A Hierarchical Markov Model to Understand the Behaviour of Agents in Business Processes

Diogo R. Ferreira[1], Fernando Szimanski[2], and Célia Ghedini Ralha[2]

[1] IST – Technical University of Lisbon, Portugal
diogo.ferreira@ist.utl.pt
[2] Universidade de Brasília (UnB), Brazil
fszimanski@gmail.com, ghedini@cic.unb.br

Abstract. Process mining techniques are able to discover process models from event logs but there is a gap between the low-level nature of events and the high-level abstraction of business activities. In this work we present a hierarchical Markov model together with mining techniques to discover the relationship between low-level events and a high-level description of the business process. This can be used to understand how agents perform activities at run-time. In a case study experiment using an agent-based simulation platform (AOR), we show how the proposed approach is able to discover the behaviour of agents in each activity of a business process for which a high-level model is known.

Keywords: Process Mining, Agent-Based Simulation, Markov Models, Expectation-Maximization, Agent-Object-Relationship (AOR).

1 Introduction

If one would try to understand a business process by observing people at work in an organization, the apparently chaotic nature of events would be quite confusing. However, if a high-level description of the business process is provided (for example, one that partitions the process into two or three main stages) then it becomes much easier to understand the sequence of events.

Using process mining techniques [1], it is possible to analyse the low-level events that are recorded in information systems as people carry out their work. However, there is often a gap between the granularity of the recorded low-level events and the high level of abstraction at which processes are understood, documented and communicated throughout an organisation. Process mining techniques are able to capture and analyse behaviour at the level of the recorded events, whereas business analysts describe the process in terms of high-level activities, where each activity may correspond to several low-level events.

Some techniques have already been proposed to address this gap by producing more abstract models from an event log. Basically, existing approaches can be divided into two main groups:

(a) Those techniques that work on the basis of models, by producing a mined model from the event log and then trying to create more abstract representations of that model. Examples are [2] and [3].

M. La Rosa and P. Soffer (Eds.): BPM 2012 Workshops, LNBIP 132, pp. 150–161, 2013.

(b) Those techniques that work on the basis of events, by translating the event log into a more abstract sequence of events, and then producing a mined model from that translated event log. Examples are [4] and [5].

In this work, we consider that the event log is produced by a number of agents who collaborate when performing the activities in a business process. Here, an agent is understood in the sense of intelligent agent [6], and it is used to represent a human actor in agent-based simulations. Each activity may involve several agents, and every time an agent performs some action, a new event is recorded. Therefore, each activity in the process may be the origin of several events in the event log. The way in which agents collaborate is assumed to be non-deterministic, and process execution is assumed to be non-deterministic as well. While the business process is described in terms of high-level activities that are familiar to business users, the collaboration of agents at run-time is recorded in the event log as a sequence of low-level events.

In general, both the macro-level description of the business process and the micro-level sequence of events can be obtained: the first is provided by business analysts or process documentation, and the second can be found in event logs. However, the way in which micro-level events can be mapped to macro-level activities is unknown, and this is precisely what we want to find out. Given a micro-level event log and a macro-level model of the business process, our goal is to discover: (1) a micro-level model for the behaviour of agents and also (2) how this micro-level model fits into the macro-level description of the business process. For this purpose, we develop a hierarchical Markov model that is able to capture the macro-behaviour of the business process, the micro-behaviour of agents as they work in each activity, and the relationship between the two.

2 An Example

Consider a business process that can be described on a high level as comprising the three main activities A, B, and C. Also, consider that three agents X, Y and Z collaborate in order to perform each activity. In particular, activity A leads to a sequence of actions by the three agents that can be represented by the sequence of events XYZ. In a similar way, activity B leads to a sequence of events in the form YZZ(Z)..., where there may be multiple actions of agent Z until a certain condition holds. Finally, activity C leads to a sequence of events in the form ZXY. This process is represented in Figure 1.

Executing this simple process corresponds to performing the sequence of activities ABC. However, in the event log we find traces such as XYZYZZZXY without having any idea of how this sequence of events can be mapped to the sequence of activities ABC. The sequence ABC will be called the *macro-sequence* and the high-level model for the business process is referred to as the *macro-model*. On the other hand, the sequence of events XYZYZZZXY will be called the *micro-sequence* and the behaviour of agents during each high-level activity is referred to as a *micro-model*. The problem addressed in this work is *how to discover*

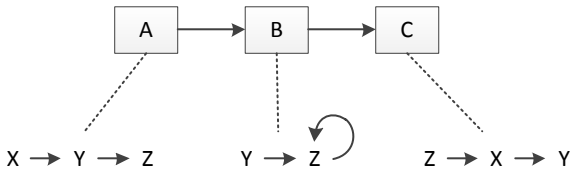


Fig. 1. A simple example of a hierarchical process model

the macro-sequence and the micro-models from a given macro-model and micro-sequence, where these models (both macro and micro) are represented as Markov chains.

