

Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models

Joachim Herbst
DaimlerChrysler AG
P.O. Box 2360, 89013 Ulm, Germany
joachim.j.herbst@daimlerchrysler.com

Dimitris Karagiannis
University of Vienna
Brünnerstr. 72, 1210 Vienna, Austria
dk@dke.univie.ac.at

Correspondence Address:

Joachim Herbst
DaimlerChrysler AG
P.O. Box 23 60
89013 Ulm
Germany
Tel.: 0049 731 505 2832
Fax.: 0049 731 505 4218
e-mail: joachim.j.herbst@daimlerchrysler.com

Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models¹

Abstract

Current workflow management systems (WFMS) offer little aid for the acquisition of workflow models and their adaptation to changing requirements. To support these activities we propose to apply techniques from machine learning, which enable an inductive approach to workflow acquisition and adaptation. We present a machine learning component that combines two different machine learning algorithms: the first one induces the structure of sequential workflows and the second one is responsible for the induction of transition conditions. The second task can be solved by applying standard decision rule induction algorithms. In this contribution we focus mainly on the algorithms for the first task. For this purpose we describe two algorithms based on the induction of hidden markov models. The first algorithm is a bottom-up, specific-to-general algorithm and the other one applies a top-down, general-to-specific strategy. Both algorithms have been implemented in a research prototype. In six scenarios we evaluate and compare the two algorithms experimentally. The induced workflow models can be imported by the business process management system ADONIS².

Keywords: Workflow management systems, artificial intelligence, machine learning

Introduction

Workflow Management Systems

Efficient business processes are an important success factor in today's competitive markets. Information technology is playing a key role as an enabling technology in achieving this efficiency. Business process management systems (see e.g. Karagiannis et al. (1996)) are increasingly used to model, analyze, simulate, enact and manage business processes. One subclass of business process management systems are workflow management systems (WFMS). WFMS concentrate on supporting enactment and management of business processes. For workflow enactment a formal model of the business processes is interpreted by a workflow engine. The WFMS interacts with the participants of the business process by informing them about the tasks that need to be done and by providing them with the relevant documents or information. Most WFMS also include interfaces for monitoring the state of workflow instances, which are representations of concrete business cases, and they offer audit components, that trace all state changes of the executed workflow instances. WFMS usually provide graphical modeling editors. Many different formalisms have been proposed for workflow modeling. Within this contribution we are using the ADONIS modeling language BOC (1999). According to the ADONIS modeling language a workflow model is a directed graph. There are seven different node types representing activities, subprocesses as well as five different control flow constructs. A short explanation of these constructs and their graphical representation is given in figure 1. The directed edges of the workflow model can be labeled either with simple boolean conditions over a set of workflow variables or with probabilities. This description of the ADONIS modeling language concentrates on the behavioral perspective (see Curtis et al. (1992)) on a workflow model, as our learning approach considers only this aspect. A complete description should for example also consider which participants are responsible for performing an activity (organizational perspective) and which data or documents (informational perspective) are required. Additionally the syntactical (e.g. "Each workflow model must have

one unique starting node.”) and semantical aspects (e.g. “Which workflow instances can be generated by the workflow model?”) of the modeling language should be explained. We refer to BOC (1999) for these aspects. For this contribution the above definition should be sufficient. An example for a workflow model is given in figure 2.








| Graphical Representation | Node Type | Explanation |
|---|------------|--|
|  | START | Starting node of a workflow model. |
|  | SUBPROCESS | A subprocess node is used for hierarchical decomposition of workflows. |
|  | ACTIVITY | An activity node. |
|  | SPLIT | An m of n split. (m of n successors may be activated) |
|  | JOIN | Join nodes synchronize the concurrent paths of their corresponding split nodes |
|  | DECISION | A decision node. (Exactly 1 of n successors may be activated) |
|  | END | End node of a workflow model. |

Figure 1: ADONIS node types

Acquisition of Workflow Models

In many state of the art workflow projects the following approach is taken. In a first step the business process as it is performed is acquired and modeled. This initial modeling is done at a rather high level of abstraction, which Karagiannis et al. (1996) call a business graph. After an optimization step the resulting optimized process model is implemented as a workflow. The implementation involves further refinement of

