPRAS-2018-30(3)-20

For citation: Nesterov R.A., Mitsyuk A.A., Lomazova I.A. Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 285-302. DOI: 10.15514/ISPRAS-2018-30(3)-20

1. Introduction

Process discovery has been actively developed over recent years [1]. Many algorithms for the automatic model synthesis from event logs have been proposed [2]–[7]. They produce process models in different notations. These can be Petri nets [3], [6], [7], fuzzy models [2], heuristics nets [4] or BPMN models [5] and many others (see [8] for the comprehensive review of process discovery algorithms).

Discovering process models from event logs helps to use information about users and system runtime behavior for proper specification, design, and maintenance of software systems [9], [10]. This topic is increasingly attracting the attention of researchers [11]–[14]. In particular, application of process mining techniques to distributed and multi-agent software systems [15], [16] is interesting and important.

The main drawback of most algorithms is that they are not appropriate for modeling highly concurrent systems. In particular, these are multi-agent systems (MAS). Such a system consists of multiple agents executing their work independently and interacting via predefined interfaces. It makes sense to use compositional approaches to model MAS's. Fortunately, such approaches have been proposed over recent years [17], [18].

The overwhelming majority of process discovery algorithms employ different heuristics. That is why testing is used to evaluate their efficiency and validity [8]. It is performed using real-life and artificially generated event logs with suitable characteristics. The latter are prepared using *event log generators*.

In this paper, we describe a new event log generator that aims at preparing artificial event logs for MAS's. We model individual agents using workflow nets, whereas interfaces are specified using special formulae. They are constructed using a declarative formalism that we introduce to describe basic asynchronous interactions between agents. Based on agent models and a declarative interface formula our generator derives the operational semantics that describes a MAS behavior. We show that both of MAS representations are equivalent, i.e. they have the same set of possible model runs. Thus, this semantics can be used to simulate the model and generate event logs.

The *main contributions of this paper* are:

- a formalism for a declarative description of the requirements for agent interactions is defined;
- the operational semantics representing the behavior of a multi-agent system with declarative requirements for interactions of agents is defined;
- an algorithm for generating event logs from given agent models and declarative constraints on their interactions based on the operational semantics is developed;
- the approach is implemented as a prototype software and evaluated.

This paper is structured as follows. The next section gives an overview of existing approaches for generating event logs and simulating process models. Section 3 introduces main notions used in the paper. In Section 4, we describe our approach to modeling multi-agent systems with the help of Petri nets. Implementation details are discussed in Section 5, and Section 6 concludes the paper.

[1]This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

2. Related Work

Process Logs Generator PLG2 [19] is one of the most popular tools for generating well-structured process models represented by dependency graphs. The tool constructs models using randomly generated context-free grammars. The user should specify desired characteristics of models: a size, a number of choices, hierarchy blocks etc. The obtained model can be used to generate an event log.

Another tool that aims at randomized event log generation is *PT and Log Generator* [20]. It generates random process trees (well-structured models) containing desired number of specified workflow patterns. In particular, generated models can be constructed from sequences, AND/XOR/OR splits and joins, as well as structured loops. The algorithm can also randomly insert elements representing activities. The tool also generates the desired number of logs from automatically constructed models.

The problem of the randomized process model generation has also been addressed by Yan, Dijkman, and Grefen in [21]. However, they have not considered event log generation within the context of their approach.

The main goal of the tools discussed above is the randomized testing using sets of models and event logs. However, in some cases there is a need to generate event logs from specific process models that have been prepared on the basis of the real data or expert knowledge. If this is the case, one can use the tool *GENA* [22]. It aims at generating sets of event logs from a Petri net model. The approach allows users to use preferences to influence a control-flow and to artificially introduce a randomized noise into an event log. The improved version of GENA can generate event logs from BPMN 2.0 models [23]. Most basic BPMN constructs are supported: tasks, gateways, messages, pools, lanes, data objects.

Colored Petri nets can be used to generate event logs [24]. Authors have developed the extension for *CPN Tools* that can generate randomized event logs based on a given colored Petri net. The main drawback of this approach is that it implies writing Standard ML scripts, which leads to possible problems during tool adaptation for a specific task. Moreover, this approach and GENA do not support multi-agent systems with independent asynchronous agents.

Declarative process models might also be used to generate event logs [25]. This approach is based on construction of a finite automaton using a *Declare* process model. The tool can generate a specified number of strings accepted by this automaton. Strings are generated using the automaton and its randomized execution. Afterwards, each string is transformed into a log trace with necessary attributes. This tool is useful, when the only information about the process is the set of constraints. This approach is also not appropriate for the MAS simulation as we suggest, because it does not support the imperative control-flow description of individual agents.

In this paper, we propose an extension to the GENA tool that is supposed to be used for generating event logs by simulating MAS models, because the tools described above cannot fully support this feature.

3. Preliminaries

Let $\mathbb{N}$ denote the set of all non-negative integers, A^+ – the set of all finite non-empty sequences over a set A, and $A^* = A^+ \cup \{\varepsilon\}$, where ε is the empty sequence. For a subset $B \subset A$, the projection of $\sigma \in A^*$ on a set B, denoted $\sigma|_B$, is the subsequence of σ including all elements belonging to B.

3.1 Petri Nets

A *Petri net* is a triple $N = (P, T, F)$, where P and T are two disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation. Pictorially, places are shown by circles, transitions – by boxes, whereas the flow relation is depicted using directed arcs (see Fig. 1 for an example).