3 Definitions

Let $\mathbf{S}$ be the set of possible states in a Markov chain, and let i and j be any two such states. Then $\mathbb{P}(j \mid i)$ is the transition probability from the current state i to a subsequent state j. In this work, as in [7], we extend the set $\mathbf{S}$ with two special states – a start state ($\circ$) and an end state ($\bullet$) – in order to include the probability of the Markov chain starting and ending in certain states. We represent this augmented set of states as $\overline{\mathbf{S}} = \mathbf{S} \cup \{\circ, \bullet\}$. For example, $\mathbb{P}(i \mid \circ)$ is the probability of the Markov chain starting in state i. Similarly, $\mathbb{P}(\bullet \mid i)$ is the probability of the Markov chain ending in state i.

By definition, $\mathbb{P}(\circ \mid i) \triangleq 0, \forall_{i \in \overline{\mathbf{S}}}$ since nothing can come before the start state. In the same way, $\mathbb{P}(i \mid \bullet) \triangleq 0, \forall_{i \in \overline{\mathbf{S}}}$ since nothing can come after the end state. Also, $\mathbb{P}(\bullet \mid \circ) \triangleq 0$ since the Markov chain cannot start and end immediately without going through an observable state.

A Markov chain is represented by a matrix $\mathbf{T} = \{p_{ij}\}$ of transition probabilities, where $p_{ij} = \mathbb{P}(j \mid i), \forall_{i,j \in \overline{\mathbf{S}}}$. More formally, a Markov chain $\mathcal{M} = \langle \overline{\mathbf{S}}, \mathbf{T} \rangle$ is defined as a tuple where $\overline{\mathbf{S}}$ is the augmented set of states and $\mathbf{T}$ is the transition matrix between those states. The nature of the Markov chain is such that $\sum_{j \in \overline{\mathbf{S}}} \mathbb{P}(j \mid i) = 1$ for all states $i \in \overline{\mathbf{S}} \setminus \{\bullet\}$. In other words, there is always some subsequent state to the current state i, except when the end state has been reached; in this case, we have $\sum_{j \in \overline{\mathbf{S}}} \mathbb{P}(j \mid \bullet) = 0$.

The fact that $\forall_{j \in \overline{\mathbf{S}}} : \mathbb{P}(j \mid \bullet) = 0$ means that the last row in matrix $\mathbf{T}$ is zero. Also, the fact that $\forall_{i \in \overline{\mathbf{S}}} : \mathbb{P}(\circ \mid i) = 0$ means that the first column in matrix $\mathbf{T}$ is zero. Finally, the fact that $\mathbb{P}(\bullet \mid \circ) = 0$ means that the last element in the first row of the matrix is zero. These facts are illustrated in Figure 2.

In a hierarchical Markov model, there is a Markov chain to describe the macro-model (upper level in Figure 1), and there is a set of Markov chains to describe the micro-model for each activity (lower level in Figure 1).

The macro-model is defined as a Markov chain $\mathcal{M}' = \langle \overline{\mathbf{S}'}, \mathbf{T}' \rangle$ where $\mathbf{S}'$ is the set of states that represent the activities in the high-level description of the business process. On the other hand, the micro-model is defined as a set of

$$\mathbf{T} = \begin{array}{c|ccccc|l} & \circ & 1 & 2 & \ldots & n & \bullet & \\ \hline \circ & \mathbf{0} & p_{01} & p_{02} & \ldots & p_{0n} & \mathbf{0} & (\sum_j p_{0j} = 1) \\ 1 & \mathbf{0} & p_{11} & p_{12} & \ldots & p_{1n} & p_{1(n+1)} & (\sum_j p_{1j} = 1) \\ 2 & \mathbf{0} & p_{21} & p_{22} & \ldots & p_{2n} & p_{2(n+1)} & (\sum_j p_{2j} = 1) \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ n & \mathbf{0} & p_{n1} & p_{n2} & \ldots & p_{nn} & p_{n(n+1)} & (\sum_j p_{nj} = 1) \\ \bullet & \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} & (\sum_j p_{(n+1)j} = 0) \end{array}$$

Fig. 2. General form of a transition matrix

$\mathcal{M}' = \langle \overline{\mathbf{S}'}, \mathbf{T}' \rangle \qquad \overline{\mathbf{S}'} = \{\circ, \mathsf{A}, \mathsf{B}, \mathsf{C}, \bullet\} \qquad \mathbf{T}' =$

	∘	A	B	C	•
∘	0	1	0	0	0
A	0	0	1	0	0
B	0	0	0	1	0
C	0	0	0	0	1
•	0	0	0	0	0

$\mathcal{M}''_{\mathsf{A}} = \langle \overline{\mathbf{S}''_{\mathsf{A}}}, \mathbf{T}''_{\mathsf{A}} \rangle \qquad \overline{\mathbf{S}''_{\mathsf{A}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \mathbf{T}''_{\mathsf{A}} =$