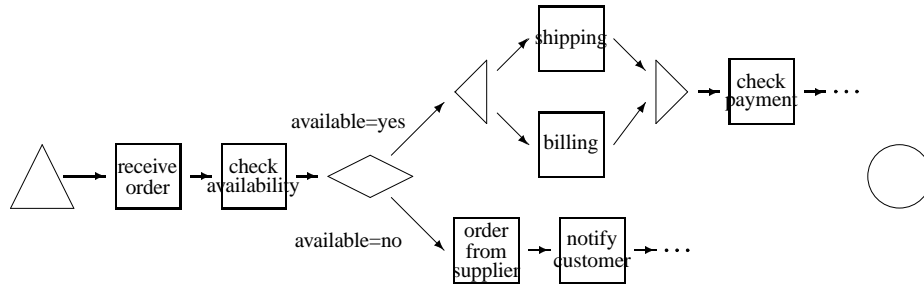


Figure 2: Part of a simple ADONIS workflow model

the process model and thus further acquisition of workflow knowledge. The refined process model is called a workflow graph in Karagiannis et al. (1996).

The acquisition of workflow knowledge is typically done by interviews or questionnaires. The people responsible for process acquisition and optimization we will call process engineers in the following. One of the most time consuming activities for the process engineers within the described approach is the acquisition of the workflow knowledge (compare e.g. Herbst and Bumiller (1997), Wargitsch (1998), Leymann and Roller (1999)). In several workflow-projects we have made the experience, that the relative times spent for the different tasks are distributed approximately as shown in table 1.

The reason, why workflow acquisition is so time consuming, is that the required workflow knowledge is usually highly distributed within the organization. It is stored in the heads of the human participants, who

| typical tasks within a workflow project | relative time spent |
|---|---------------------|
| acquisition of workflow knowledge | 60% |
| optimization | 20% |
| implementation as workflow | 20% |

Table 1: Relative times spent within selected workflow projects at DaimlerChrysler

are actively involved in the execution of the business process. Participants have in-depth knowledge about the activities (functional perspective) they are involved in. They know the required data, documents and applications (informational perspective) and they know who is responsible (organizational perspective). It is thus not too difficult to obtain a detailed picture of each activity within a workflow model using traditional means of knowledge acquisition. On the other hand participants often have little knowledge about the control flow (behavioral perspective) of the process they participate in (compare e.g. Leymann and Roller (1999)). Furthermore the participants usually belong to different organizational units, they use different words to describe the same semantic entities and they often have conflicting interests. This makes control flow acquisition an extremely difficult task.

As the control flow of a workflow model can be compared to a program, its validation by the participants - who are generally not programmers - is also difficult. Although visual representations and simulations of the workflow are employed, many deficiencies of the workflow model are discovered after the WFMS has been taken into operation. These deficiencies include e.g. modeling errors as well as missing requirements. Modeling errors are easily overseen during the validation of a workflow model if the participants do not have a deep understanding of the semantics of the modeling language. These people often develop their own understanding and are disappointed when the WFMS behaves different than they have expected (compare Abbott and Sarin (1994)). On the other hand additional requirements are often discovered once the WFMS is productive. As workflow environments are open systems (compare Hewitt (1986)) there may be

situations, that have not been anticipated by the model. Some of these situations could be rare or unimportant exceptions, that should be handled by the adaptation mechanisms of the WFMS (see below), but for other more frequent or important situations it might make sense to handle them within the workflow model. Thus trying to meet all requirements of the human participants, who are expected to work with the workflow every day, leads to a highly iterative and time consuming workflow acquisition and validation. Therefore we decided to investigate the use of machine learning techniques, that allow a semi-automatic control flow acquisition.

Adaptation of Workflow Models

Many of the current WFMS products enforce the execution of a business process exactly according to the defined workflow model. Although this may be desirable for some domains, experience has shown, that many other domains require a much higher degree of flexibility to reflect the human aspects of cooperative work. It is commonly accepted (see e.g. Ellis et al. (1995), Reichert and Dadam (1997) or Pareschi et al. (1996)) that it is not possible to anticipate all situations that may occur throughout the execution of a workflow instance in advance at the time the workflow model is defined. In exceptional situations it is necessary, that the workflow participants are able to take partial control over the workflow and deviate from the modeled control flow. WFMS offering these possibilities are often called adaptive WFMS. Examples for such adaptive WFMS are commercial products such as e.g. InConcert (Sarin (1996)) and Teamware Flow (Teamware (1997)) and research prototypes such as ADEPT_{flex} (Reichert and Dadam (1997)) and DYNAMITE (Heimann et al. (1996)), addressing more advanced features of adaptive WFMS. There is one serious drawback of these approaches. Changes at the workflow instance level usually have no effects to the model level. If over a longer period of time a certain kind of deviation from the workflow model becomes common practice rather than rare exception it seems reasonable, to adapt also the workflow model. In a way workflow management has to do with predicting the tasks that need to be done. If the probability of

a wrong prediction, requiring correction by the participants, is too high, the WFMS will be an impediment to the participants rather than a support. In our opinion an adaptive WFMS should supply mechanisms supporting the improvement of the workflow models based on feedback from the workflow instance level. We are convinced that such feedback mechanisms could be based on machine learning techniques.