We suppose that transitions of a Petri net are labeled with activity names from $\mathcal{A} \cup \{\tau\}$, where $\mathcal{A}$ is a set of visible activity names, and τ is a label for an invisible action. Labels are assigned to transitions via a labeling function $\lambda: T \to \mathcal{A} \cup \{\tau\}$.

A marking (*state*) of a Petri net N is a function $m: P \to \mathbb{N}$ assigning numbers to places. A marking m is designated by putting $m(p)$ black dots into each place p. By m_0 we denote the initial marking.

Let $X = P \cup T$. For $x \in X$, ${}^\bullet x = \{y \in X \mid (y, x) \in F\}$ is the set of *input* nodes of x in N, and $x^\bullet = \{y \in X \mid (x, y) \in F\}$ is the set of its *output* nodes.

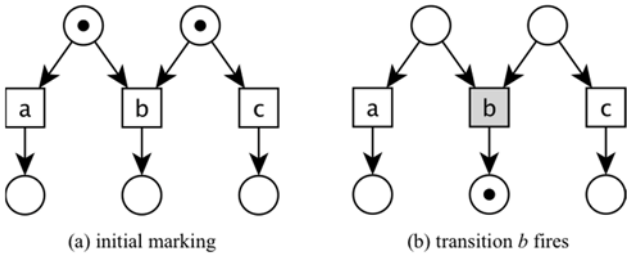


Fig. 1. A Petri net

A marking m *enables* a transition $t \in T$ iff there is at least one token in all places which are input for t. An enabled transition may *fire* yielding a new marking m' (denoted $m[t\rangle m'$), consuming one token from each of its input places and producing a token into each of its output places (see Fig. 1b).

A sequence $w = t_1 t_2 ... t_n$ over T is a *firing sequence* iff $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m_n$ (denoted $m_0[w\rangle m_n$).

Let $w = t_1 t_2 ... t_n$ be a firing sequence of the net N, λ – a labeling function over a set of activity names $\mathcal{A}$. Define $\lambda(w) = \lambda(t_1)\lambda(t_2)\ldots\lambda(t_n)$. Then $\lambda(w)|_{\mathcal{A}}$ is called an (observable) *run* in N.

A marking m is *reachable* iff $\exists\, w \in T^*$: $m_0[w\rangle m$. A reachable marking is called *dead* if it does not enable any transition.

Workflow nets (WF-nets) form a subclass of Petri nets used for business process modeling. A *Petri net* is a triple $N = (P, T, F, m_0)$ is a WF-net iff:

- there is a single source place i and a single source place f, s.t. ${}^\bullet i = f^\bullet = \varnothing$;
- each node in $P \cup T$ lies on a path from i to f.

The initial marking m_0 of a WF-net contains exactly one token in its source place i.

3.2 Event Logs

A *multiset* over a set A is a map $B: A \to \mathbb{N}$. The set of all multisets over A is denoted by $\mathcal{B}(A)$.

Let $\mathcal{A}$ be a set of activity names. A *trace* σ over $\mathcal{A}$ is defined as a finite non-empty sequence over $\mathcal{A}$, i.e. $\sigma \in \mathcal{A}^+$. An *event log* L over $\mathcal{A}$ is a finite multiset of traces, i.e. $L \in \mathcal{B}(\mathcal{A}^+)$.

4. Modeling Multi-Agent Systems

In this section, we present formalism for modeling multi-agent systems consisting of several asynchronously interacting agents.

A model for a system of k agents will consist of k WF-nets $N_1, N_2, \ldots, N_k$, representing behavior of individual agents (called *agent nets*), and constraints on their asynchronous interaction $\mathcal{I}$ (called *interface*).

We assume that transitions of agent nets have individual labels. In other words, different agents implement different activities. We also assume that agent interactions are *acyclic*, namely, activities in interaction constraints do not belong to cycles and therefore occur in each system run not more than once.

Interfaces are defined as positive logical formulae over atomic constraints. Let us give the exact definitions.

Let $N_1, N_2, \ldots, N_k$ be agent nets with pairwise disjoint sets of activity names $\lambda_1(T_1)$, $\lambda_2(T_2), \ldots, \lambda_k(T_k)$ respectively. We define two types of *atomic constraints*, namely $A \triangleleft B$ and $A \overline{\triangleleft} B$, where A and B are activity names from two different sets, i.e. $A \in \lambda_i(T_i)$, $B \in \lambda_j(T_j)$ and $i \neq j$.

The *validity* of atomic constraints for a given trace σ over the set of activity names $\mathcal{A} = \lambda_1(T_1) \cup \lambda_2(T_2) \cup \ldots \cup \lambda_k(T_k)$ is defined as follows:

- $\sigma \vDash A \triangleleft B \Leftrightarrow$ if B occurs in σ, then A occurs before B;
- $\sigma \vDash A \overline{\triangleleft} B \Leftrightarrow$ if A does not occur before B in σ.

When $\sigma \vDash \phi$, we say that ϕ is *valid* for σ, and σ *satisfies* ϕ. The validity of the atomic constraints has a natural interpretation.