	∘	X	Y	Z	•
∘	0	1	0	0	0
X	0	0	1	0	0
Y	0	0	0	1	0
Z	0	0	0	0	1
•	0	0	0	0	0

$\mathcal{M}''_{\mathsf{B}} = \langle \overline{\mathbf{S}''_{\mathsf{B}}}, \mathbf{T}''_{\mathsf{B}} \rangle \qquad \overline{\mathbf{S}''_{\mathsf{B}}} = \{\circ, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \mathbf{T}''_{\mathsf{B}} =$

	∘	Y	Z	•
∘	0	1	0	0
Y	0	0	1	0
Z	0	0	.5	.5
•	0	0	0	0

$\mathcal{M}''_{\mathsf{C}} = \langle \overline{\mathbf{S}''_{\mathsf{C}}}, \mathbf{T}''_{\mathsf{C}} \rangle \qquad \overline{\mathbf{S}''_{\mathsf{C}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \mathbf{T}''_{\mathsf{C}} =$

	∘	X	Y	Z	•
∘	0	0	0	1	0
X	0	0	1	0	0
Y	0	0	0	0	1
Z	0	1	0	0	0
•	0	0	0	0	0

Fig. 3. An example of a hierarchical Markov model

Markov chains $\{\mathcal{M}''_i : i \in \mathbf{S}'\}$ where $\mathcal{M}''_i = \langle \overline{\mathbf{S}''_i}, \mathbf{T}''_i \rangle$ is a Markov chain that describes the behaviour of agents when performing activity $i \in \mathbf{S}'$.

For the example in Figure 1, one possible model is shown in Figure 3. In $\mathbf{T}''_{\mathsf{B}}$ it is assumed that the probability of going from state Z to the same state Z is equal to the probability of terminating the Markov chain in that state. The execution semantics for a hierarchical Markov model can be described as follows:

(a) Run the macro-model $\mathcal{M}' = (\overline{\mathbf{S}'}, \mathbf{T}')$ as Markov chain, beginning with the start state ($\circ$) and generating a number of transitions according to the transition probabilities in $\mathbf{T}'$, until the end state ($\bullet$) is reached.
(b) For each macro-state i that the macro-model $\mathcal{M}'$ goes into, run the corresponding micro-model $\mathcal{M}''_i$ as a Markov chain, again beginning with the start state ($\circ$) and generating a number of transitions according to the transition probabilities in $\mathbf{T}''_i$, until the end state ($\bullet$) is reached.
(c) The *micro-sequence* $\mathbf{s}''$ is obtained by concatenating every state observed at the micro-level. The *macro-sequence* $\mathbf{s}'$ is the sequence of states that the

macro-model was in at the time when each micro-state was observed. In the previous example, we had $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ and $\mathbf{s}' = \mathsf{AAABBBCCC}$.

Our goal is to find the macro-sequence $\mathbf{s}'$ and the micro-model $\mathcal{M}_i''$ for every state $i \in \mathbf{S}'$. For this purpose, only the micro-sequence $\mathbf{s}''$ and the macro-model $\mathcal{M}'$ are known. Note that, without knowing $\mathbf{s}'$ nor $\{\mathcal{M}_i''\}$ to start with, there is no idea about how an observed micro-sequence such as $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ can be partitioned into a set of activities. An algorithm to find an estimate for both $\mathbf{s}'$ and $\{\mathcal{M}_i''\}$ is developed in the next section.

4 Algorithms

The problem above is equivalent to that of finding the unknown parameters $\{\mathcal{M}_i''\}$ for a model that produces both observed data ($\mathbf{s}''$) and unobserved data ($\mathbf{s}'$). Such type of problem fits well into the framework of Expectation-Maximization [8,9]. If the missing data $\mathbf{s}'$ were known, then it would be possible to calculate $\{\mathcal{M}_i''\}$ directly from $\mathbf{s}'$ and $\mathbf{s}''$. On the other hand, if the model parameters $\{\mathcal{M}_i''\}$ were known, then it would be possible to determine the missing data $\mathbf{s}'$. What makes the problem especially difficult is the fact that both $\{\mathcal{M}_i''\}$ and $\mathbf{s}'$ are unavailable.

For this kind of problem, it is possible to devise an Expectation-Maximization procedure along the following lines:

(a) Obtain, by some means, an initial estimate for the missing data.
(b) With the current estimate for the missing data, obtain an improved estimated for the unknown model parameters.
(c) With the current estimate for the model parameters, obtain an improved estimate for the missing data.
(d) Repeat the sequence of steps (b) and (c) above until the missing data and the model parameters converge.

Algorithm 1 describes an adaptation of the above procedure to solve our problem. We start by randomizing the macro-sequence $\mathbf{s}'$ (step 1) and then use this sequence to obtain an estimate for the micro-models $\{\mathcal{M}_i''\}$ (step 2). After that, we use the current estimate of $\{\mathcal{M}_i''\}$ to obtain a better estimate for $\mathbf{s}'$ (step 3), and then use this $\mathbf{s}'$ to obtain a better estimate for $\{\mathcal{M}_i''\}$ (step 2), and so on, until both estimates converge. The problem now is how to perform steps 2 and 3 in Algorithm 1. A solution to these sub-problems is described in Sections 4.1 and 4.2, respectively.

