# Mining the low-level behavior of agents in high-level business processes

Diogo R. Ferreira*

IST – Technical University of Lisbon,
Campus do Taguspark,
Avenida Prof. Dr. Cavaco Silva,
2744-016 Porto Salvo, Portugal
E-mail: diogo.ferreira@ist.utl.pt
*Corresponding author

Fernando Szimanski and Célia Ghedini Ralha

Universidade de Brasília (UnB),
Departamento de Ciência da Computação,
Campus Universitário Darcy Ribeiro,
ICC Ala Norte, Caixa postal 4466,
70904-970 Brasília, DF - Brasil
E-mail: fszimanski@gmail.com
E-mail: ghedini@cic.unb.br

**Abstract:** Currently there is a gap between the high level of abstraction at which business processes are modeled and the low level nature of the events that are recorded during process execution. When applying process mining techniques, it is possible to discover the logic behind low-level events but it is difficult to determine the relationship between those low-level events and the high-level activities in a given process model. In this work, we introduce a hierarchical Markov model to capture both the high-level behavior of activities and the low-level behavior of events. We also develop an Expectation-Maximization technique to discover that kind of hierarchical model from a given event log and a high-level description of the business process. We use this technique to understand the behavior of agents in business processes, from the control-flow perspective and from the organizational perspective as well. Using an agent-based simulation platform (AOR), we implemented a purchasing process and generated an event log in order to illustrate the benefits of the proposed approach and to compare the results with existing process mining techniques, namely the ones that are available in the ProM framework.

**Keywords:** Process Mining, Agent-Based Simulation, Hierarchical Markov Model, Expectation-Maximization, Agent-Object-Relationship (AOR)

**Reference** to this paper should be made as follows: Ferreira, D.R., Szimanski, F., Ralha, C.G. (2013) 'Mining the low-level behavior of agents in high-level business processes', *Int. J. Business Process Integration and Management*, Vol. 6, No. 2, pp.146–166.

**Biographical notes:** Diogo R. Ferreira is professor of information systems at the Technical University of Lisbon, and he is an active researcher in the field of business process management, particularly in the area of process mining.

Fernando Szimanski is an adjunct professor at University Center UNIRG, and a PhD student at the University of Brasília. His research focuses on using agent-based simulation and process mining for business process improvement.

Célia Ghedini Ralha is an associate professor at the Computer Science Department, University of Brasília, and she is an active researcher in the field of intelligent information systems, particularly using agent-based models.

This work is a revised and expanded version of a paper entitled 'A Hierarchical Markov Model to Understand the Behaviour of Agents in Business Processes' presented at the 8th International Workshop on Business Process Intelligence (BPI 2012), Tallinn, Estonia, 3 September 2012.

## 1 Introduction

When a business process is performed over a systems infrastructure, it is possible to record all events that occur during process execution. These events can then be extracted as an event log for further analysis. *Process mining* (van der Aalst, 2011) is an emerging field which concerns the development of techniques to discover process models that explain the behavior found in an event log. In the context of process mining, an event log is usually a list of events in chronological order, where each event refers to an activity that has been performed by some agent during the execution of some process instance.

As illustrated in Figure 1(a), each event in the event log contains the following elements: a *case id* which identifies the process instance; the *activity* that has been executed in that process instance; the *agent* who performed the activity; and the *timestamp* of when the event occurred. By applying process mining techniques it is possible to extract different kinds of behavioral models from such an event log. An example is the control-flow model depicted in Figure 1(b), which has been obtained by analyzing the sequence of activities recorded in the *activity* column.

Each state in the control-flow model of Figure 1(b) represents a different activity and the arrows between states represent the transitions that have been observed in the event log (where each transition is between two consecutive events with the same case id). Some transitions have occurred only once, while others occurred multiple times, in different process instances. Let the state where an arrow begins be called the *source state* and the state where the arrow ends be called the *target state*. The label next to each arrow indicates the number of times the transition has occurred divided by the total number of times that the source state was recorded in the event log. In a frequentist interpretation, this can be taken as a measure of probability of going from the source state to the target state, and so this model can be regarded as a form of Markov chain.

The special states marked with "○" (for "open") and "●" (for "closed") are used to designate the beginning and end of a process instance. These states are not explicitly recorded in the event log, but they can be figured out from the point where a case id appears for the first time and for the last time, respectively. In practice, the events from different process instances may be interleaved in time and therefore appear interleaved in the event log, but it is possible regroup them according to case id.

Figure 1(c) shows the result of a similar analysis carried out for the *agent* column. Here, each state represents a different agent and the transitions represent the handover of work between agents, i.e. a transition represents the fact that an agent performs an activity before handing the process instance over to another agent. In this example, there are cases where the same agent performs two consecutive activities, and this is the reason why there are self-loops in Figure 1(c). In any case, the method that was used to build this model was the same as in Figure 1(b), and again the transition labels can be taken as a measure of probability of going from a source state into a target state.

### 1.1 The problem of high-level activities

In Figure 1(a) the events that are recorded in the event log refer to high-level activities such as *Requisition*, *Dispatch Product*, *Approve Purchase*, etc. These are activity labels that could be used in a high-level description of the business process. However, in practice it often happens that the events that can be captured by the supporting systems are of a low-level nature, such as e.g. "agent $X$ sent a message $M$ to agent $Y$", without an explicit connection to the high-level activity that is being performed. In addition, each activity may be the cause of several events, so there is clearly a gap between the high level of abstraction at which business processes are defined and the low-level nature of events that are recorded during process execution.

In such scenario, analyzing the event log alone will provide little insight into how a process, defined in terms of high-level activities, is being performed at run-time, because the mapping between low-level events and high-level activities is missing. In this work, we present an approach that is able to discover such mapping, under certain assumptions. We assume that a high-level description of the process is available and that such description can be translated into a high-level behavioral model (i.e. a control-flow model). On the other hand, we have seen in Figure 1 that it is possible to extract a low-level behavioral model from an event log (either a control-flow or an interaction model, as in Figures 1(a) and 1(b), respectively). Our approach is based on the idea that when the low-level behavioral model is being discovered from the event log, the high-level model should also be provided as input, so that not only the low-level model but also the relationship between the low-level model and the high-level model can be discovered at the same time.

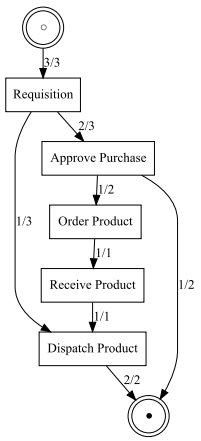
A more formal definition of the problem can be found in Section 3, and a solution to the problem is developed in Section 4. In Section 5 we show that the proposed solution is able to deal with workflow patterns that are commonly found in practice, and in Section 6 we present a case study application involving a purchasing process implemented using agent-based simulation.

### 1.2 The role of agent-based simulation

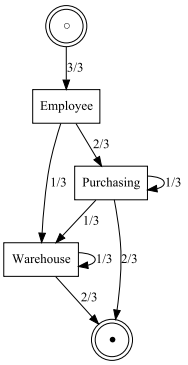
To study the problem above and to develop the proposed solution, we found it very useful to have a way to simulate the generation of low-level events from high-level activities. Traditional process simulation tools are not entirely appropriate for this purpose, since one needs to simulate not only the high-level process, but also the way in which a set of participants execute the process, thereby generating a sequence of non-deterministic, low-level events within the scope of each activity. Agent-based platforms are especially convenient for this purpose, since agents can have interesting characteristics, namely autonomy and pro-activity (Wooldridge and Jennings, 1995), that can be used to mimic the way humans interact while performing their tasks.

| *case id* | *activity* | *agent* | *timestamp* |
|---|---|---|---|
| 1 | Requisition | Employee | 2012-06-14 10:23 |
| 1 | Dispatch Product | Warehouse | 2012-06-14 15:45 |
| 2 | Requisition | Employee | 2012-06-15 11:21 |
| 2 | Approve Purchase | Purchasing | 2012-06-19 14:34 |
| 3 | Requisition | Employee | 2012-06-21 09:31 |
| 3 | Approve Purchase | Purchasing | 2012-06-23 10:22 |
| 3 | Order Product | Purchasing | 2012-06-23 11:53 |
| 3 | Receive Product | Warehouse | 2012-06-27 12:07 |
| 3 | Dispatch Product | Warehouse | 2012-06-28 08:25 |
| ... | ... | ... | ... |

(a) Event log




(b) Control-flow model




(c) Interaction model

**Figure 1** An example of an event log together with the extracted control-flow model

In this work we make use of a particular agent-based platform, namely the Agent-Object-Relationship (AOR) framework (Wagner, 2004). This platform allowed us to fully implement a purchasing process by specifying a set of agents and their behavior within the scope of each high-level activity. The AOR framework itself has the ability to record all events that are generated during simulation in an XML file, and with some processing it is possible to transform this file into an event log that is similar to the kind of event logs used for process mining. Provided with the event log and with a model for the high-level process, our approach successfully discovers the low-level behavior associated with each high-level activity.

Although in Section 6 we turn our attention to a practical scenario involving a purchase process, the same approach can be used to simulate and discover business processes in a wider range of applications. Section 5 presents the results of a set of experiments with workflow patterns; these experiments provide an indication that the proposed approach can handle general patterns of behavior. Through agent-based simulation, it is possible to generate non-deterministic behavior. On the other hand, the use of Markov models provides resilience to noise. Therefore, the approach described in this paper – particularly, the algorithms described in Section 4 – can be used in realistic scenarios to discover the run-time behavior of business processes for which a high-level model is known.

### *1.3 Previous work*

This work is an extension of (Ferreira et al., 2012). In particular, Section 5, which reports on a set of experiments with basic workflow patterns, is entirely new, and Section 6 has been expanded with experiments using the ProM framework (Section 6.6) in order to compare the results obtained with our approach to those obtained using existing process mining techniques. Basically, the difference is in the fact that it becomes possible to capture a separate model of low-level behavior for each higher-level activity, rather than capturing all the low-level behavior at once in a single model which, in general, is hard to understand and interpret in terms of a set of high-level activities.