The constraint $A \triangleleft B$ means that B should be always preceded by A, e.g. a message can be received only if it has already been sent. Thus, $A \triangleleft B$ is valid for a trace $\sigma = \ldots A \ldots B \ldots$ and is not valid for a trace $\sigma = \ldots <except\ A> \ldots B \ldots$ The constraint $A \overline{\triangleleft} B$ means that B cannot occur if A has happened before, e.g. if a message has been

already sent by mail, we should not fax it again. A trace $\sigma = \ldots <except\ A> \ldots B \ldots$ satisfies this constraint, and a trace $\sigma = \ldots A \ldots B \ldots$ does not satisfy it. However, these atomic constraints are not negations of each other. Both $A \triangleleft B$ and $A \overline{\triangleleft} B$ are valid for a trace that does not contain B.

Now a language of interface constraints is defined by the following grammar rules:

$$Atom ::= A \triangleleft B \mid A \overline{\triangleleft} B,$$
$$\phi ::= Atom \mid \phi \vee \phi \mid \phi \wedge \phi,$$

where *Atom* is an atomic constraint, and ϕ is a constraint formula.

Validity of a constraint formula φ for a given trace σ is defined in a standard way:

$$\sigma \vDash \phi_1 \wedge \phi_2 \Leftrightarrow \sigma \vDash \phi_1 \text{ and } \sigma \vDash \phi_2,$$
$$\sigma \vDash \phi_1 \vee \phi_2 \Leftrightarrow \sigma \vDash \phi_1 \text{ or } \sigma \vDash \phi_2.$$

Let L be an event log over a set $\mathcal{A}$ of activity names, and ϕ be a constraint formula, then ϕ is valid for L iff ϕ is valid for each trace in L.

Interface formulae allow us to express different useful interaction constraints, e.g. the formula $\phi = A \overline{\triangleleft} B \wedge B \overline{\triangleleft} A$ describes a *conflict* between A and B, i.e. A and B cannot occur in the same trace.

Recall that a MAS model consists of k agent nets $N_1, N_2, \ldots, N_k$, where $N_i = (P_i, T_i, F_i, m_0^i, \lambda_i)$, and a constraint formula $\mathcal{I}$ (*interface*) with atomic constraints that defines the relations on activities of different agents.

It is easy to see that the *union* of Petri nets (considering several disjoint graphs as one disconnected graph) is also a Petri net. Thus, we can consider k agent nets as a single Petri net N. Recall that a run for a Petri net N is a sequence of activity names, corresponding to a firing sequence of N, and a trace from the related event log. Then a run of a MAS model $S = (N_1, N_2, \ldots, N_k, \mathcal{I})$ is defined as a run ρ in N satisfying $\mathcal{I}$, i.e. $\rho \vDash \mathcal{I}$.

The following proposition is the immediate consequence of the definitions.

Proposition 1: Let $S = (N_1, N_2, \ldots, N_k, \mathcal{I})$ be a MAS model, and ρ be a run in S. Then for all i the projection $\rho|_{\lambda_i(T_i)}$ on transition labels of an agent net N_i is a run in N_i.

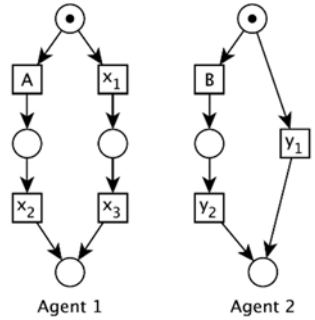


Fig. 2. A multi-agent system with two interacting systems

Consider as an example the system shown in Fig. 2 with $\mathcal{J} = A \overline{\triangleleft} B \wedge B \overline{\triangleleft} A$ meaning that A conflicts with B. Consider a run $\sigma = x_1 B y_2 x_3$ satisfying $\mathcal{J}$. Projecting σ on agent nets gives traces $x_1 x_3$ and $B y_2$, which are runs of the corresponding agent nets. This property will be further used for designing the simulation algorithm presented in the next section.

5. Simulating MAS Process Models

In this section, we describe an algorithm for simulating MAS models. It has been implemented as a ProM 6 plug-in extending GENA tool [22].

5.1 An Interface-Driven Firing Rule

A constraint formula in a MAS model defines *declarative* restrictions on the model behavior. To simulate the model behavior, we need to define operational semantics for MAS models based on a special firing rule for selecting and executing the next step in the run of the model. We call this rule an *interface-driven firing rule* to distinguish it from the standard Petri net firing rule. Naturally, this rule should be consistent with the declarative definitions of MAS model behavior.

Let $S = (N, \mathcal{J})$ be a MAS model, where a Petri net $N = (P, T, F, m_0, \lambda)$ is a union of all agent nets.

Firstly, we convert $\mathcal{J}$ to a disjunctive normal form (DNF) using standard logical laws. Then, an interface $\mathcal{J} = \bigvee C_j$ for $j = 1, 2, \ldots, n$, where $C_j = \bigwedge S_l$, and S_l is an atomic constraint for $l = 1, 2, \ldots, m$. By abuse of notation, we also denote by $\mathcal{J}$ the set of its conjuncts, and by C_j – the set of atomic constraints in a conjunct C_j.

Obviously, a trace σ satisfies $\mathcal{J}$ iff $\exists\, C_j \in \mathcal{J}$: $\sigma \vDash C_j$, i.e. it should satisfy at least one conjunct in $\mathcal{J}$. Thus, to generate a model run, we choose a conjunct C_j and fire transitions of N only if they do not violate C_j.