4.1 Finding $\{\mathcal{M}_i''\}$ When $\mathbf{s}'$ Is Known

In this section we suppose that the macro-sequence $\mathbf{s}'$ is known, for example $\mathbf{s}' = \mathsf{AAABBBCCC}$. Then, what is left to find out is $\mathcal{M}_i''$ for all states $i \in \mathbf{S}'$. This is described in Algorithm 2. Basically, one considers the transitions that occur in the micro-sequence $\mathbf{s}''$ within each state in macro-sequence $\mathbf{s}'$. For $\mathbf{s}'' = \mathsf{XYZYZZZXY}$

Algorithm 1. Estimate the micro-models $\{\mathcal{M}_i''\}$ and the macro-sequence $\mathbf{s}'$ from the macro-model $\mathcal{M}'$ and the micro-sequence $\mathbf{s}''$

1. Draw a random sequence $\tilde{\mathbf{s}}$ from the Markov chain $\mathcal{M}'$ and use this sequence as the basis to build a macro-sequence $\mathbf{s}'$ with the same length as $\mathbf{s}''$ (for example, if $\tilde{\mathbf{s}} = \mathsf{ABC}$ and $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ then $\mathbf{s}' = \mathsf{AAABBBCCC}$)
2. From the micro-sequence $\mathbf{s}''$, the macro-model $\mathcal{M}'$ and the current estimate for $\mathbf{s}'$ find an estimate for $\{\mathcal{M}_i''\}$ (see Algorithm 2 in Section 4.1)
3. From the micro-sequence $\mathbf{s}''$, the macro-model $\mathcal{M}'$ and the current estimate for $\{\mathcal{M}_i''\}$ find an estimate for $\mathbf{s}'$ (see Algorithm 3 in Section 4.2)
4. Go back to step 2 and repeat until the estimates for $\mathbf{s}'$ and $\{\mathcal{M}_i''\}$ converge.

and $\mathbf{s}'$= AAABBBCCC, the substring for state A is ∘XYZ•, the substring for state B is ∘YZZ•, and the substring for state C is ∘ZXY•. (Note that if state A would appear again in $\mathbf{s}'$ then a second substring would be associated with A, and similarly for other states.) From the set of substrings associated with each state i (step 1), one counts the number of transitions (step 2b) and, after normalization (step 2c), the result yields $\mathcal{M}_i''$.

4.2 Finding s′ When $\{\mathcal{M}_i''\}$ Are Known

In this section, we suppose that the micro-model $\mathcal{M}_i''$ for each state $i \in \mathbf{S}'$ is available, but the macro-sequence $\mathbf{s}'$ is unknown, so we want to determine $\mathbf{s}'$ from $\mathbf{s}''$, $\{\mathcal{M}_i''\}$ and $\mathcal{M}'$. Note that the macro-sequence $\mathbf{s}'$ is produced by the macro-model $\mathcal{M}'$, which is a Markov chain, so there may be several possibilities for $\mathbf{s}'$. In general, we will be interested in finding the most likely solution for $\mathbf{s}'$.

The most likely $\mathbf{s}'$ is given by the sequence of macro-states that is able to produce $\mathbf{s}''$ with highest probability. In the example above, we had $\mathbf{s}''$ = XYZYZZZXY. We know that $\mathbf{s}''$ begins with X and therefore the macro-sequence $\mathbf{s}'$ must be initiated by a macro-state whose micro-model can begin with X. As it happens, there is a single such macro-state in Figure 3, and it is A. So now that we have begun with A, we try to parse the following symbols in $\mathbf{s}''$ with the micro-model $\mathcal{M}_{\mathsf{A}}''$. We find that this micro-model can account for the substring XYZ, after which it ends, so a new macro-state must be chosen to account for the second Y in $\mathbf{s}''$.

In Figure 3, the only micro-model that begins with Y is $\mathcal{M}_{\mathsf{B}}''$. Therefore, the second macro-state is B. We now use $\mathcal{M}_{\mathsf{B}}''$ to parse the following symbols of $\mathbf{s}''$, taking us all the way through YZZZ, when $\mathcal{M}_{\mathsf{B}}''$ cannot parse the following X. A third macro-state is needed to parse the final XY but no suitable solution can be found, because the micro-model $\mathcal{M}_{\mathsf{A}}''$ begins with X but does not end in Y. The problem is that the parsing of micro-model $\mathcal{M}_{\mathsf{B}}''$ went too far. It should have stopped on YZZ and let the final ZXY be parsed by micro-model $\mathcal{M}_{\mathsf{C}}''$. In this case we would have $\mathbf{s}'$ = AAABBBCCC.