## 2 Related work

The gap between high-level activities and low-level events is a well-known problem in the field of process mining (Greco et al., 2005; Günther and van der Aalst, 2007; Günther et al., 2010; Bose et al., 2012). Despite the development of a wide range of process mining techniques, most of these are able to discover behavioral models that are at the same level of abstraction as the events recorded in the event log. However, with an increasing general interest in process mining, end users are looking for solutions to analyze event data and visualize the results at a higher level of abstraction, preferably at the same level as they are accustomed to when modeling their business processes.

Recently, the research community has been looking into this problem and, while it is still a topic of ongoing research, a few approaches have already been proposed to be able to produce more abstract models from an event log. These approaches can be divided into two main groups:

(a) First, there are techniques that work on the basis of models, by extracting a low-level model from the event log and then creating more abstract representations of that model. Examples are (Greco et al., 2005) and (Günther and van der Aalst, 2007). Basically, these techniques work by aggregating nodes in the model, in order to produce a simplified and more abstract picture of the process. In general, these approaches allow a stepwise simplification of the process until, in the limit, everything is aggregated into a single node. It is the end user who must know how far to carry the simplification in order to obtain meaningful results. A disadvantage of these approaches is that it is not possible to automatically identify aggregated nodes as meaningful business activities (they are simply labeled as “Cluster A”, “Cluster B”, etc.), so it may be difficult for the end user to understand and analyze the results.

(b) Second, there are techniques that work on the basis of the events, by translating the event log into a more abstract sequence of events and then producing a model from that translated event log. Examples are (Günther et al., 2010) and (Bose et al., 2012). Basically, these techniques work by identifying frequent patterns of events in the event log, and then substituting each of these patterns by a single, higher-level event. After all substitutions have been made, the event log becomes a sequence of more abstract events. As a final step, it is possible to extract a model by usual techniques, such as the $\alpha$-*algorithm* (van der Aalst et al., 2004), the *heuristics miner* (Weijters et al., 2006), or the *genetic miner* (de Medeiros and Weijters, 2005). As with other approaches, it is possible to perform this abstraction in multiple stages, but there is no guarantee that the patterns of events that are identified in the event log correspond to meaningful business activities, so it is up to the end user to determine whether such correspondence actually exists.

The problem with these approaches is that, although they represent powerful mechanisms of abstraction, they do not take into account that often there is already an abstract notion of the process. This abstract notion, if translated into a high-level process model, can provide valuable input as to what are the main blocks (i.e. high-level activities) to be expected when building abstractions over the behavior observed in the event log.

In this work, we introduce a hierarchical Markov model to capture both the high-level behavior of the business process and the low-level behavior that can be extracted from the event log. The advantage of using such hierarchical model is that it captures also the relationship between high-level activities and low-level events, so that when a pattern of low-level behavior is discovered, it can be identified with a certain high-level activity.

To the best of our knowledge, hierarchical Markov models have been used in image processing (Collet and Murtagh, 2004; Zhao et al., 2006; Provost et al., 2004; Demonceaux and Kachi-Akkouche, 2006), wireless communications (Karande et al., 2003; Yang and Alouini, 2002; Khayam and Radha, 2003; Tao et al., 2001), and data mining (Youngblood and Cook, 2007; Liao et al., 2007; Wu and Aberer, 2005; Cook et al., 2006), but this is the first time that such kind of model is being applied to the analysis of business processes, and process mining in particular.

## 3 The hierarchical Markov model

A hierarchical Markov model is basically a Markov chain where each state contains another Markov chain. While the upper-level Markov chain is in a certain state (let us call it "macro" state), the lower-level Markov chain for that macro state can be switching between "micro" states. A hierarchical Markov model may have an arbitrary number of layers, but for our purposes if suffices to use a model with just two layers: one to represent the high-level process, and another to represent the low-level behavior associated with each high-level activity.

To illustrate a simple example of the proposed hierarchical Markov model, consider a business process that can be described on a high level as comprising the sequence of activities A, B, and C. This sequence of activities represents the control-flow model for the high-level process, and is depicted in Figure 2. On the other hand, consider that when each of these activities is performed, this results in some sequence of low-level events being recorded in the event log. We will refer to these low-level events as X, Y and Z. These low-level events may represent one of several different things: for example, they may represent a set of low-level tasks being performed by agents; they may represent the agents themselves; or they may represent a set of low-level messages exchanged between agents.

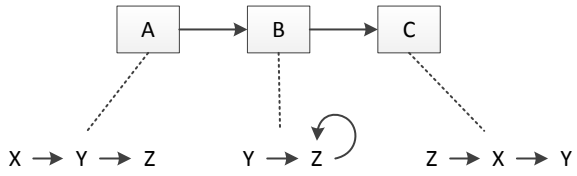



**Figure 2** A simple example of a hierarchical process model

Whatever the meaning of the low-level events X, Y and Z, we consider that activity A results in the sequence of events XYZ. In a similar way, activity B results in a sequence of events in the form YZZ..., where there may be multiple Z's until a certain condition becomes true. Finally, activity C results in a sequence of events in the form ZXY. These sequences of events are represented as low-level Markov chains in Figure 2.

Executing this model 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 in terms of the activities A, B, and C is referred to as the *macro-model*. On the other hand, the sequence of events XYZYZZZXY will be called the *micro-sequence*, and the low-level Markov chain that describes the behavior of each macro-activity in terms of the events X, Y and Z is referred to as a *micro-model*.

The problem addressed in this work is *how to discover the macro-sequence and the micro-models from a given macro-model and micro-sequence*. In other words:

- The inputs are the macro-model and the micro-sequence. The macro-model represents the prior knowledge about the business process in terms of a set of high-level activities; it is a macro-level control-flow model expressed as a Markov chain. On the other hand, the micro-sequence represents a trace of low-level events that can be observed in an event log. If there are multiple process instances, there will be multiple micro-sequences in the event log.

- The outputs are the macro-sequence and the micro-models. The macro-sequence represents the actual path in the macro-model (i.e. the sequence of macro-states) that explains how a given micro-sequence was generated during execution. If there are multiple process instances, there will be multiple micro-sequences, and there will be a separate macro-sequence for each micro-sequence. On the other hand, the micro-models represent the behavior of low-level events for each macro-level activity. Each micro-model is expressed as a Markov chain.

### *3.1 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 probability that the next state will be $j$ given that the current state is $i$. For convenience, this will be referred to as the transition probability from the current state $i$ to a subsequent state $j$. In this work, as in (Veiga and Ferreira, 2010), 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 any 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 Markov chain is subject to the stochastic constraint $\sum_{j \in \overline{\mathbf{S}}} \mathbb{P}(j \mid i) = 1$ 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. For the particular case of the end state, 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 3, in the elements marked as **0**.

In a hierarchical Markov model, there is a Markov chain to describe the macro-model (upper level in Figure 2), and there is a set of Markov chains to describe the micro-model for each activity (lower level in Figure 2).

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-models are defined as a set of Markov chains $\{\mathcal{M}''_i : i \in \mathbf{S}'\}$ where each $\mathcal{M}''_i = \langle \overline{\mathbf{S}''_i}, \mathbf{T}''_i \rangle$ is a Markov chain that describes the behavior of agents when performing activity $i \in \mathbf{S}'$.

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

**Figure 3** General form of a transition matrix

For the example in Figure 2, one possible model is shown in Figure 4. In this figure, the macro-model is denoted by $\mathcal{M}'$ and the micro-models are denoted by $\mathcal{M}''_{\mathsf{A}}$, $\mathcal{M}''_{\mathsf{B}}$ and $\mathcal{M}''_{\mathsf{C}}$, respectively. In particular, 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, since both are ½.

### *3.2 Execution*

In general, 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 going through some sequence of states according to the transition probabilities in $\mathbf{T}'$, until the end state ($\bullet$) is reached.

(b) For each 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 going through some sequence of states according to the transition probabilities in $\mathbf{T}''_i$, until the end state ($\bullet$) is reached. Only then can the macro-model proceed to the next state.

(c) The *micro-sequence* $\mathbf{s}''$ is obtained by concatenating every state observed at the micro-level. An example is $\mathbf{s}'' = \mathsf{XYZYZZZXY}$. Clearly, every such micro-state belongs to some macro-state, in the sense that each micro-state was produced by some micro-model associated with a macro-state. In $\mathbf{s}'' = \mathsf{XYZYZZZXY}$, the first micro-states XYZ belong to A; the middle YZZ belong to B; and the last ZXY belong to C. Let $\mathbf{s}'$ be the *macro-sequence* defined as the sequence of states that the macro-model was in at the time when each micro-state was generated. Then $\mathbf{s}' = \mathsf{AAABBBCCC}$.

Our goal is to find the macro-sequence $\mathbf{s}'$ and the micro-models $\{\mathcal{M}''_i\}$ for each macro-state $i \in \mathbf{S}'$. For this purpose, only the micro-sequence $\mathbf{s}''$ and the macro-model $\mathcal{M}'$ are known. Knowing the macro-model $\mathcal{M}'$ does not solve the problem since, in general, the macro-model may be able to generate several possible macro-sequences, and one does not know which macro-sequence has actually occurred for a given micro-sequence. On the other hand, knowing the micro-sequence $\mathbf{s}''$ does not solve the problem either, since

$$\mathcal{M}' = \langle \overline{\mathbf{S}'}, \mathbf{T}' \rangle \qquad \overline{\mathbf{S}'} = \{\circ, \mathsf{A}, \mathsf{B}, \mathsf{C}, \bullet\} \qquad \mathbf{T}' = \begin{array}{c|ccccc} & \circ & \mathsf{A} & \mathsf{B} & \mathsf{C} & \bullet \\ \hline \circ & 0 & 1 & 0 & 0 & 0 \\ \mathsf{A} & 0 & 0 & 1 & 0 & 0 \\ \mathsf{B} & 0 & 0 & 0 & 1 & 0 \\ \mathsf{C} & 0 & 0 & 0 & 0 & 1 \\ \bullet & 0 & 0 & 0 & 0 & 0 \end{array}$$