Then we define $T_{\mathcal{J}} \subseteq T$ to be the set of transitions involved in agent interaction, i.e. $t \in T_{\mathcal{J}}$ iff $\lambda(t)$ occurs in $\mathcal{J}$. We call transitions from $T_{\mathcal{J}}$ *interface* transitions. Independent transitions from $T \backslash T_{\mathcal{J}}$ fire according to the standard firing rule for Petri nets. The firing of interface transitions is restricted by the constraint formula. To check whether firing of a transition t violates C_j, we keep the current historical model run, i.e. a sequence of already fired activities. When a transition $t \in T_{\mathcal{J}}$ is enabled according to the standard Petri net firing rule at a current marking m, and an atomic constraint $A \triangleleft \lambda(t)$ occurs in C_j, then t is defined to be enabled only if A occurs in the current run. Similarly, if $A \overline{\triangleleft} \lambda(t)$ occurs in C_j, then t is enabled only if A does not occur in the current run. Otherwise, a transition t is enabled in the model, when it is enabled in N.

Now the *operational semantics* of a MAS model $S = (N, \mathcal{J})$, where $N = (P, T, F, m_0, \lambda)$ and $\mathcal{J} = \bigvee C_j$ for $j = 1, 2, \ldots, n$, is defined by the following procedure.

Step 1. Choose nondeterministically a conjunct C in $\mathcal{J}$.

Step 2. Start with the initial marking m_0 and ε for the current run σ.

Step 3. For a current marking m and a current run σ repeat while there are enabled transitions in N:

1) compute the set T_{ok} of all transitions enabled at m and not violating constraints from C w.r.t. σ;
2) choose nondeterministically a transition t from T_{ok};
3) fire t by changing the current marking to m', $m[t\rangle m'$, and adding $\lambda(t)$ to σ.

5.2 Event Log Generation

This subsection presents an algorithm for generating an event log by simulating behavior of a MAS model.

Let $S = (N, \mathcal{J})$ be a MAS model, where $N = (P, T, F, m_0, \lambda)$ is a Petri net, and $\mathcal{J}$ is in DNF. Firstly, for each conjunct C occurring in $\mathcal{J}$, we run (*simulate*) S to check if it is possible to obtain a trace σ satisfying C. If we cannot obtain such a trace, we exclude this conjunct. As a result, we come to a set of conjuncts $\mathcal{J}' \subseteq \mathcal{J}$, which can be actually satisfied by traces of S or an empty set if $\mathcal{J}$ cannot be satisfied by traces of S. If $\mathcal{J}' = \varnothing$, then the simulation is terminated producing an empty event log L.

That is why we can simulate S w.r.t. conjuncts occurring in $\mathcal{J}'$ only. Starting a new *iteration* of simulation, we randomly choose a conjunct from $\mathcal{J}'$ and fire transitions of N according to the interface-driven firing rule.

The end user specifies the final marking m_f, which is actually the set of sink places of agent nets. Apart from that, the log generation is regulated by the number of logs, the number of traces in a log, and by the maximum number of steps which can be executed while generating a single trace (denoted further by *maxSteps*).

Algorithm 1 is used for generating a single trace that satisfies C from $\mathcal{J}'$.

Algorithm 1. Single trace generation

```
Input: N = (P, T, F, m0, λ), J', and mf
Output: a trace σ, s.t. σ ⊨ J'
σ ← ε; m ← m0; i ← 1; C ← pickRandomConjunct(J')
while (i ≤ maxSteps) ∧ (m ≠ mf) do
    Tok ← findEnabledTransitions(N, m, C, σ)
    if Tok ≠ ∅ then
        t ← pickRandomTransition(Tok)
        m ← fireTransition(N, m, t)
        if λ(t) ≠ τ then
            σ ← σ + λ(t); i ← i + 1
        end
    else
        σ ← ε; break
    end
end
```

Algorithm 2 is used for finding enabled transitions, which do not violate constraints of C. Firstly, we find a set of transitions enabled at a reachable marking m according

to the standard firing rule. Secondly, if m enables interface transitions, we check whether the current run $\sigma = \lambda(w)|_{\mathcal{A}}$, s.t. $m_0[w\rangle m$, satisfies constraints of C using the interface-driven firing rule. A run σ is a trace to be recorded into an event log L.

Algorithm 2. Function findEnabledTransitions

```
Input: N = (P, T, F, m0, λ), m ∈ [m0⟩, C ∈ ℑ', σ
Output: a set Tok of transitions enabled w.r.t. C
Tm ← stEnabledTransitions(N, m)
Tok ← Tm \ Tℑ
foreach t ∈ Tm ∩ Tℑ do
    foreach S ∈ C do
        if S = X ◁ λ(t) then
            if σ = uXv then Tok ← Tok ∪ t
        else if S = X ◁̄ λ(t) then
            if σ≠uXv then Tok ← Tok ∪ t
        end
    end
end
```

We do not show here how the transition firing is implemented. It is discussed in detail in [22] where the original GENA plug-in is described.