This simple example is enough to realize that there may be the need to backtrack and there may be several possible solutions for $\mathbf{s}'$. With both $\mathbf{s}'$ and $\mathbf{s}''$,

Algorithm 2. Estimate the micro-models $\{\mathcal{M}''_i\}$ from the micro-sequence $\mathbf{s}''$ and the macro-sequence $\mathbf{s}'$

1. Separate the micro-sequence $\mathbf{s}''$ into a set of substrings corresponding to the different macro-states in $\mathbf{s}'$. Let $\mathbf{s}''[n_1 : n_2]$ denote a substring of $\mathbf{s}''$ from position n_1 to position n_2. Then, for $\mathbf{s}'$ in the form,

$$\mathbf{s}' = \underbrace{aa...a}_{n_a}\underbrace{bb...b}_{n_b}...\underbrace{cc...c}_{n_c}$$

   pick the first n_a elements in the micro-sequence $\mathbf{s}''$ and create a substring ($\mathbf{s}''[1 : n_a]$) associated with state a, pick the following n_b elements of $\mathbf{s}''$ and create a substring ($\mathbf{s}''[n_a{+}1 : n_a{+}n_b]$) associated with state b, and so on. Each substring should include the start ($\circ$) and end ($\bullet$) states. In the next step, $subs(i)$ is used to denote the set of substrings associated with state i.
2. For each distinct state i found in $\mathbf{s}'$, do the following:
   (a) Initialize the corresponding micro-model $\mathcal{M}''_i = (\overline{\mathbf{S}''_i}, \mathbf{T}''_i)$ where $\overline{\mathbf{S}''_i}$ is the set of distinct states found in the substrings of $subs(i)$ and $\mathbf{T}''_i$ is a transition matrix initialized with zeros.
   (b) For every consecutive pair of micro-states $\mathbf{s}''[k : k{+}1]$ in each substring in $subs(i)$, count the transition from micro-state $\mathbf{s}''[k]$ to micro-state $\mathbf{s}''[k{+}1]$ by incrementing the corresponding position in matrix $\mathbf{T}''_i$. Such counting includes the start ($\circ$) and end ($\bullet$) states as well.
   (c) Normalize each row of the transition matrix $\mathbf{T}''_i$ such that the sum of values in each row is equal to 1.0, except for the last row which represents the end state and therefore its sum should be zero as in Figure 2.

together with $\mathcal{M}'$ and $\{\mathcal{M}''_i\}$, it is possible to calculate the probability of observing a particular micro-sequence $\mathbf{s}''$. This is the product of all transition probabilities in the macro- and micro-models. Let $\mathbf{T}(i, j)$ denote the transition probability from state i to state j in a transition matrix $\mathbf{T}$. Then, in the example above, we have:

$\mathbf{s}'[1] = \mathsf{A}$	$\mathbf{s}''[1] = \mathsf{X}$	$\mathbf{T}'(\circ, \mathsf{A}) \times \mathbf{T}''_{\mathsf{A}}(\circ, \mathsf{X})$	$= 1.0 \times 1.0$
$\mathbf{s}'[2] = \mathsf{A}$	$\mathbf{s}''[2] = \mathsf{Y}$	$\mathbf{T}''_{\mathsf{A}}(\mathsf{X}, \mathsf{Y})$	$= 1.0$
$\mathbf{s}'[3] = \mathsf{A}$	$\mathbf{s}''[3] = \mathsf{Z}$	$\mathbf{T}''_{\mathsf{A}}(\mathsf{Y}, \mathsf{Z})$	$= 1.0$
$\mathbf{s}'[4] = \mathsf{B}$	$\mathbf{s}''[4] = \mathsf{Y}$	$\mathbf{T}''_{\mathsf{A}}(\mathsf{Z}, \bullet) \times \mathbf{T}'(\mathsf{A}, \mathsf{B}) \times \mathbf{T}''_{\mathsf{B}}(\circ, \mathsf{Y})$	$= 1.0 \times 1.0 \times 1.0$
$\mathbf{s}'[5] = \mathsf{B}$	$\mathbf{s}''[5] = \mathsf{Z}$	$\mathbf{T}''_{\mathsf{B}}(\mathsf{Y}, \mathsf{Z})$	$= 1.0$
$\mathbf{s}'[6] = \mathsf{B}$	$\mathbf{s}''[6] = \mathsf{Z}$	$\mathbf{T}''_{\mathsf{B}}(\mathsf{Z}, \mathsf{Z})$	$= 0.5$
$\mathbf{s}'[7] = \mathsf{C}$	$\mathbf{s}''[7] = \mathsf{Z}$	$\mathbf{T}''_{\mathsf{B}}(\mathsf{Z}, \bullet) \times \mathbf{T}'(\mathsf{B}, \mathsf{C}) \times \mathbf{T}''_{\mathsf{C}}(\circ, \mathsf{Z})$	$= 0.5 \times 1.0 \times 1.0$
$\mathbf{s}'[8] = \mathsf{C}$	$\mathbf{s}''[8] = \mathsf{X}$	$\mathbf{T}''_{\mathsf{C}}(\mathsf{Z}, \mathsf{X})$	$= 1.0$
$\mathbf{s}'[9] = \mathsf{C}$	$\mathbf{s}''[9] = \mathsf{Y}$	$\mathbf{T}''_{\mathsf{C}}(\mathsf{X}, \mathsf{Y}) \times \mathbf{T}''_{\mathsf{C}}(\mathsf{Y}, \bullet) \times \mathbf{T}'(\mathsf{C}, \bullet)$	$= 1.0 \times 1.0 \times 1.0$