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

$$\overline{\mathbf{S}''_{\mathsf{A}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \overline{\mathbf{S}''_{\mathsf{B}}} = \{\circ, \mathsf{Y}, \mathsf{Z}, \bullet\} \qquad \overline{\mathbf{S}''_{\mathsf{C}}} = \{\circ, \mathsf{X}, \mathsf{Y}, \mathsf{Z}, \bullet\}$$

$$\mathbf{T}''_{\mathsf{A}} = \begin{array}{c|ccccc} & \circ & \mathsf{X} & \mathsf{Y} & \mathsf{Z} & \bullet \\ \hline \circ & 0 & 1 & 0 & 0 & 0 \\ \mathsf{X} & 0 & 0 & 1 & 0 & 0 \\ \mathsf{Y} & 0 & 0 & 0 & 1 & 0 \\ \mathsf{Z} & 0 & 0 & 0 & 0 & 1 \\ \bullet & 0 & 0 & 0 & 0 & 0 \end{array} \qquad \mathbf{T}''_{\mathsf{B}} = \begin{array}{c|cccc} & \circ & \mathsf{Y} & \mathsf{Z} & \bullet \\ \hline \circ & 0 & 1 & 0 & 0 \\ \mathsf{Y} & 0 & 0 & 1 & 0 \\ \mathsf{Z} & 0 & 0 & ½ & ½ \\ \bullet & 0 & 0 & 0 & 0 \end{array} \qquad \mathbf{T}''_{\mathsf{C}} = \begin{array}{c|ccccc} & \circ & \mathsf{X} & \mathsf{Y} & \mathsf{Z} & \bullet \\ \hline \circ & 0 & 0 & 0 & 1 & 0 \\ \mathsf{X} & 0 & 0 & 1 & 0 & 0 \\ \mathsf{Y} & 0 & 0 & 0 & 0 & 1 \\ \mathsf{Z} & 0 & 1 & 0 & 0 & 0 \\ \bullet & 0 & 0 & 0 & 0 & 0 \end{array}$$

**Figure 4** An example of a hierarchical Markov model

there is no idea about how the observed micro-sequence should be partitioned into a set of macro-activities. An algorithm to find an estimate for both $\mathbf{s}'$ and $\{\mathcal{M}''_i\}$ from $\mathcal{M}'$ and $\mathbf{s}''$ 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 (Dempster et al., 1977; McLachlan and Krishnan, 2008). 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 (EM) procedure along the following lines:

(a) Obtain, by some means, an initial estimate for the missing data $\mathbf{s}'$.

(b) With the current estimate for the missing data, obtain an improved estimated for the unknown model parameters $\{\mathcal{M}''_i\}$.

(c) With the current estimate for the model parameters, obtain an improved estimate for the missing data $\mathbf{s}'$.

(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 main 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.

**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 $\mathbf{s}$ 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 $\mathbf{s} = \mathsf{ABC}$ and $\mathbf{s}'' = \mathsf{XYZYZZZXY}$ then $\mathbf{s}' = \mathsf{AAABBBCCC}$)
2. Given 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 4.2)
3. Given 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 4.3)
4. Go back to step 2 and repeat from there until the estimates for $\mathbf{s}'$ and $\{\mathcal{M}''_i\}$ 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 4.2 and 4.3, respectively.

### 4.1 Sequence expansion

In the example of step 1 in Algorithm 1, the random sequence $\mathbf{s} = \mathsf{ABC}$ extracted from the macro-model $\mathcal{M}'$ has been expanded to $\mathbf{s}' = \mathsf{AAABBBCCC}$, which includes an equal number of A's, B's and C's. Nothing establishes that $\mathbf{s}$ should be expanded in this way. Other expansions are possible (e.g. $\mathbf{s}' = \mathsf{AAAABBBBC}$ or $\mathbf{s}' = \mathsf{ABCCCCCCC}$), as long as the expansion complies with the given sequence of

macro-states (i.e. $\mathfrak{s} =$ ABC). In this work we expand the sequence $\mathfrak{s}$ by repeatedly choosing a state at random from that sequence and inserting an equal symbol next to it, until the sequence length reaches the same length as the given micro-sequence. For example, for the sequence $\mathfrak{s} =$ ABC, we pick a random number of A's, B's and C's to expand this sequence into a macro-sequence with the same length as $\mathbf{s}'' =$ XYZYZZZXY. The general procedure can be described as follows:

- For a micro-sequence $\mathbf{s}''$ of length $m = |\mathbf{s}''|$ and a sequence of macro-states $\mathfrak{s}$ of length $n = |\mathfrak{s}|$, draw a sequence of random numbers $\mathbf{r} = \langle r_1, ..., r_n \rangle$ such that their overall sum is equal to the length of $\mathbf{s}''$ (i.e. $\sum_{i=1}^{n} r_i = m$).
- Assemble the macro-sequence $\mathbf{s}'$ from $\mathfrak{s}$ and $\mathbf{r} = \langle r_1, ..., r_n \rangle$ by concatenating $r_1$ copies of $\mathfrak{s}[1]$ with $r_2$ copies of $\mathfrak{s}[2]$ with $r_3$ copies of $\mathfrak{s}[3]$, and so on.

### *4.2 Finding $\{\mathcal{M}''_i\}$ when $\mathbf{s}'$ is known*

In this section we suppose that the macro-sequence $\mathbf{s}'$ is known, for example $\mathbf{s}'$= 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 $\mathbf{s}''$= XYZYZZZXY and $\mathbf{s}'$= AAABBBCCC, we have the following mapping between micro-states and macro-states:

XYZYZZZXY
| | | | | | | | |
AAABBBCCC

Algorithm 2 begins by fetching the substrings $subs(i)$ for each macro-state $i$. For example, 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 macro-state, Algorithm 2 counts the number of transitions (step 2b) and, after normalization (step 2c), the result yields $\mathcal{M}''_i$.

### *4.3 Finding $\mathbf{s}'$ 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 4, 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 a new macro-state must be chosen to account for the second Y in $\mathbf{s}''$.

In Figure 4, 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

---

**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 (∘) and end (•) 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 of $subs(i)$, 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 (∘) and end (•) states as well.
   (c) Normalize each row of the transition matrix $\mathbf{T}''_i$ such that the sum of the values in each row is equal to 1, except for the last row which represents the end state and therefore its sum should be zero as in Figure 3.

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

**Figure 5** Example of calculating the total probability of producing both $\mathbf{s}'$ and $\mathbf{s}''$

possible solutions for $\mathbf{s}'$. With both $\mathbf{s}'$ and $\mathbf{s}''$, 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, for the example above, we have the sequence of calculations shown in Figure 5.

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 exploration has the form of a tree, since the possibilities for $\mathbf{s}'[k{+}1]$ are built upon the possibilities for $\mathbf{s}'[k]$. Every path from the root ($k = 1$) to a leaf ($k = n$) in this tree 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.

To improve efficiency, the best candidate found so far and its corresponding log-probability can be kept in global variables. When going down a path in the tree (i.e. when building a new candidate through the recursion in step 2), as soon as the log-probability for that candidate gets below the log-probability for the best candidate found so far, that branch can be pruned and the search can proceed immediately to the next branch.

### *4.4 Working with multiple micro-sequences*

Up to this point we have considered the use of a single micro-sequence $\mathbf{s}''$ (and a macro-model $\mathcal{M}'$) to determine the micro-models $\{\mathcal{M}''_i\}$ and the macro-sequence $\mathbf{s}'$. However, an event log contains events from multiple process instances, and each process instance corresponds to a separate micro-sequence. In addition, each micro-sequence is associated with its own macro-sequence. Let $\Omega''$ denote the multiset of micro-sequences (i.e. a set of micro-sequences that may include repeated elements) and let $\Omega'$ denote the multiset of the corresponding macro-sequences. Then Algorithms 1–3 can be easily extended in order to handle multiple micro-sequences, through the following adaptations:

**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 $i = \mathbf{s}'[n]$. If accepted, store the candidate in a list of candidates.
3. From all candidates for $\mathbf{s}'$ collected in step 2(c), return the candidate which provides the highest log-probability for $\mathbf{s}''$.

(a) In step 1 of Algorithm 1 draw a random sequence $\mathbf{s}$ from $\mathcal{M}'$ for each micro-sequence $\mathbf{s}'' \in \Omega''$. Also, in steps 2 and 3 of Algorithm 1 consider the set of micro-sequences $\Omega''$ rather than a single micro-sequence $\mathbf{s}''$ (see remarks about Algorithms 2 and 3 below). Reinterpret step 4 of Algorithm 1 to mean "repeat steps 2 and 3 until the estimates for every $\mathbf{s}' \in \Omega'$ and $\{\mathcal{M}''_i\}$ converge".

(b) In Algorithm 2, step 1, build the substrings $subs(i)$ for each macro-state $i$ by separating every micro-sequence $\mathbf{s}'' \in \Omega''$ according to the corresponding macro-sequence $\mathbf{s}' \in \Omega'$. Once all substrings have been collected in $subs(i)$, step 2 of Algorithm 2 can be run without change.

(c) Algorithm 3 needs to be run for each micro-sequence $\mathbf{s}'' \in \Omega''$ in order to determine the corresponding macro-sequence $\mathbf{s}'$. No changes are necessary to this algorithm.

An important issue when working with multiple micro-sequences has to do with the initial randomization of the corresponding macro-sequence. Suppose that a macro-model is able to generate two sequences AB and ABC with equal probability. Also, suppose that the observed micro-sequences are XXXYYY and XXXYYYZZZ. When randomizing the macro-sequence for each of these micro-sequences, it seems more appropriate to relate XXXYYY to AB and XXXYYYZZZ to ABC rather than the other way around. However, the macro-sequence will be drawn randomly from AB and ABC so any assignment is possible. The possible scenarios are the following:

(a) The macro-sequences for XXXYYY and XXXYYYZZZ are both expansions of AB.

(b) The macro-sequence for XXXYYY is an expansion of AB and the micro-sequence for XXXYYYZZZ is an expansion of ABC.

(c) The macro-sequence for XXXYYY is an expansion of ABC and the micro-sequence for XXXYYYZZZ is an expansion of AB.

(d) The macro-sequences for XXXYYY and XXXYYYZZZ are both expansions of ABC.

Clearly, (b) is the most desirable option, since it leads to the simplest model of all, and the one which is able to generate the observed micro-sequences with highest log-probability. In this case, the micro-model for A just emits X's, the micro-model for B just emits Y's, and the micro-model for C just emits Z's.

In practice, this issue manifests itself in the different lengths of traces, where a trace is the (micro-)sequence of events recorded for a given case id in the event log. In general, the longest micro-sequences should be assigned to the longest sequences that are drawn from the macro-model $\mathcal{M}'$. Therefore, in step 1 of Algorithm 1, rather than drawing a sequence $\mathbf{s}$ for each given micro-sequence $\mathbf{s}'' \in \Omega''$, one draws $N = |\Omega''|$ such sequences at once, where $N$ is the number of given micro-sequences. Then one sorts the micro-sequences and the random sequences by length and only then performs the assignment between them. This simple optimization allows Algorithm 1 to provide better results, finding micro-models which are able to produce the given micro-sequences with higher log-probability.