Integrating WFMS and Machine Learning

In the following we present a machine learning component, which we propose to integrate into a WFMS. The machine learning component supports acquisition and adaptation of workflow models. It generalizes a number of workflow instances described in a workflow trace to a single workflow model that is able to generate all observed instances. An extract of the contents of a workflow trace that could be generated from the workflow model of figure 2 is shown in table 2.

| Time and Date | Event | Workflow | Workflow Instance | Activity | Activity Instance |
|---------------|----------|-----------|-------------------|--------------------|----------------------|
| 8:12 11/25 | StartWF | sellGoods | sellGoods.34 | | |
| 8:14 11/25 | BeginAct | sellGoods | sellGoods.34 | receive order | receive order.6 |
| 8:21 11/25 | EndAct | sellGoods | sellGoods.34 | receive order | receive order.6 |
| 8:32 11/25 | BeginAct | sellGoods | sellGoods.34 | check availability | check availability.7 |
| 8:32 11/25 | StartWF | sellGoods | sellGoods.35 | | |
| 8:34 11/25 | BeginAct | sellGoods | sellGoods.35 | receive order | receive order.8 |
| 8:34 11/25 | EndAct | sellGoods | sellGoods.34 | check availability | check availability.7 |
| 8:50 11/25 | BeginAct | sellGoods | sellGoods.34 | shipping | shipping.9 |
| 8:51 11/25 | EndAct | sellGoods | sellGoods.35 | receive order | receive order.8 |
| 8:55 11/25 | BeginAct | sellGoods | sellGoods.34 | billing | billing.10 |
| 9:03 11/25 | EndAct | sellGoods | sellGoods.34 | billing | billing.10 |
| 9:10 11/25 | EndAct | sellGoods | sellGoods.34 | shipping | shipping.9 |
| 7:55 12/12 | BeginAct | sellGoods | sellGoods.34 | check payment | check payment.11 |
| 8:00 12/12 | EndAct | sellGoods | sellGoods.34 | check payment | check payment.11 |

Table 2: A workflow trace

From such a workflow trace, individual workflow instances can be extracted. Workflow instances can be described as a directed acyclic graph. The nodes represent the activities that were executed. The edges of the directed graph describe a partial order, indicating the temporal order of execution. When talking

about workflow instances that are generated by sequential workflow models the edges of the workflow instance graph define a linear order. A workflow trace often contains more information related to a workflow instance. These are e.g. the participant who performed an activity, the documents used, or the values of the workflow variables. But for our needs within this paper, the reduced definition of a workflow instance is sufficient. Figure 3 shows the workflow instance “sellGoods.34”, that could be extracted from the workflow trace of table 2.

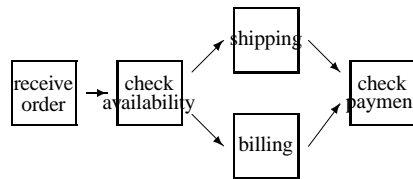


Figure 3: A workflow instance

There are some adaptive WFMS that realize deviations at runtime by allowing the users to change a workflow model that is only valid for the instance under consideration. For these WFMS it would be possible to use these instance specific workflow models as input for the induction, as these may be enriched with more information than workflow instances (like e.g. a transition condition specifying when a certain change is valid). We decided to keep the requirements of the machine learning component as low as possible, to make our approach more widely applicable. Relying on workflow instances as input also simplifies the design of user interfaces, which are critical for the acceptance of the adaptive WFMS by the participants (compare e.g. Abbott and Sarin (1994)). Such an interface could for example rely on the metaphor of workflow folders, that are passed from participant to participant.

The integration of the machine learning component into a WFMS enables a new inductive approach to the creation of a workflow application. We believe that this will significantly reduce the time needed to setup

a workflow application and at the same time it should lead to higher quality models, as requirements are collected, while the system is productive. By helping to transform tacit knowledge into explicit knowledge this approach contributes to supporting organizational learning (see Agyris and Schoen (1978)). It allows to handle exceptions at the workflow instance level and yet it is able to trace these manually enacted workflows and to adapt the corresponding workflow models.

Inductive Workflow Acquisition and Adaptation

We propose to apply an inductive approach to workflow management. This approach is divided into three separate phases, which are arranged in a cycle. These three phases and their order are shown in figure 4.