The product of all these probabilities is $p = 0.25$. For computational reasons, we use the log probability $log(p)$ instead. In general, we choose the solution for $\mathbf{s}'$ which yields the highest value for the log probability. The procedure is described in Algorithm 3. In particular, step 2 in Algorithm 3 is a recursive function that explores all possibilities for $\mathbf{s}'$ with non-zero probability. Such recursive

Algorithm 3. Determine the most likely macro-sequence $\mathbf{s}'$ for a given micro-sequence $\mathbf{s}''$ when both $\mathcal{M}'$ and $\{\mathcal{M}''_i\}$ are known

1. Let $\mathbf{s}''[k]$ be the micro-state at position k in the micro-sequence $\mathbf{s}''$ and let $\mathbf{s}'[k]$ be the corresponding macro-state which is to be determined. Both sequences start at $k = 1$ and end at $k = n$. Run step 2 recursively, starting from $k = 1$.
2. Consider the following possibilities for $\mathbf{s}'[k]$:
   (a) If $k = 1$ then $\mathbf{s}'[k]$ can be any macro-state i such that $\mathbf{T}'(\circ, i) > 0$ and $\mathbf{T}''_i(\circ, \mathbf{s}''[k]) > 0$. For every such macro-state i, set $\mathbf{s}'[k] \coloneqq i$ and run step 2 for $k \coloneqq k{+}1$.
   (b) If $1 < k \leq n$ then consider the following cases:
      i. if both $\mathbf{s}''[k{-}1]$ and $\mathbf{s}''[k]$ come from the same micro-model $\mathcal{M}''_i$ then $\mathbf{s}'[k{-}1] = \mathbf{s}'[k] = i$. Consider this case only if $\mathbf{T}''_i(\mathbf{s}''[k{-}1], \mathbf{s}''[k]) > 0$. If so, set $\mathbf{s}'[k] \coloneqq i$ and run step 2 for $k \coloneqq k{+}1$.
      ii. if $\mathbf{s}''[k{-}1]$ comes from $\mathcal{M}''_i$ and $\mathbf{s}''[k]$ comes from $\mathcal{M}''_j$ (with $i \neq j$) then $\mathbf{s}'[k{-}1] = i$ and $\mathbf{s}'[k] = j$. Consider every possible macro-state j for which $\mathbf{T}''_i(\mathbf{s}''[k{-}1], \bullet) \times \mathbf{T}'(i, j) \times \mathbf{T}''_j(\circ, \mathbf{s}''[k]) > 0$. For every such macro-state j, set $\mathbf{s}'[k] \coloneqq j$ and run step 2 for $k \coloneqq k{+}1$.
   (c) If $k = n$ then we have reached the end of $\mathbf{s}''$ and now have a complete candidate for $\mathbf{s}'$. Accept this candidate only if it terminates correctly, i.e. only if $\mathbf{T}''_i(\mathbf{s}''[n], \bullet) \times \mathbf{T}'(i, \bullet) > 0$ where $i = \mathbf{s}'[n]$.
3. From all candidates for $\mathbf{s}'$ collected in step 2, return the candidate which provides the highest log probability for $\mathbf{s}''$.

exploration has the form of a tree, since the possibilities for $\mathbf{s}'[k{+}1]$ are built upon the possibilities for $\mathbf{s}'[k]$. The complete path from root ($k = 1$) to every leaf ($k = n$) represents a different candidate for $\mathbf{s}'$. In step 3, the algorithm returns the candidate with highest log probability, where this log probability is the sum of the log probabilities along the path in the tree.

5 Case Study

Agent-based simulation [10,11] is an effective means to study the behaviour of systems involving the actions and interactions of autonomous agents. Although there are several platforms for agent-based simulation [12], here we turn our attention to the Agent-Object-Relationship approach (AOR) introduced by [13] and which can be used to model and simulate business processes [14].

The AOR system is a simulation platform where agents respond to events in their environment by executing actions and interacting with each other, which in turn generates new events. There are basically two different kinds of events. An *exogenous* event is an external event (such as the arrival of a new customer) which does not depend on the actions of agents. Usually, the occurrence of an exogenous event is what triggers a simulation run. To run multiple instances of a business process, the AOR system schedules the occurrence of exogenous events to trigger the whole process at different points in time.

The second kind of event is a *message* and it is the basis of simulation in the AOR system. Agents send messages to one another, which in turn generates new messages. For example, if agent X sends a message M1 to agent Y, then this may result in a new message M2 being sent from Y to Z. Such chaining of messages keeps the simulation running until there are no more messages to be exchanged. At that point, a new exogenous event is required to trigger a new simulation run. In this work, we represent the exchange of a message M being sent from agent X to agent Y as: $X \xrightarrow{M} Y$.