Consider an example based on the system shown in Fig. 2. Assume $\mathcal{I} = (A \triangleleft B) \vee (y_1 \triangleleft x_1 \wedge x_2 \mathbin{\bar{\triangleleft}} y_1)$. $C = y_1 \triangleleft x_1 \wedge x_2 \mathbin{\bar{\triangleleft}} y_1$ is chosen. We are at the initial marking, i.e. $\sigma = \varepsilon$. Enabled transitions are $\{A, x_1, B, y_1\}$. However, x_1 cannot fire, since it should wait until y_1 is executed. Then B fires nondetermenistically. Subsequently, the run is $\sigma = B$, and the enabled transitions are $\{A, x_1, y_2\}$, but x_1 still cannot fire. We can choose A to fire. Then the run is $\sigma = BA$, and the enabled transitions are $\{x_2, y_2\}$ firing of which is not influenced by C. As a result, we can obtain a trace $\sigma = BAy_2x_2$ satisfying C, and the projections of σ on agent transitions, Ax_2 and By_2, are the runs of corresponding agent nets.

5.3 Experimental Simulation

We have developed the extension to the ProM[2] plug-in GENA implementing the proposed simulation algorithm and allowing users to obtain a set of event logs by simulating a given MAS model w.r.t. interaction constraints.

We have prepared five use cases for evaluating the proposed simulation approach. In each case, we have generated event logs with 5000 traces. In addition, we provide a "filtered" version of a generated event log w.r.t. interacting actions, s.t. it is clear whether the corresponding interface is exactly observed.

We have used Disco[3] to visualize generated event logs. Insignificant parts of agent nets are shown by shaded ovals.

[2] ProM 6 Framework page: http://www.promtools.org

[3] Fluxicon Disco page: https://fluxicon.com/disco/

a) Sequencing: Consider a system with three interacting agents (see Fig. 3). Each agent always executes one action. We have simulated it w.r.t. the interface $\mathcal{I} = A \triangleleft B \wedge B \triangleleft C$. Intuitively, in this case each interacting agent prepares resources needed for the other agent.

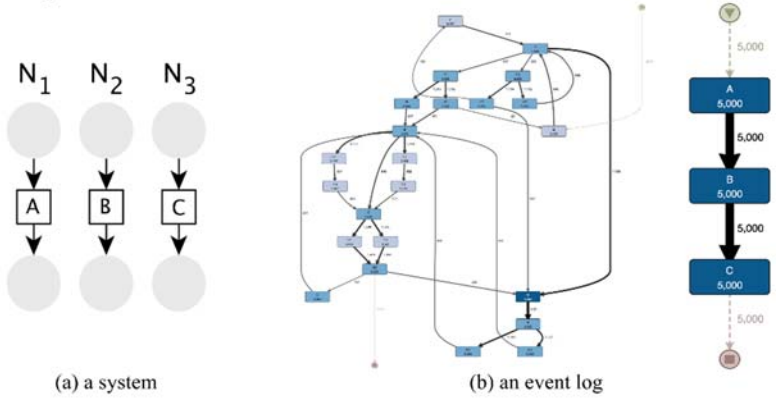


Fig. 3. Sequential interaction

b) Conditional sequencing: As opposed to sequencing, conditional sequencing allows for several execution options. In this case, a system consists of two agents, one of which has two alternative branches (see Fig. 4). The interface for the conditional sequencing is as follows: $\mathcal{I} = A \triangleleft C \vee C \triangleleft B$.

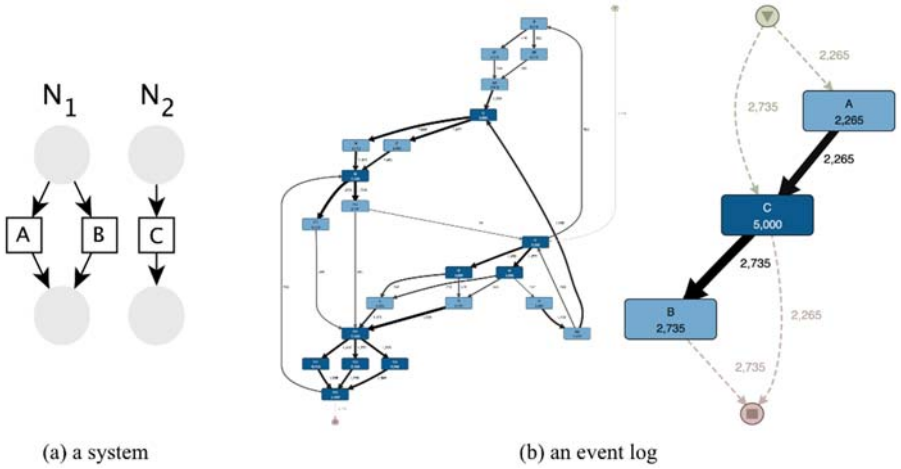
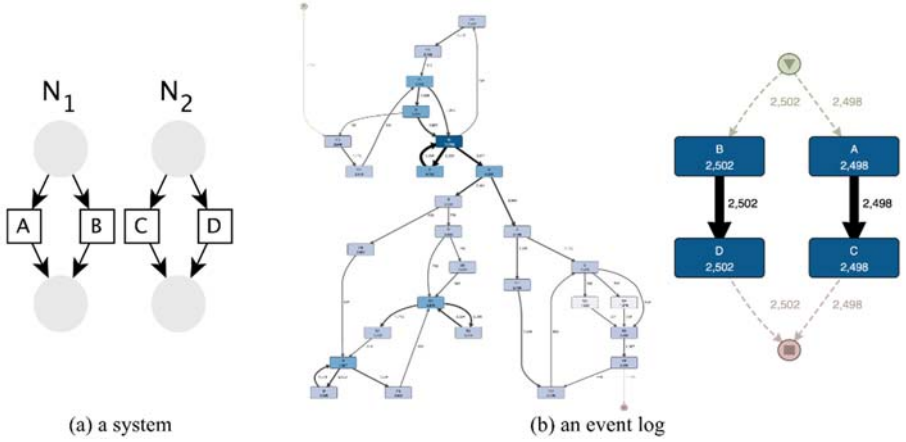


Fig. 4. Sequential interaction with options

c) Alternative interaction: The alternative interaction implies that one of two interacting agents influences the choice done by the other agent. A system consists of

two interacting agents both having two alternative branches (see Fig. 5). The interface formula for this case is as follows: $\mathcal{I} = A \triangleleft C \vee B \triangleleft D$.