### *4.5 Running the algorithm multiple times*

The first step in Algorithm 1 is to obtain a random initialization of the macro-sequence (or macro-sequences, if several micro-sequences are being used). Since this initialization is random, the algorithm may produce different solutions across multiple runs. This may yield slightly different micro-models, where a certain part of the observed low-level behavior ends up being attributed to one macro-activity instead of another.

For example, for the micro-sequence $\mathbf{s}'' =$ XYZYZZZXY the random initialization step in Algorithm 1 may yield the macro-sequence $\mathbf{s}' =$ AAABBBCCC. Then it becomes clear which event belongs to which macro-activity:

```
XYZYZZZXY
|||||||||
AAABBBCCC
```

From the mapping above one can immediately derive the solution (i.e. the micro-models) in Figure 4 by following the procedure described in Algorithm 2.

However, if the macro-sequence is initialized to $\mathbf{s}' =$ AABBBBBCC then we will have the following mapping:

```
XYZYZZZXY
|||||||||
AABBBBBCC
```

This mapping originates a different solution, where the micro-models $\mathcal{M}''_{\mathsf{A}}$ and $\mathcal{M}''_{\mathsf{C}}$ will be two simple models that produce just XY and $\mathcal{M}''_{\mathsf{B}}$ will be a more complicated model that begins with Z and from Z it may go to Y, or to Z, or to the end. The log-probability of the micro-sequence $\mathbf{s}'' =$ XYZYZZZXY in this solution is $-4.16$, while in the former solution the log-probability is $-1.39$. Therefore, the former solution is preferred. Again, the preferred solution is the one that is able to produce the given micro-sequence with the highest log-probability.

This concept can be easily extended to the case of multiple micro-sequences. Once a solution is obtained, then summing up the log-probabilities for all micro-sequences yields a measure of the relative quality of that solution. Therefore, another optimization that we use is to run Algorithm 1 multiple times in order to pick the best result, i.e. the set of micro-models that are able to produce the given micro-sequences with the highest overall log-probability. The number of runs is arbitrary and can be chosen by the user. In practice, we have found that a number of runs $K$ somewhere in between $\sqrt{N} \leq K \leq N$, where $N$ is the number of input micro-sequences, is usually enough to obtain the best possible solution.

Before we proceed, Algorithm 4 summarizes the adaptations that have been done to Algorithm 1 in order

to take into account the optimizations from this section (Section 4.5) and the previous section as well (Section 4.4). It includes the adaptations to deal with multiple micro-sequences in steps 1–4; it includes the sorting by length of micro-sequence and macro-sequences in steps 1(a)–1(d); and it includes multiple runs and the choice of the best solution in steps A–B.

**Algorithm 4** Estimate the micro-models $\{\mathcal{M}_i''\}$ and the macro-sequences $\Omega'$ from the macro-model $\mathcal{M}'$ and the micro-sequences $\Omega''$ (replaces Algorithm 1)

A. Run the following steps $K$ times, where $K$ is to be chosen by the user:

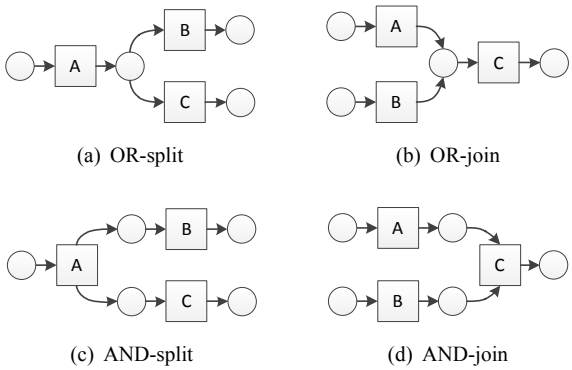
   1. Initialize the macro-sequences $\Omega'$ according to the following procedure:
      (a) Draw $N = |\Omega''|$ random sequences from the Markov model $\mathcal{M}'$, where $N$ is the number of micro-sequences in $\Omega''$. Let $\Omega'$ be the multiset of random sequences just drawn.
      (b) Sort the micro-sequences $\Omega''$ and the sequences in $\Omega'$ by length.
      (c) After sorting, take each sequence in $\Omega'$ to be the sequence that corresponds to the micro-sequence at the same position in $\Omega''$.
      (d) Expand each sequence in $\Omega'$ to become the macro-sequence for the corresponding micro-sequence in $\Omega''$ (the expansion procedure is explained in Section 4.1)
   2. Feed all micro-sequences $\Omega''$ and their corresponding macro-sequences $\Omega'$ to Algorithm 2. In step 1 of Algorithm 2, separate all micro-sequences according to their respective macro-sequences.
   3. For each micro-sequence $\mathbf{s}'' \in \Omega''$, run Algorithm 3 in order to determine the most likely macro-sequence $\mathbf{s}'$. Update $\Omega'$ so that $\Omega'$ contains macro-sequences obtained from Algorithm 3.
   4. Go back to step 2 and repeat until both $\Omega'$ and $\{\mathcal{M}_i''\}$ converge.
   5. Use the macro-model $\mathcal{M}'$ and the micro-models $\{\mathcal{M}_i''\}$ to compute the log-probability of producing each micro-sequence $\mathbf{s}'' \in \Omega''$. Sum the log-probabilities for all micro-sequences in $\Omega''$.

B. Out of the $K$ solutions found in step A, return the solution that has the highest value for the sum of log-probabilities.

## 5 Discovery of basic workflow patterns

The macro-model that was used as a running example in the previous sections is able to generate just the simple sequence ABC. However, in practice there are many types of behavior that a business process model may contain, such as OR-splits, AND-joins, loops, etc. These types of behavior have been identified and classified as a set of *workflow patterns* (van der Aalst et al., 2003). Given that these patterns are very common in business process modeling, it is likely that any high-level process model will contain at least some of the most basic patterns.

In this section, our goal is to carry out a sanity check of the proposed approach to ensure that it is possible to discover the behavior of agents in macro-level processes that involve more than just a linear sequence of steps. In particular, we would like to ensure that the proposed approach is able to deal with at least the most basic patterns, namely OR-splits, OR-joins, AND-splits, and AND-joins, as shown in Figure 6.




(a) OR-split (b) OR-join

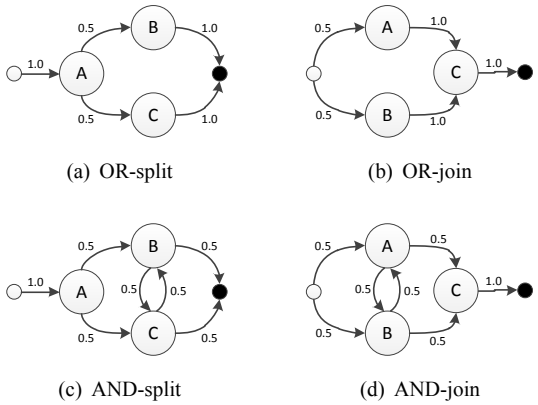
(c) AND-split (d) AND-join

**Figure 6** Basic control-flow patterns expressed as Petri nets

### 5.1 Describing patterns as Markov models

There is no problem in representing an OR-split or an OR-join by means of a Markov model, as in Figures 6(a) and 6(b), respectively. However, the parallel nature of AND-splits and AND-joins cannot be captured exactly by a first-order Markov chain. In Figure 6(c), the AND-split is able to generate the sequences ABC and ACB. Assuming that these two sequences can occur with equal probability, we can derive a Markov model from the sequences ○ABC● and ○ACB●. The resulting Markov model is shown in Figure 7(c). The model begins always with an A, followed by either B or C. After B, a C may follow or the sequence may end. In a similar way, after C a B may follow or the sequence may end.

The problem with the model in Figure 7(c) is that it allows for more behavior than just the sequences ABC and ACB. In fact, this model may produce several iterations involving B and C before the sequence ends. The model also allows for shorter sequences, namely ○AB● and ○AC●. In the particular case of an AND-split such as this one, which involves just two activities in parallel, the problem could be addressed by making use of a second-order Markov model. Such model would take into account the two previous states when determining the next state. However, increasing the order of the Markov model is not




(a) OR-split (b) OR-join

(c) AND-split (d) AND-join

**Figure 7** Basic control-flow patterns expressed as Markov models

a scalable approach since, in general, a process model may include an arbitrary number of activities in parallel.

A similar situation occurs with the AND-join pattern shown in Figure 6(d). Here, the model specifies that either ABC or BAC may occur. In any case, both A and B must have been completed before C can occur; hence, this is also known as the *synchronization* pattern (van der Aalst et al., 2003). Assuming that both sequences can occur with equal probability, one can derive from ○ABC● and ○BAC● the Markov model shown in Figure 7(d). Again, this model allows for more behavior than originally intended, namely: several A's and B's may precede C, and the shorter sequences AC and BC are also possible.

Both the AND-split model of Figure 7(c) and the AND-join model of Figure 7(d) suffer from a problem of *underfitting* (van der Aalst et al., 2010) since they allow for more behavior than what can actually be produced by a true AND-split or AND-join, respectively. Anyway, for the purpose of our experiments this is not an impeding problem. On the contrary, if our approach succeeds in discovering the behavior produced by the models in Figure 7(c) and Figure 7(d), then it will certainly succeed in discovering the behavior of an AND-split and an AND-join, since the latter is a subset of the former. Therefore, in our experiments we used the four models shown in Figure 7.

### 5.2 Discovering the micro-models $\{\mathcal{M}_i''\}$

In this experiment, the goal is to check whether Algorithm 4 is able to discover the micro-models from a set of input micro-sequences and a given macro-model, where the macro-model is one of the models in Figure 7. For simplicity, we use the same micro-models $\{\mathcal{M}_{\mathsf{A}}'', \mathcal{M}_{\mathsf{B}}'', \mathcal{M}_{\mathsf{C}}''\}$ as in Figure 4, together with each macro-model in Figure 7 to obtain a different hierarchical Markov model. From each of these hierarchical models, we generate $N = 100$ micro-sequences by simulation. Then we run Algorithm 4 on these micro-sequences and the given macro-model to re-discover the micro-models $\{\mathcal{M}_{\mathsf{A}}'', \mathcal{M}_{\mathsf{B}}'', \mathcal{M}_{\mathsf{C}}''\}$. As explained in Section 4.5, step 'A' of Algorithm 4 may be run an arbitrary number of times $K$. For this experiment, we have chosen $K = 10$.