Our case study is based on the implementation of a purchasing scenario in the AOR system. On a high (macro) level, the process can be represented as in Figure 4 and can be described as follows. In a company, an employee needs a certain commodity (e.g. a printer cartridge). If the product is available at the warehouse, then the warehouse dispatches the product to the employee. Otherwise, the product must be purchased from an external supplier. All purchases must be previously approved by the purchasing department. If the purchase is not approved, the process ends immediately. If the purchase is approved, the process proceeds with the purchasing department ordering and paying for the product from the supplier. The supplier delivers the product to the warehouse, and the warehouse dispatches the product to the employee.

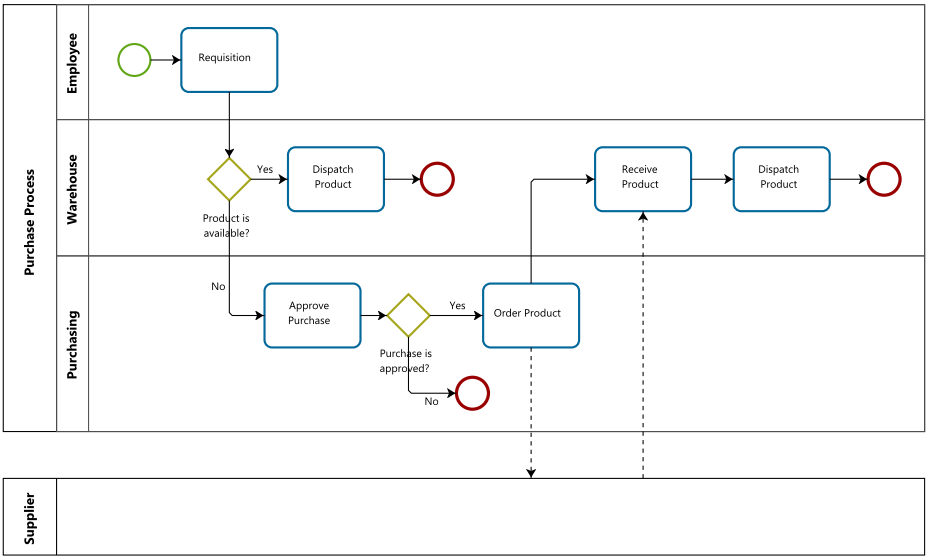
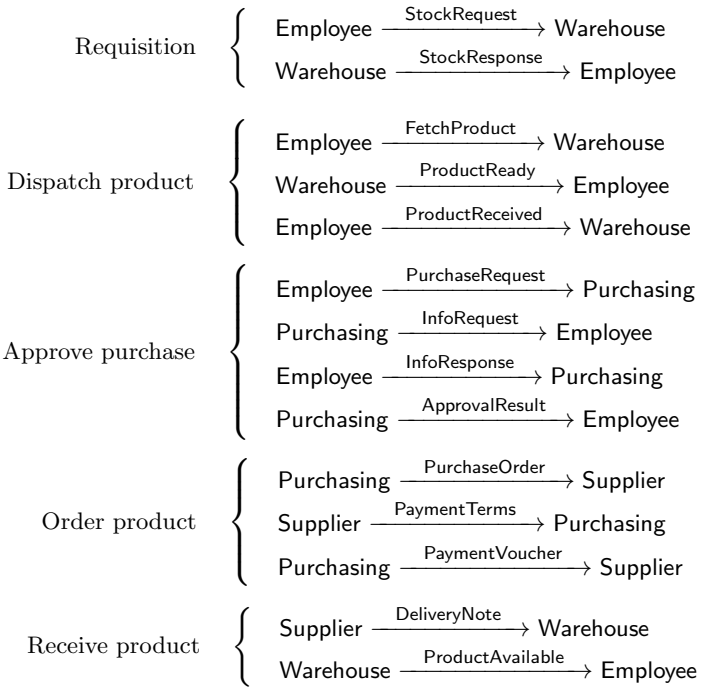


Fig. 4. Macro-level description of a purchase process

This process was implemented in the AOR system using the AOR Simulation Language (AORSL) [15] to specify the message exchanges between agents. There are four types of agent: Employee, Warehouse, Purchasing, and Supplier. There is one instance of the Warehouse agent and one instance of the Purchasing agent. However,

there are multiple instances of the Employee agent (each instance exists during a simulation run; it is created at the start of the run and destroyed when the run finishes). We could have done the same for the Supplier agent, but for simplicity we considered only one instance of Supplier.

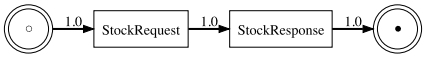
The process includes the following message exchanges:



It should be noted that the AOR system has no knowledge about the macro-level activities on the left-hand side. Instead, the agents have rules to implement the message exchanges on the right-hand side. In addition, we suppose that:

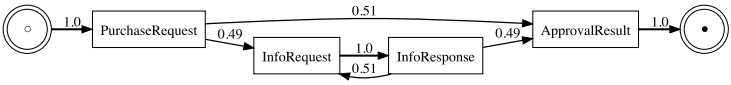
- For the purchase request to be approved, the purchasing department may enquire the employee an arbitrary number of times to obtain further info about the purchase request. This means that the exchanges InfoRequest and InfoResponse may occur multiple times (or even not occur at all).
- The purchasing department may not be satisfied with the payment terms requested by a particular supplier, and may choose to negotiate those terms or get in contact with another supplier. This means that PurchaseOrder and PaymentTerms may occur multiple times (but they must occur at least once).