(a) a system (b) an event log

Fig. 5. Alternative interaction

d) Interaction using negative constraints: Assume we have a system of two interacting agents with two alternative branches as shown in Fig. 5a. The result of simulating this system w.r.t. the interface $\mathcal{I} = A \overline{\triangleleft} C$ is shown in Fig. 6. It is clear from the simulation result that C is never preceded by A. Intuitively, negative constraints allow for a more compact way of interface construction.

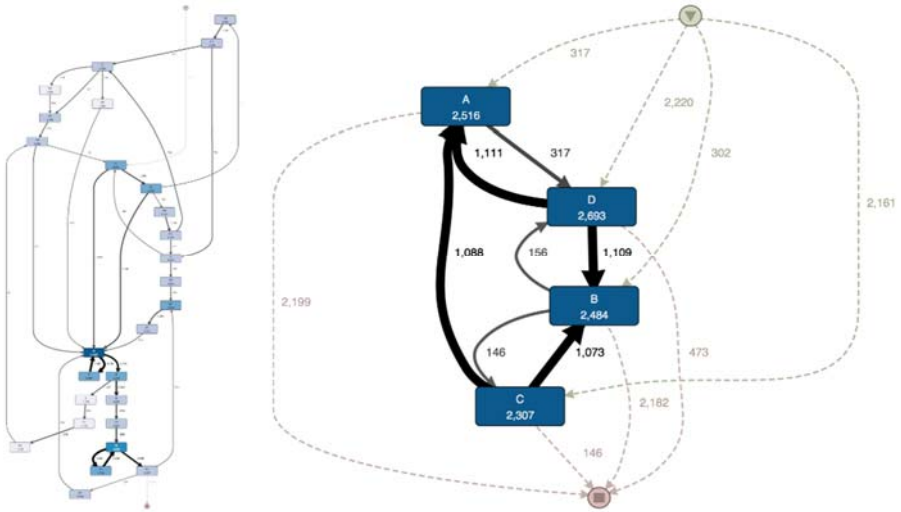
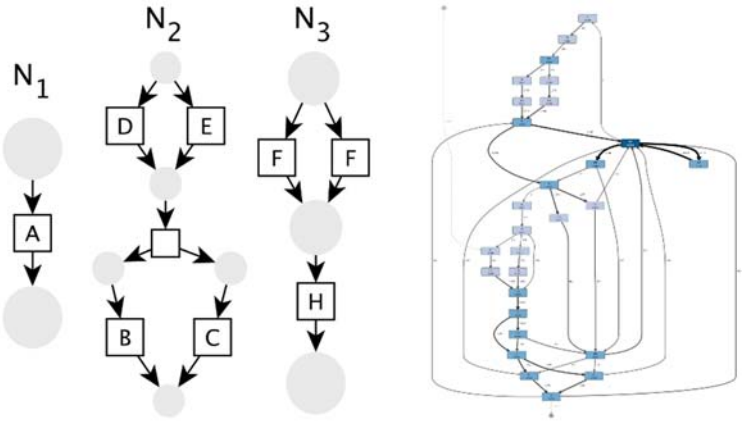


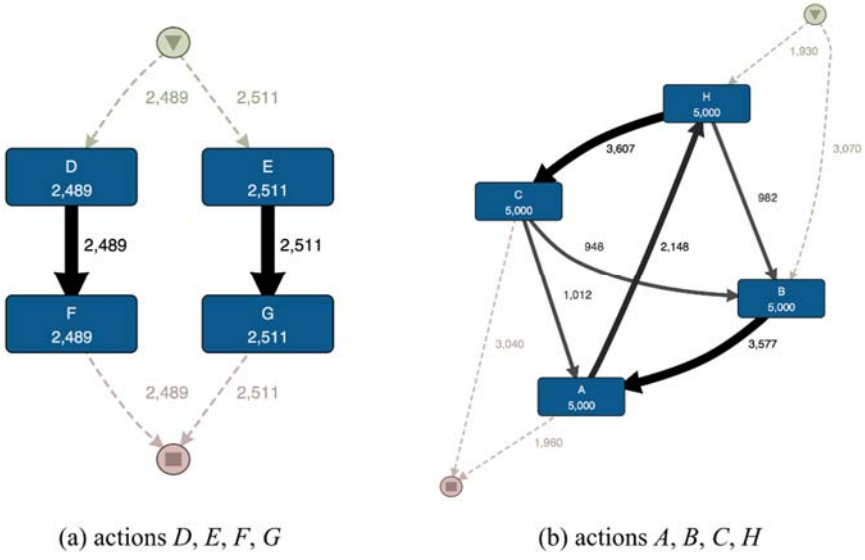
Fig. 6. Interaction using negative constraints: an event log

e) Complex interaction: In this case, we show several ways of interaction among three different agents (see Fig. 7a). For convenience, we have filtered the obtained log in two ways (see Fig. 8). We have used the following interface formula (given in a conjunctive normal form for the convenience of a reader): $\mathcal{I} = B \triangleleft A \wedge H \triangleleft C \wedge (D \triangleleft F \vee E \triangleleft G)$.