In all cases, Algorithm 4 discovers the correct micro-models, where $\mathcal{M}_{\mathsf{A}}''$ generates XYZ, $\mathcal{M}_{\mathsf{B}}''$ generates YZ(Z), and $\mathcal{M}_{\mathsf{C}}''$ generates ZXY. The transition probabilities $\mathbf{T}_{\mathsf{B}}''(\mathsf{Z},\mathsf{Z})$ and $\mathbf{T}_{\mathsf{B}}''(\mathsf{Z},\bullet)$ are not exactly equal to ½ as in Figure 4 since in a random draw of $N = 100$ sequences it may happen that there are not exactly as many transitions from Z to Z as there are from Z to ●. In any case, Algorithm 4 is able to capture the exact transition probabilities that can be found in the input micro-sequences.

However, there are marked differences in terms of number of iterations and computation time between OR-patterns and AND-patterns. In a sense, these differences are to be expected since the AND-patterns in Figure 7 allow for more (non-deterministic) behavior than the OR-patterns. Table 1 shows the results of the experiment. The number of Expectation-Maximization iterations (steps 2–4 of Algorithm 4) are reported for each run of step 'A'. Here it is apparent that the AND-patterns require more iterations for convergence, which is attributed to the fact that the micro-sequences generated for these models tend to be longer (9 symbols on average) than the micro-sequences obtained from the OR-patterns (6 symbols on average).

The most noticeable difference is in the total running time. While the micro-models for a macro-model that contains an OR-pattern can be discovered rather quickly, discovering the same micro-models for a macro-model that contains an AND-pattern takes significantly longer. The numbers vary widely: the total running time for AND-patterns are in the range of a few seconds to a few minutes, whereas for OR-patterns this is always under one second.

The difference in the number of iterations is not large enough to explain the difference in running time, so there must be another factor that explains the relatively long running times for AND-patterns. Going deeper into the experimental results reveals that most of that running time is spent during the first run of step 3 in Algorithm 4. It seems that when a macro-sequence is being computed for the first time, the recursive tree that Algorithm 3 goes across is too large, and therefore most of the time is spent in determining the best candidate macro-sequence, since there are many candidates to choose from. But once the first macro-sequence is obtained, the following iterations of steps 2–4 in Algorithm 4 run fairly quickly, almost as quickly as in the case of OR-patterns.

### 5.3 Macro-models with loop patterns

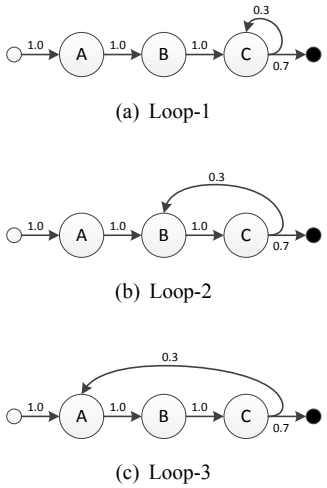
In the previous experiments there was a loop in one of the micro-models (specifically, in $\mathcal{M}_{\mathsf{B}}''$ which can produce YZZ...) but not in the macro-model. Here we investigate what happens when there is a loop in the macro-model. Such loop may include an arbitrary number of activities; when it includes just one or two activities, it is called a *short loop* (de Medeiros et al., 2004). For simplicity, using the three activities A, B, and C as in the previous examples, it is possible to have loops of length 1, length 2, and

| | Number of EM iterations for each run | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pattern | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg. | Total time |
| OR-split | 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2.9 | $<$ 1s |
| OR-join | 3 | 2 | 4 | 3 | 4 | 2 | 3 | 3 | 4 | 3 | 3.1 | $<$ 1s |
| AND-split | 4 | 4 | 3 | 5 | 6 | 5 | 4 | 4 | 3 | 4 | 4.2 | $\sim$ 41s |
| AND-join | 3 | 5 | 4 | 4 | 7 | 2 | 5 | 6 | 5 | 7 | 4.8 | $\sim$ 37s |

**Table 1** Sample results on the four models of Figure 7 with $N = 100$ sequences and $K = 10$ runs

| | Number of EM iterations for each run | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pattern | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg. | Total time |
| Loop-1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2.0 | $<$ 1s |
| Loop-2 | 4 | 3 | 5 | 5 | 4 | 5 | 5 | 5 | 2 | 3 | 4.1 | $\sim$ 32s |
| Loop-3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 2 | 2.3 | $\sim$ 95s |

**Table 2** Sample results for the three loops in Figure 8 with $N = 100$ sequences and $K = 10$ runs




(a) Loop-1

(b) Loop-2

(c) Loop-3

**Figure 8** Basic control-flow patterns expressed as Markov models

length 3. Figure 8 shows an example of each. As before, we use the micro-models $\{\mathcal{M}''_{\mathsf{A}}, \mathcal{M}''_{\mathsf{B}}, \mathcal{M}''_{\mathsf{C}}\}$ of Figure 4 together with each macro-model in Figure 8 to obtain a different hierarchical Markov model. From each of these models, we generate $N = 100$ micro-sequences, and we run Algorithm 4 with $K = 10$.

The results are shown in Table 2. Algorithm 4 discovers the correct micro-models fairly quickly for the case of loop-1, whereas for loop-2 and loop-3 it takes significantly longer. This can be explained by the fact that $\mathcal{M}''_{\mathsf{C}}$ produces the simple sequence ZXY, whereas $\mathcal{M}''_{\mathsf{B}}$ contains a loop of its own (a low-level loop of length 1 where Z repeats in YZZ...). Since both loop-2 and loop-3 include activity B, every time B executes it introduces its own repetitions of Z in the micro-sequence. Finding the best macro-sequence for such micro-sequence takes longer, since there are more candidates to choose from. Ultimately, this is the same reason why it took longer to find the solution for AND-patterns in Table 1: looking at Figure 7(c) and 7(d), one can see that these AND-patterns, when expressed as Markov chains, contain a sort of loop as well.

## 6 Case study: a purchase process

In this section we turn to the application of Algorithm 4 in a practical scenario that involves a purchase process. A description of this process (i.e. the macro-model) is available as a BPMN diagram. At the micro-level, the process is implemented as a set of interactions between agents in an agent-based simulation platform. During simulation, an event log is recorded; this event log contains the micro-sequences that can be used to discover the micro-models associated with the behavior of agents when performing the purchase process. In this scenario, Algorithm 4 should be able to re-discover the original micro-models that have been used to implement the purchasing process in the agent-based platform. In this case study, we use the Agent-Object-Relationship (AOR) platform introduced by (Wagner, 2004).

### *6.1 Agent-based simulation with AOR*

Agent-based modeling and simulation (Bonabeau, 2002; Axelrod, 2006; Davidsson et al., 2007) is an effective means to study the behavior of systems involving the actions and interactions of autonomous agents. For example, agent-based systems have been used to study the dynamics of financial markets by generating time series data that resemble the evolution of stock prices (Hoffmann et al., 2006; Neri, 2012). Here we use an agent-based system to simulate the execution of a business process, thereby generating an event log with low-level events.

Although there are several platforms for agent-based simulation (Railsback et al., 2006), we turn our attention to the Agent-Object-Relationship (AOR) approach introduced by (Wagner, 2004), which can be used to model and simulate business processes (Wagner et al., 2009).

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

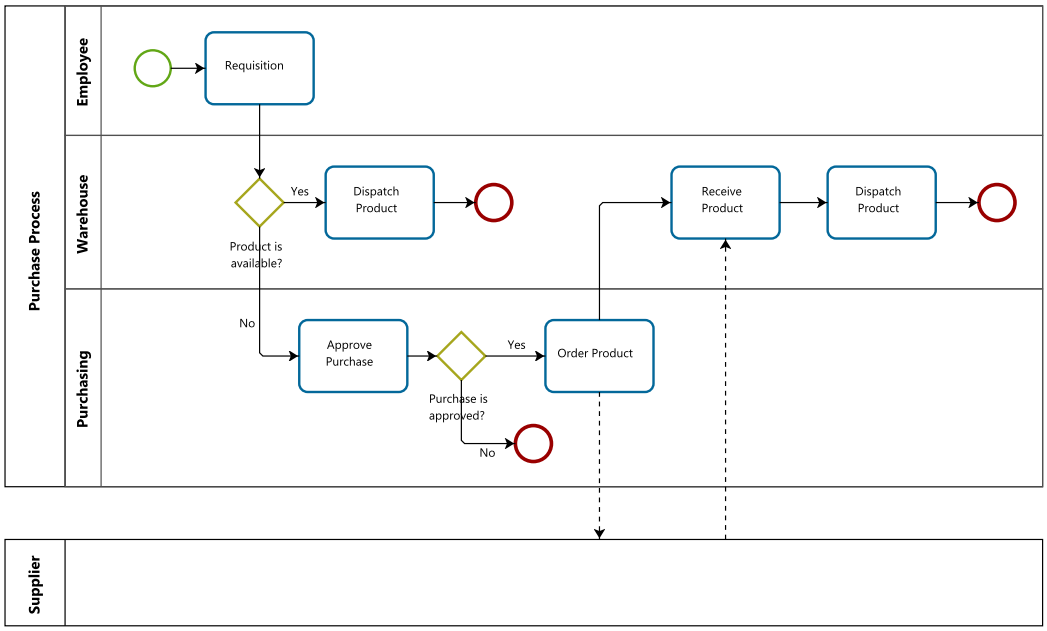



**Figure 9** Macro-level description of the purchase process

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:

$$X \xrightarrow{M} Y$$

In the AOR system, the specification of a new simulation scenario begins by defining a set of *entity types*. These entity types include the types of agents, messages and events that will be used in the scenario. The behavior of agents is specified by means of *agent rules*. Typically, an agent rule defines that when a certain message is received, another message is produced and sent to some other agent. Since the rules for each agent are defined separately, the simulation scenario is effectively implemented in a decentralized way by the combined behavior of agents.