Simulating this process in AOR produces an event log with an AOR-specific XML structure. From this event log, it is possible to recognize each new instance of the Employee agent as a different instance of the process. Therefore, we collected the sequence of events for each Employee agent; these represent our traces, i.e. the micro-sequences. The process in Figure 4 represents the macro-model, and it was converted to a Markov chain representation. Feeding the micro-sequences

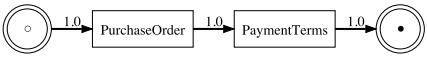


(a) Requisition

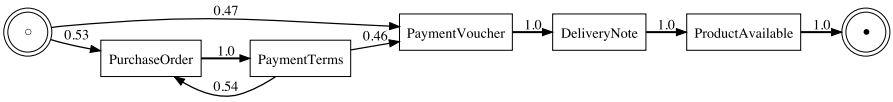
(b) Dispatch product



(c) Approve purchase



(d) Order product



(e) Receive product

Fig. 5. Results obtained for an AOR simulation with 10.000 simulation steps

and the macro-model to Algorithm 1, we obtained the micro-models shown in Figure 5, in graphical form.

The results in Figure 5 represent the correct behaviour, except for the fact that PurchaseOrder, PaymentTerms and, PaymentVoucher also appear in the "Receive product" activity rather than only in the "Order product" activity. Since these two activities always occur together and sequentially in the model of Figure 4, there is no way to determine that those events belong to the first activity and not to the second. In the random initialization step in Algorithm 1, some of these events are attributed to "Order product" and others are attributed to "Receive product". Also, when the loop between PurchaseOrder and PaymentTerms repeats, the micro-sequence is longer and therefore it becomes more likely that some of these events are captured by "Receive product".

6 Conclusion

In this work we have described a hierarchical Markov model and an Expectation-Maximization procedure to discover the relationship between the micro-level events recorded in an event log and the macro-level activities in a business process model. We have shown that the proposed approach is able to perform a correct discovery, at least in an experimental setting where we used an agent simulation platform. Except for a couple of events, the algorithm was able to

associate the micro-events with the correct macro-activity, while also providing a micro-model for the behaviour of agents in each of those activities. In future work, we will be looking into the problems that arise when applying the approach to real-world event logs with noise. The approach can be applied in scenarios where a macro-level description of the business process is available.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Greco, G., Guzzo, A., Pontieri, L.: Mining Hierarchies of Models: From Abstract Views to Concrete Specifications. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 32–47. Springer, Heidelberg (2005)
3. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
4. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity Mining by Global Trace Segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010)
5. Bose, R.P.J.C., Verbeek, E.H.M.W., van der Aalst, W.M.P.: Discovering Hierarchical Process Models Using ProM. In: Nurcan, S. (ed.) CAiSE Forum 2011. LNBIP, vol. 107, pp. 33–48. Springer, Heidelberg (2012)
6. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. Knowledge Engineering Review 10(2), 115–152 (1995)
7. Veiga, G.M., Ferreira, D.R.: Understanding Spaghetti Models with Sequence Clustering for ProM. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009 Workshops. LNBIP, vol. 43, pp. 92–103. Springer, Heidelberg (2010)
8. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society 39(1), 1–38 (1977)
9. McLachlan, G.J., Krishnan, T.: The EM Algorithm and Extensions. Wiley Series in Probability and Statistics. Wiley-Interscience (2008)
10. Bonabeau, E.: Agent-based modeling: Methods and techniques for simulating human systems. PNAS 99(suppl. 3), 7280–7287 (2002)
11. Davidsson, P., Holmgren, J., Kyhlbäck, H., Mengistu, D., Persson, M.: Applications of Agent Based Simulation. In: Antunes, L., Takadama, K. (eds.) MABS 2006. LNCS (LNAI), vol. 4442, pp. 15–27. Springer, Heidelberg (2007)
12. Railsback, S.F., Lytinen, S.L., Jackson, S.K.: Agent-based simulation platforms: Review and development recommendations. Simulation 82(9), 609–623 (2006)
13. Wagner, G.: AOR Modelling and Simulation: Towards a General Architecture for Agent-Based Discrete Event Simulation. In: Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2003. LNCS (LNAI), vol. 3030, pp. 174–188. Springer, Heidelberg (2004)
14. Wagner, G., Nicolae, O., Werner, J.: Extending discrete event simulation by adding an activity concept for business process modeling and simulation. In: Proceedings of the 2009 Winter Simulation Conference, pp. 2951–2962 (2009)
15. Nicolae, O., Wagner, G., Werner, J.: Towards an Executable Semantics for Activities Using Discrete Event Simulation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009 Workshops. LNBIP, vol. 43, pp. 369–380. Springer, Heidelberg (2010)