(a) a system (b) a full event log

Fig. 7. Complex interaction



(a) actions D, E, F, G (b) actions A, B, C, H

Fig. 8. Complex interaction: filtered event logs

6. Conclusion

We have proposed the new approach to model and simulate multi-agent systems using Petri nets. Independent agents are modeled as a set of labeled workflow nets, and their interaction is described using a declarative interface. The interface is constructed as a logic formula over atomic constraints describing the order of internal agent actions. This study has considered only acyclic agent interactions described by two kinds of atomic constraints, s.t. interacting activities are implemented only once. If cyclic interactions are allowed, subtler relations on interacting activities are needed to express such constraints as "*each B* should be preceded by *A*" or "*at least one B* should be preceded by *A*". This is a subject for further research.

An algorithm for simulating process models of multi-agent systems with respect to the interface has been developed. We have implemented the algorithm within the existing ProM 6 plug-in GENA and have evaluated it using five different cases of agent interactions. The experiment results show how to obtain artificial event logs by simulating process models of multi-agent systems with a finite number of asynchronously interacting agents.

References

[1]. van der Aalst W.M.P. Process Mining – Data Science in Action. Springer, Heidelberg, 2016, 467 p.

[2]. Günther C.W., van der Aalst W.M.P. Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. BPM 2007. LNCS, vol. 4714, 2007, pp. 328-343.

[3]. van der Werf J.M.E.M., van Dongen B.F., Hurkens C.A.J., Serebrenik A. Process Discovery using Integer Linear Programming. Fundamenta Informaticae, vol. 94, no. 3-4, 2009, pp. 387-412.

[4]. Weijters A.J.M.M., Ribeiro J.T.S. Flexible Heuristics Miner (FHM). In Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310-317.

[5]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. ICATPN 2014. LNCS, vol. 8489, 2014, pp. 71-90.

[6]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Scalable Process Discovery with Guarantees. BPMDS 2015, EMMSAD 2015. LNBIP, vol 214, 2015, pp. 85-101.

[7]. Begicheva A.K., Lomazova I.A. Discovering high-level process models from event logs. Modeling and Analysis of Information Systems, vol. 24, no. 2, 2017, pp. 125–140..

[8]. Augusto A., Conforti R., Dumas M., La Rosa M., Maria Maggi F., Marrella A., Mecella M., Soo A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. CoRR, 2017, vol. abs/1705.02288.

[9]. Rubin V.A., Mitsyuk A.A., Lomazova I.A., van der Aalst W.M.P. Process Mining can be applied to software too! In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14), 2014, pp. 1-8.

[10]. Leemans M., van der Aalst W.M.P. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. MODELS 2015, pp. 44-53.

[11]. Leemans M., van der Aalst W.M.P., van den Brand M. Recursion Aware Modeling and Discovery for Hierarchical Software Event Log Analysis (Extended). CoRR, 2017, vol. abs/1710.09323.

[12]. Liu C., van Dongen B.F., Assy N., van der Aalst W.M.P. Component behavior discovery from software execution data. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1-8.

[13]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10.

[14]. 3TU: Big software on the run. [Online]. Available: http://www.3tu-bsr.nl. Accessed: 09.06.2018.

[15]. Cabac L., Denz N. Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, 2008, pp. 86-103.

[16]. Cabac L., Knaak N., Moldt D., Rölke H. Analysis of Multi-Agent Interactions with Process Mining Techniques. MATES 2006. LNCS, vol. 4196, 2006, pp. 12-23.

[17]. Nesterov R.A., Lomazova I.A. Using Interface Patterns for Compositional Discovery of Distributed System Models. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 4, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(4)-2.

[18]. Nesterov R.A., Lomazova I.A. Compositional Process Model Synthesis Based on Interface Patterns. TMPA 2017. CCIS, vol. 779, 2018, pp. 151-162.

[19]. Burattin A. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.

[20]. Jouck T., Depaire B. PTandLogGenerator: A Generator for Artificial Event Data. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.

[21]. Yan Z., Dijkman R.M., Grefen P. Generating process model collections. Software and System Modeling, 2017, vol. 16, issue 4, pp. 979-995.

[22]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. In Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88-95. DOI: 10.15514/SYRCOSE-2014-8-13.

[23]. Mitsyuk A.A., Shugurov I.S., Kalenkova A.A., van der Aalst W.M.P. Generating event logs for high-level process models. Simulation Modelling Practice and Theory, vol. 74, 2017, pp. 1-16.

[24]. de Medeiros A.K.A., Günther C.W. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In Proceedings of CPN 2005. DAIMI, vol. 576, 2005, pp. 177-190.

[25]. Di Ciccio C., Luca Bernardi M., Cimitile M., Maria Maggi F. Generating Event Logs Through the Simulation of Declare Models. EOMAS 2015. LNBIP, vol. 231, 2015, pp. 20-36.