Another kind of rules that exist in the AOR system are *environment rules*. Basically, these have to do with the occurrence of exogenous events and they define what should be done when such events occur. Typically, an environment rule specifies that when a certain event occurs, a message should be sent to some agent. Sending this message then triggers a rule of the receiving agent, which creates a chain of message exchanges that puts the whole simulation in motion.

Environment rules also have the ability to create and destroy agents. This is especially useful to simulate, for example, the arrival (or leaving) of new customers. The agents that are created dynamically at run-time must be of a certain type that has already been defined before. These agents also have rules and they participate in the simulation by exchanging messages with other agents. A set of *initial conditions* for the simulation scenario specifies which agents already exist at the beginning of the simulation. The initial conditions also include a schedule for the occurrence of at least one exogenous event to trigger the simulation.

All of these constructs (i.e. entity types, agent rules, environment rules, and initial conditions) are specified using an XML-based language known as AOR Simulation Language (AORSL) (Nicolae et al., 2010). This language also allows embedding Java code in order to implement auxiliary functions and expressions. In fact, the scenario specification in AORSL is transformed into Java code by the AOR system. Running the simulation amounts to compiling and running the auto-generated Java code.

### *6.2 Implementing the purchase process*

Our case study is based on the implementation of a purchase process in the AOR system. At the macro-level, the process is represented as a BPMN model in Figure 9 and can be described as follows:

> In a company, an employee needs a certain commodity (e.g. a printer cartridge) and submits a request for that product to the warehouse. 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 approved by the purchasing department. If the purchase is not approved, the process ends at that point. On the other hand, if the purchase is approved, the

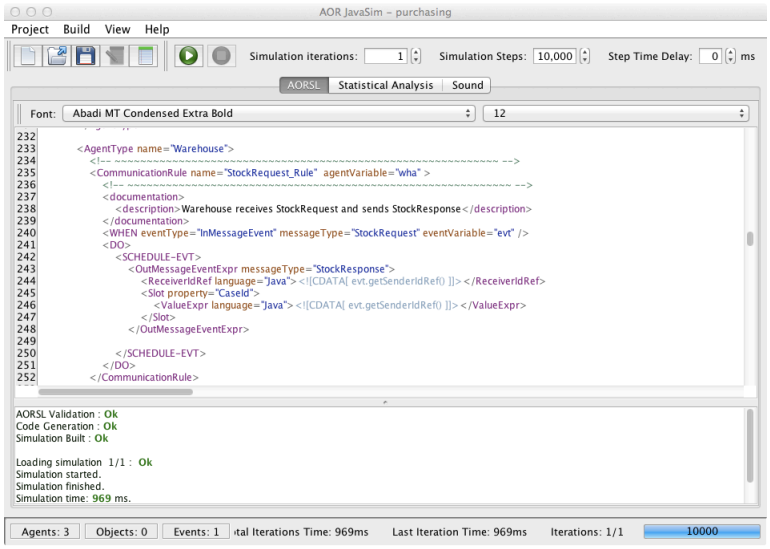



**Figure 10** The AOR Simulator application with the purchasing scenario specification

purchasing department orders and pays for the product from the supplier. The supplier delivers the product to the warehouse, and the warehouse dispatches the product to the employee.

This process was implemented in the AOR system, using AORSL to specify the message exchanges between agents. There are four types of agents: Employee, Warehouse, Purchasing, and Supplier. There is one pre-existing instance of the Warehouse agent and one pre-existing instance of the Purchasing agent. However, there are multiple instances of the Employee agent created at run-time (each instance is created at the start of the simulation 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, since this has no effect in the results.

The process includes the following message exchanges:

- Requisition
  - Employee —StockRequest→ Warehouse
  - Warehouse —StockResponse→ Employee
- Dispatch product
  - Employee —FetchProduct→ Warehouse
  - Warehouse —ProductReady→ Employee
  - Employee —ProductReceived→ Warehouse
- Approve purchase
  - Employee —PurchaseRequest→ Purchasing
  - Purchasing —InfoRequest→ Employee
  - Employee —InfoResponse→ Purchasing
  - Purchasing —ApprovalResult→ Employee
- Order product
  - Purchasing —PurchaseOrder→ Supplier
  - Supplier —PaymentTerms→ Purchasing
  - Purchasing —PaymentVoucher→ Supplier
- Receive product
  - Supplier —DeliveryNote→ Warehouse
  - Warehouse —ProductAvailable→ Employee

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 of 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).

Figure 10 shows a screenshot of the resulting AORSL specification in the AOR simulation environment. At the top of Figure 10, the "Build" menu is used to generate the Java code from the AORSL specification shown in the middle pane. The "play" button in the toolbar is used to launch the simulation with the parameters that can be configured to the right-hand side of that button. From these, the most important parameter is the number of simulation steps, since this controls how long the simulation will run (in the agents' time scale). The remaining parameters, "simulation iterations" and "step time delay", are used to run the simulation multiple times and to insert a time delay between consecutive steps, respectively.

### 6.3 *Generating the event log*

Simulating this process in AOR produces an event log with an AOR-specific XML structure. Basically, this XML-based event log has an entry for each simulation step.

Each of these entries records: which agents were active at that point in time; which exogenous events (if any) were received by the environment and which environment rules were activated by those events; which messages (if any) were received by each agent and which agent rules were activated by those messages; and, finally, which messages were produced as a result of activating any of those rules.

In the event log, it is possible to recognize each process instance as being associated with a different instance of the Employee agent. When the AOR system creates a new Employee agent, it assigns a unique number to that agent, and this number increases automatically for each new Employee being created. The numbers are usually negative for dynamically created agents, in order to distinguish from pre-existing agents which are typically assigned positive numbers. For example, Warehouse is agent no. 1, Purchasing is agent no. 2, and Supplier is agent no. 3. On the other hand, the Employee agents have numbers such as -1, -2, -3, etc., and these values are used as the case id for the corresponding process instances.

The AOR system records in the event log which agent (no.) sends each message and which agent receives it. Since Employee agents have no communication with each other in this process, whenever a message is being sent to or received from an Employee agent it is possible to figure out immediately the process instance to which the message belongs. In messages that are exchanged between agents other than Employee agents, our implementation specifies that these messages must carry a property called CaseId which contains the Employee number that corresponds to that process instance. This way, it is possible to determine the process instance for every message.

These conventions make it possible to convert the AOR event log automatically into an event log in the form of Table 3. As explained above, the *case id* column corresponds to the Employee number; the *sender* and *receiver* columns could be agent numbers as well, but instead they have been converted to the corresponding agent types; and the *timestamp* refers to the time step of the AOR simulation when the event occurred (if needed, this timestamp can be converted to a more usual format with date and time). For this case study, we ran a simulation with 10,000 steps. This produced an event log with 140 process instances and a total of 1136 events.

| *case id* | *sender* | *message* | *receiver* | *timestamp* |
|---|---|---|---|---|
| -1 | Employee | StockRequest | Warehouse | 2 |
| -1 | Warehouse | StockResponse | Employee | 4 |
| -1 | Employee | FetchProduct | Warehouse | 5 |
| -1 | Warehouse | ProductReady | Employee | 6 |
| -1 | Employee | ProductReceived | Warehouse | 7 |
| -2 | Employee | StockRequest | Warehouse | 82 |
| -2 | Warehouse | StockResponse | Employee | 84 |
| -2 | Employee | PurchaseRequest | Purchasing | 85 |
| -2 | Purchasing | InfoRequest | Employee | 86 |
| -2 | Employee | InfoResponse | Purchasing | 87 |
| -2 | Purchasing | ApprovalResult | Employee | 88 |
| -2 | Purchasing | PurchaseOrder | Supplier | 89 |
| ... | ... | ... | ... | ... |

**Table 3** Example of an event log obtained from a simulation in the AOR system

### *6.4 Mining the event log: control-flow perspective*

From an event log in the form of Table 3 it is possible to get different types of micro-sequences, depending on which column is chosen for analysis. Here there are three possible choices: the *sender* column, the *message* column, and the *receiver* column. The *message* column can be used to study the sequence of messages, while the sender and receiver columns can be used to derive interaction models.

In any case, the events can be grouped by case id and sorted by timestamp. For example, for case id -1 we have the sequence of messages,

$$\mathsf{StockRequest} \rightarrow \mathsf{StockResponse} \rightarrow \mathsf{FetchProduct} \rightarrow \mathsf{ProductReady} \rightarrow \mathsf{ProductReceived}$$

and the sequence of senders,

$$\mathsf{Employee} \rightarrow \mathsf{Warehouse} \rightarrow \mathsf{Employee} \rightarrow \mathsf{Warehouse} \rightarrow \mathsf{Employee}$$

and the sequence of receivers,

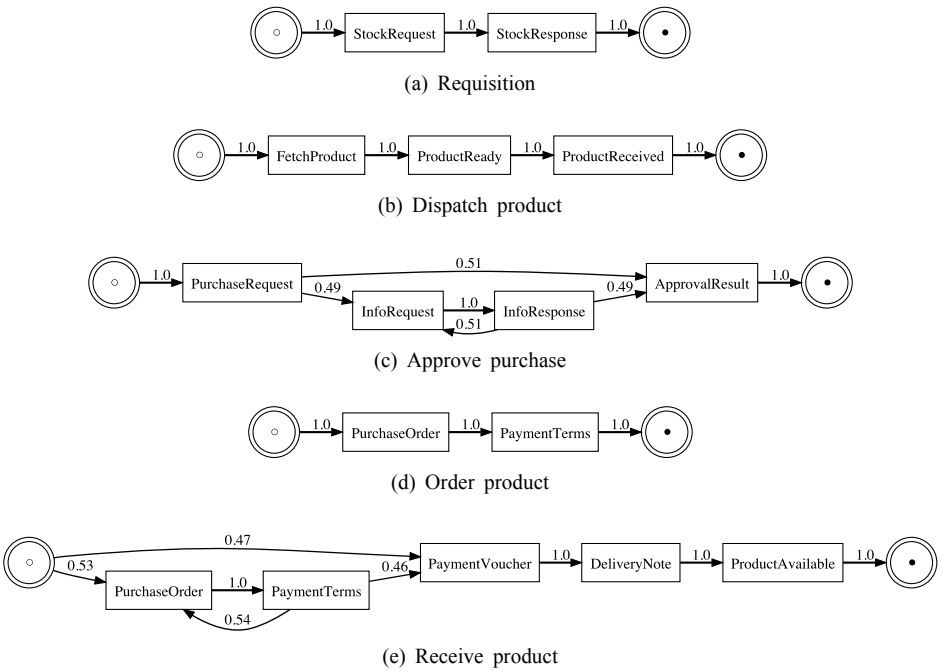
$$\mathsf{Warehouse} \rightarrow \mathsf{Employee} \rightarrow \mathsf{Warehouse} \rightarrow \mathsf{Employee} \rightarrow \mathsf{Warehouse}.$$

These different kinds of micro-sequences allow us to study both the control-flow perspective (from the sequence of messages) and the organizational perspective (from either the sequence or senders or the sequence or receivers). Both of these perspectives are well-known in the process mining literature (Mans et al., 2008; Bozkaya et al., 2009). While the control-flow perspective addresses the sequence of activities in the process, the organization perspective allows studying other aspects such as the handover of work between agents (Song and van der Aalst, 2008).

Our goal is to use Algorithm 4 to discover the micro-model for each macro-level activity in Figure 9. We start with the sequences of messages exchanged between agents. There are 140 process instances in the event log and therefore there are 140 such micro-sequences. But in addition to the set of micro-sequences $\Omega''$, Algorithm 4 also requires the macro-model $\mathcal{M}'$. This we obtained by producing a Markov-chain representation of the macro-level process shown in Figure 9, where we assumed alternative branches to be equally likely to occur (i.e. $0.5$ probability for both branches coming out of a two-way gateway).

Feeding the $N = 140$ micro-sequences and the macro-model to Algorithm 4, and setting a number of $K = 50$ runs, we obtained the micro-models shown in Figure 11.

The results in Figure 11 represent the correct behavior, except for the fact that PurchaseOrder, PaymentTerms and, PaymentVoucher appear in Figure 11(e) ("Receive product") rather than only in the Figure 11(d) ("Order product"). Since these two macro-level activities always occur together and sequentially in the model of Figure 9, there is no way to determine that those events belong to the first activity and not to the second. In the random initialization (step 1 of Algorithm 4), some of these events are assigned to "Order product" and others are assigned to "Receive product" indistinctly. Also, when the loop between PurchaseOrder




(a) Requisition

(b) Dispatch product

(c) Approve purchase

(d) Order product

(e) Receive product

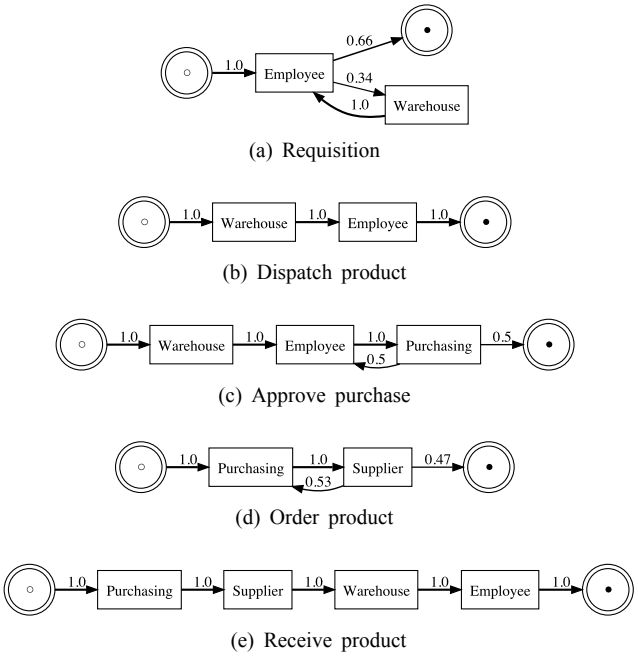
**Figure 11** Results obtained for an AOR simulation with 10,000 steps

and PaymentTerms repeats, it becomes more likely that those events are distributed evenly between "Receive product" and "Order product".

### 6.5 Mining the event log: organizational perspective

Regarding the organizational perspective, with the micro-sequences for senders and the micro-sequences for receivers we obtained the results shown in Figure 12 and Figure 13, respectively. The results in Figure 12 have a few mismatches in comparison to the actual sequence of message exchanges associated with each macro-level activity. For example, the loop between Employee and Warehouse that appears in Figure 12(a) ("Requisition") should have appeared instead in Figure 12(b) ("Dispatch product"). Again, this is related to the way macro-sequences are being initialized and to the solution that Algorithm 4 converges to. Also, the Warehouse has been captured in Figure 12(c) ("Approve purchase"), but it should not have been. The interaction between Purchasing and Supplier in Figure 12(d) ("Order product") is correct, but these same agents appear in Figure 12(e) ("Receive product"), where only Warehouse and Employee should appear.

The results in Figure 13 are more accurate. The micro-models in Figure 13(a), Figure 13(b) and Figure 13(c) are absolutely correct. The micro-model in Figure 13(d) ("Order product") is also correct, except that in reality the activity ends with Supplier receiving a message (PaymentVoucher) rather than Purchasing. The message exchange that is missing in "Order product" ended up being captured in "Receive product". In fact, the micro-model in Figure 13(e) ("Receive product") has an extra Supplier which belongs to the "Order product" activity. This is similar to what happened in the results of Figure 11, where




(a) Requisition

(b) Dispatch product

(c) Approve purchase

(d) Order product

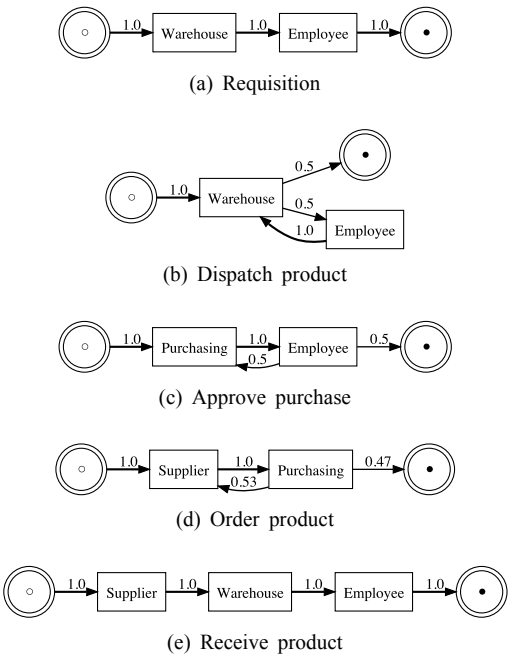
(e) Receive product

**Figure 12** Interaction between agents from the perspective of message senders

part of the exchanges for "Order product" ended up being captured by "Receive product", as previously explained.

### 6.6 Comparison with ProM

In the field of process mining, the ProM framework (van Dongen et al., 2005) is a reference tool that includes an implementation of many process mining techniques available today. ProM is called a "framework" since it is an application that can be extended with third-party




(a) Requisition

(b) Dispatch product

(c) Approve purchase

(d) Order product

(e) Receive product

**Figure 13** Interaction between agents from the perspective of message receivers

plug-ins, where each plug-in implements a different mining technique. In this section, our aim is to compare the mining techniques available in ProM with the approach developed in this work, in order to highlight the differences. Although ProM includes numerous plug-ins with advanced capabilities, here we make use of only the most basic features that are related to the perspectives analyzed above, namely the control-flow perspective and the organizational perspective. For this purpose, we use the *heuristics miner* (Weijters et al., 2006) and the *social network miner* (Song and van der Aalst, 2008), respectively.

First, we converted the AOR event log into the MXML format used by ProM (van Dongen and van der Aalst, 2005). Then we opened the file in ProM and invoked the heuristics miner plug-in, which produced the model shown in Figure 14. In this figure, we have all the low-level behavior captured in a single model. It is possible to recognize some features of the business process: for example, the loop between InfoRequest and InfoResponse is immediately recognizable, as well as the loop between PurchaseOrder and PaymentTerms. By following the flow it is also possible to recognize an OR-split in StockResponse and an OR-join in FetchProduct. However, from this model it is hard to identify the relationship between these low-level events and the high-level activities in Figure 9.

In contrast, our approach produces the results shown in Figure 11, where each graph describes the low-level behavior that occurs within each high-level activity. It is this sort of advantage that we wished to highlight in comparison with the traditional process mining techniques available in ProM. In practice, the analysis of behavior based exclusively on micro-level events recorded in an event log (as in Figure 14) often leads to very large and complex models that are difficult to interpret and that are referred to as *spaghetti models* (van der Aalst and Günther, 2007). An advantage of our approach is that this behavior is partitioned across a set of macro-level activities and therefore becomes easier to understand and interpret.

An analysis of the same event log with the social network miner available in ProM is presented in Figure 15. These results were obtained using the metric *handover of work* (van der Aalst et al., 2005) both for the *sender* and for the *receiver* columns in the event log of Table 3. In both graphs of Figure 15 it is clear that the employee has an interaction with the warehouse, and a separate interaction with the purchasing department. On the other hand, the warehouse and the purchasing department do not interact directly. In a similar way, the employee does not interact with the supplier directly; it is the purchasing department who interacts with the supplier. Hence, some useful conclusions can be drawn about the process, even from models that are obtained from low-level events alone. However, the micro-models in Figure 12 and 13 have the distinct advantage of showing the interactions that take place within each high-level activity. Again, as in the case of the control-flow perspective, partitioning the observed behavior into a set of high-level activities facilitates the analysis of what actually takes place at run-time.

## 7 Conclusion

In this work we have introduced a hierarchical Markov model to capture the relationship between the macro-level activities in a business process model and the micro-level events recorded in an event log. We have also developed an Expectation-Maximization procedure to estimate the parameters of such model. This approach can be used as a process mining technique in scenarios where an event log is available, together with a high-level description of the business process.

In this context, we have shown that the proposed technique (Algorithm 4) is able to discover the micro-models for each macro-activity. This was demonstrated in experiments with a set of basic workflow patterns, as well as in a case-study application where we used a state-of-the-art agent-based platform to implement a purchase process and to generate the event log through simulation. As illustrated in the case study, the approach can be used to analyze both the control-flow and the organizational perspectives.