Симуляция поведения мультиагентных систем с ациклически взаимодействующим агентами

[1, 2] *Р.А. Нестеров <rnesterov@hse.ru, r.nesterov@campus.unimib.it>*
[1] *А.А. Мицюк <amitsyuk@hse.ru>*
[1] *И.А. Ломазова <ilomazova@hse.ru>*
[1] *Национальный исследовательский университет «Высшая школа экономики», 101000, Россия, г. Москва, ул. Мясницкая, д. 20.*
[2] *Департамент информатики, систем и коммуникаций, Миланский университет-Бикокка, 20126, Италия, г. Милан, Viale Sarca 336 – Edificio U14*

Аннотация. В работе предложен подход для моделирования и симуляции поведения мультиагентных систем (МАС) с применением сетей Петри. МАС представляется как конечное множество сетей потоков работ. Асинхронные взаимодействия агентов описываются с помощью интерфейса, который определяется логической формулой над множеством атомарных ограничений. Эти ограничения задают порядок выполнения внутренних действий агентов. В статье рассматриваются только ациклические взаимодействия агентов. Также был разработан алгоритм симуляции поведения МАС с учетом ограничений взаимодействия агентов. Алгоритм реализован в виде подключаемого модуля для инструмента ProM 6. Предложенный подход может быть использован для оценки качества алгоритмов извлечения процессов (process discovery) с точки зрения характеристик получаемых моделей процессов.

Ключевые слова: сети Петри; мультиагентные системы; взаимодействие; интерфейс; симуляция; журналы событий

DOI: 10.15514/ISPRAS-2018-30(3)-20

Для цитирования: Нестеров Р.А., Мицюк А.А., Ломазова И.А. Симуляция поведения мультиагентных систем с ациклически взаимодействующими агентами. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 285-302 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-20

Список литературы

[1]. van der Aalst W.M.P. Process Mining – Data Science in Action. Springer, Heidelberg, 2016, 467 p.
[2]. Günther C.W., van der Aalst W.M.P. Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. BPM 2007. LNCS, vol. 4714, 2007, pp. 328-343.
[3]. van der Werf J.M.E.M., van Dongen B.F., Hurkens C.A.J., Serebrenik A. Process Discovery using Integer Linear Programming. Fundamenta Informaticae, vol. 94, no. 3-4, 2009, pp. 387-412.
[4]. Weijters A.J.M.M., Ribeiro J.T.S. Flexible Heuristics Miner (FHM). In Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310-317.

[5]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. ICATPN 2014. LNCS, vol. 8489, 2014, pp. 71-90.
[6]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Scalable Process Discovery with Guarantees. BPMDS 2015, EMMSAD 2015. LNBIP, vol 214, 2015, pp. 85-101.
[7]. Begicheva A.K., Lomazova I.A. Discovering high-level process models from event logs. Modeling and Analysis of Information Systems, vol. 24, no. 2, 2017, pp. 125–140..
[8]. Augusto A., Conforti R., Dumas M., La Rosa M., Maria Maggi F., Marrella A., Mecella M., Soo A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. CoRR, 2017, vol. abs/1705.02288.
[9]. Rubin V.A., Mitsyuk A.A., Lomazova I.A., van der Aalst W.M.P. Process Mining can be applied to software too! In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM ’14), 2014, pp. 1-8.
[10]. Leemans M., van der Aalst W.M.P. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. MODELS 2015, pp. 44-53.
[11]. Leemans M., van der Aalst W.M.P., van den Brand M. Recursion Aware Modeling and Discovery for Hierarchical Software Event Log Analysis (Extended). CoRR, 2017, vol. abs/1710.09323.
[12]. Liu C., van Dongen B.F., Assy N., van der Aalst W.M.P. Component behavior discovery from software execution data. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1-8.
[13]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10.
[14]. 3TU: Big software on the run. [Online]. Available: http://www.3tu-bsr.nl. Accessed: 09.06.2018.
[15]. Cabac L., Denz N. Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, 2008, pp. 86-103.
[16]. Cabac L., Knaak N., Moldt D., Rölke H. Analysis of Multi-Agent Interactions with Process Mining Techniques. MATES 2006. LNCS, vol. 4196, 2006, pp. 12-23.
[17]. Nesterov R.A., Lomazova I.A. Using Interface Patterns for Compositional Discovery of Distributed System Models. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 4, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(4)-2.
[18]. Nesterov R.A., Lomazova I.A. Compositional Process Model Synthesis Based on Interface Patterns. TMPA 2017. CCIS, vol. 779, 2018, pp. 151-162.
[19]. Burattin A. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.
[20]. Jouck T., Depaire B. PTandLogGenerator: A Generator for Artificial Event Data. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.
[21]. Yan Z., Dijkman R.M., Grefen P. Generating process model collections. Software and System Modeling, 2017, vol. 16, issue 4, pp. 979-995.
[22]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. In Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88-95. DOI: 10.15514/SYRCOSE-2014-8-13.

[23]. Mitsyuk A.A., Shugurov I.S., Kalenkova A.A., van der Aalst W.M.P. Generating event logs for high-level process models. Simulation Modelling Practice and Theory, vol. 74, 2017, pp. 1-16.

[24]. de Medeiros A.K.A., Günther C.W. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In Proceedings of CPN 2005. DAIMI, vol. 576, 2005, pp. 177-190.

[25]. Di Ciccio C., Luca Bernardi M., Cimitile M., Maria Maggi F. Generating Event Logs Through the Simulation of Declare Models. EOMAS 2015. LNBIP, vol. 231, 2015, pp. 20-36.