In future work, we will be looking at the possibility of using additional heuristics in the random initialization step of Algorithm 4. Sorting the micro- and macro-sequences by length, as explained in Section 4.4, provides a major improvement in terms of the log-likelihood of the solutions found by the algorithm. It is possible that additional heuristics, based on the fact that the macro-model is known, will contribute to find the best possible solution in fewer runs. On the other hand, we will also be looking forward to further developing the possible connections and interplay between process mining and agent-based simulation for the discovery and analysis of business processes.

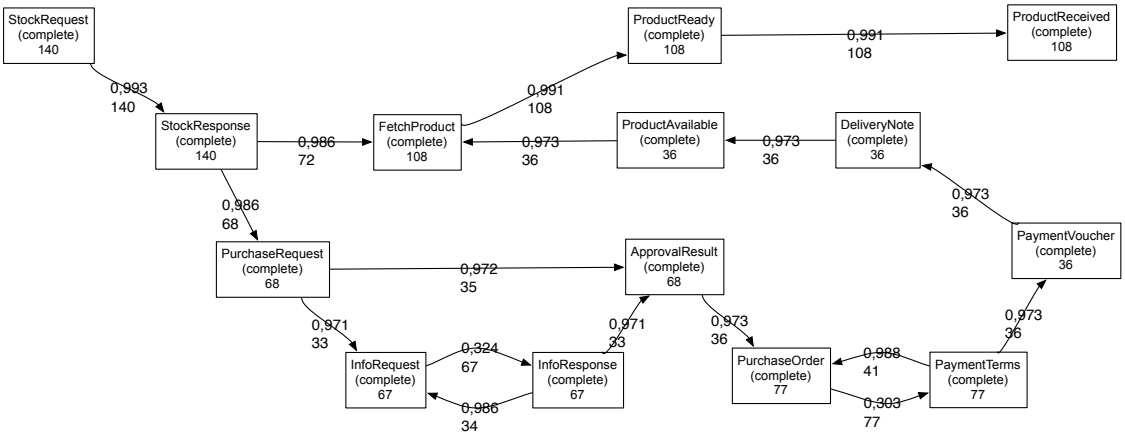
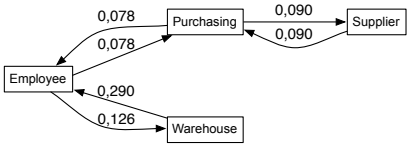
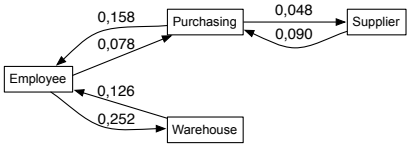



**Figure 14** Result obtained using the heuristics miner plug-in in ProM




(a) sender perspective




(b) receiver perspective

**Figure 15** Results obtained using the social network miner in ProM

## References


Axelrod, R. (2006). Agent-based modeling as a bridge between disciplines. In Tesfatsion, L. and Judd, K. L., editors, *Handbook of Computational Economics*, volume 2, chapter 33, pages 1565–1584. Elsevier.

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *PNAS*, 99(Suppl 3):7280–7287.

Bose, R. P. J. C., Verbeek, H. M. W., and van der Aalst, W. M. P. (2012). Discovering hierarchical process models using ProM. In *CAiSE Forum 2011*, volume 107 of *LNBIP*, pages 33–48. Springer.

Bozkaya, M., Gabriels, J., and van der Werf, J. (2009). Process diagnostics: A method based on process mining. In *International Conference on Information, Process, and Knowledge Management (eKNOW '09)*, pages 22–27.

Collet, C. and Murtagh, F. (2004). Multiband segmentation based on a hierarchical markov model. *Pattern Recognition*, 37(12):2337–2347.

Cook, D., Youngblood, M., and Das, S. (2006). A multi-agent approach to controlling a smart environment. In *Designing Smart Homes*, volume 4008 of *LNCS*, pages 165–182. Springer.

Davidsson, P., Holmgren, J., Kyhlbäck, H., Mengistu, D., and Persson, M. (2007). Applications of agent based simulation. In *7th International Workshop on Multi-Agent-Based Simulation*, volume 4442 of *LNCS*. Springer.

de Medeiros, A. K. A., van Dongen, B., van der Aalst, W. M. P., and Weijters, A. J. M. M. (2004). Process mining: Extending the $\alpha$-algorithm to mine short loops. BETA Working Paper Series WP 113, Eindhoven University of Technology.

de Medeiros, A. K. A. and Weijters, A. J. M. M. (2005). Genetic process mining. *Lecture Notes in Computer Science*, 3536:48–69.

Demonceaux, C. and Kachi-Akkouche, D. (2006). Motion detection using wavelet analysis and hierarchical markov models. In *Spatial Coherence for Visual Motion Analysis*, volume 3667 of *LNCS*, pages 64–75. Springer.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38.

Ferreira, D. R., Szimanski, F., and Ralha, C. G. (2012). A hierarchical Markov model to understand the behaviour of agents in business processes. In *8th International Workshop on Business Process Intelligence*, Tallinn, Estonia.

Greco, G., Guzzo, A., and Pontieri, L. (2005). Mining hierarchies of models: From abstract views to concrete specifications. In *3rd International Conference on Business Process Management*, volume 3649 of *LNCS*, pages 32–47. Springer.

Günther, C. W., Rozinat, A., and van der Aalst, W. M. P. (2010). Activity mining by global trace segmentation. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 128–139. Springer.

Günther, C. W. and van der Aalst, W. M. P. (2007). Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In *5th International Conference on Business Process Management*, volume 4714 of *LNCS*, pages 328–343. Springer.

Hoffmann, A. O. I., Delre, S. A., von Eije, J. H., and Jager, W. (2006). Artificial multi-agent stock markets: Simple strategies, complex outcomes. In Bruun, C., editor, *Advances in Artificial Economics*, volume 584 of *Lecture Notes in Economics and Mathematical Systems*. Springer.

Karande, S., Khayam, S. A., Krappel, M., and Radha, H. (2003). Analysis and modeling of errors at the 802.11b link layer. In *Proceedings of the 2003 International Conference on Multimedia and Expo - Volume 2*, pages 673–676. IEEE Computer Society.

Khayam, S. A. and Radha, H. (2003). Markov-based modeling of wireless local area networks. In *Proceedings of the 6th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 100–107. ACM.

Liao, L., Patterson, D. J., Fox, D., and Kautz, H. (2007). Learning and inferring transportation routines. *Artificial Intelligence*, 171(5□6):311–331.

Mans, R. S., Schonenberg, M. H., Song, M., van der Aalst, W. M. P., and Bakker, P. J. M. (2008). Process mining in healthcare: A case study. In *Proceedings of the International Conference on Health Informatics (HEALTHINF'08)*, pages 118–125. INSTICC.

McLachlan, G. J. and Krishnan, T. (2008). *The EM Algorithm and Extensions*. Wiley Series in Probability and Statistics. Wiley-Interscience.

Neri, F. (2012). A comparative study of a financial agent based simulator across learning scenarios. In Cao, L., Bazzan, A., Symeonidis, A., Gorodetsky, V., Weiss, G., and Yu, P., editors, *Agents and Data Mining Interaction*, volume 7103 of *LNCS*, pages 86–97. Springer.

Nicolae, O., Wagner, G., and Werner, J. (2010). Towards an executable semantics for activities using discrete event simulation. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 369–380. Springer.

Provost, J.-N., Collet, C., Rostaing, P., P□rez, P., and Bouthemy, P. (2004). Hierarchical markovian segmentation of multispectral images for the reconstruction of water depth maps. *Computer Vision and Image Understanding*, 93(2):155–174.

Railsback, S. F., Lytinen, S. L., and Jackson, S. K. (2006). Agent-based simulation platforms: Review and development recommendations. *Simulation*, 82(9):609–623.

Song, M. and van der Aalst, W. M. (2008). Towards comprehensive support for organizational mining. *Decision Support Systems*, 46(1):300–317.

Tao, T., Lu, J., and Chuang, J. (2001). Hierarchical markov model for burst error analysis in wireless communications. In *IEEE 53rd Vehicular Technology Conference*, volume 4, pages 2843–2847.

van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer.

van der Aalst, W. M. P. and Günther, C. W. (2007). Finding structure in unstructured processes: The case for process mining. In *Proceedings the 7th International Conference on Applications of Concurrency to System Design (ACSD 2007)*, pages 3–12. IEEE Computer Society Press.

van der Aalst, W. M. P., Reijers, H. A., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593.

van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B., Kindler, E., and Günther, C. (2010). Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9:87–111.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51.

van der Aalst, W. M. P., Weijters, A. J. M. M., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:1128–1142.

van Dongen, B., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *LNCS*, pages 444–454. Springer.

van Dongen, B. F. and van der Aalst, W. M. P. (2005). A meta model for process mining data. In *Proceedings of the CAiSE'05 Workshops (EMOI-INTEROP Workshop)*, volume 2, pages 309–320.

Veiga, G. M. and Ferreira, D. R. (2010). Understanding spaghetti models with sequence clustering for ProM. In *BPM 2009 International Workshops*, volume 43 of *LNBIP*, pages 92–103. Springer.

Wagner, G. (2004). AOR modelling and simulation: Towards a general architecture for agent-based discrete event simulation. In *5th International Bi-Conference Workshop on Agent-Oriented Information Systems*, volume 3030 of *LNCS*, pages 174–188. Springer.

Wagner, G., Nicolae, O., and Werner, J. (2009). Extending discrete event simulation by adding an activity concept for business process modeling and simulation. In *Proceedings of the 2009 Winter Simulation Conference*, pages 2951–2962.

Weijters, A. J. M. M., van der Aalst, W. M. P., and de Medeiros, A. K. A. (2006). Process mining with the HeuristicsMiner algorithm. BETA Working Paper Series WP 166, Eindhoven University of Technology.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152.

Wu, J. and Aberer, K. (2005). Using a layered markov model for distributed web ranking computation. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, pages 533–542. IEEE Computer Society.

Yang, H. and Alouini, M.-S. (2002). A hierarchical markov model for wireless shadowed fading channels. In *IEEE 55th Vehicular Technology Conference*, volume 2, pages 640–644.

Youngblood, G. M. and Cook, D. J. (2007). Data mining for hierarchical model creation. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 37(4):561–572.

Zhao, N., Chen, S.-C., and Shyu, M.-L. (2006). Video database modeling and temporal pattern retrieval using hierarchical markov model mediator. In *Proceedings of the 22nd International Conference on Data Engineering Workshops*. IEEE Computer Society.