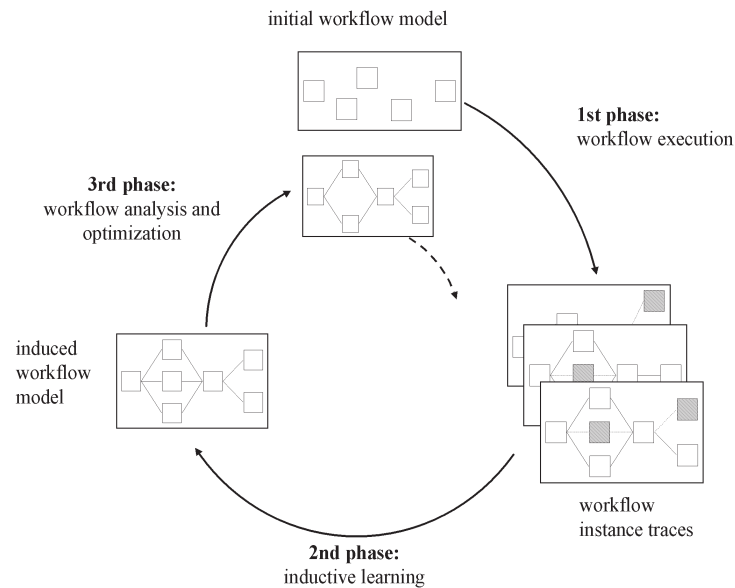


Figure 4: Inductive workflow acquisition and adaptation

Instead of waiting until a complete and correct workflow model is available, we propose to model only the elementary activities as the basic building blocks of an initial workflow model. An adaptive WFMS

allowing two modes of operation, a passive and an active mode, is then immediately taken into operation. In the first phase - the workflow execution phase - this adaptive WFMS is started in passive mode, which means that the workflow engine is switched off and the routing of tasks is done manually. It allows users to start and complete process instances, create and execute tasks, route tasks to other participants manually. At this point the WFMS only serves as a provider for traces of executed workflow instances, stored in trace files.

In the second phase - the inductive learning phase - the extracted workflow instances are interpreted by the machine learning component, which generalizes observed instances to a workflow model. Because the machine learning component has no knowledge about the goals of activities, processes and the organization, it can not reason about, what is a good and what is a bad workflow model. The induced workflow model is thus a description of how the work was actually performed and not how the work should be done regarding the goals of the organization. As a common practice is not necessarily a best practice, the induced workflow model should be analyzed and possibly optimized by experts.

Therefore we have included a third phase - the workflow analysis and optimization phase. In this phase the induced workflow model may be used as a basis for a decision whether to enact the business process with the workflow engine. If the induced workflow model contains little structure, automatic enactment makes no sense. In this case it might be better to keep the manual enactment. Workflow optimization is performed as in the standard approach described above. As requirements are collected while the WFMS is productive and providing that enough workflow instances have been collected, we expect the quality of the induced workflow model to be higher than that of a workflow model acquired offline, using only traditional means of workflow acquisition such as interviews and questionnaires. Additionally the induced workflow model may be enriched with quantitative parameters such as interarrival or execution times as well as transition probabilities. This is an invaluable input for simulation improving the validity of the simulation model (compare e.g. Herbst et al. (1997)).

If the decision is taken, to enact the business process using the workflow engine, we are back in the first phase. The WFMS is activated in the active mode, to enact the business process according to the optimized workflow model. In exceptional situations the actors may switch back to the passive mode thereby taking control over the workflow as in the acquisition phase. All activities are traced and passed to the machine learning component, which induces a workflow model for the second iteration. This induced workflow model serves as input for the next optimization cycle. It shows the process engineers how well the designed workflow model was able to guide the execution of real business cases. In this way the participants are constantly involved in the design of the workflow model and continuous process improvement is supported.

Induction of Workflow Models

To provide the means enabling the described inductive approach to workflow management, we need to solve the following induction task:

- given a multiset of workflow instances belonging to the same workflow type
- find a good approximation M of the workflow model M_0 , that determined the generation of the observed workflow instances

Of course the workflow model M_0 need not exist. Within the above definition it is simply a modeling hypothesis. M_0 could e.g. be a semi-formal model of the business process described in a process handbook or it may just exist in the heads of the participants involved in the workflow. We have decomposed this induction task into two subtasks:

- Induction of structure - within this subtask the nodes, the edges, and the transition probabilities of the workflow model M are induced.

- Induction of conditions - where possible, local conditions for transitions following a split or a decision node are induced.

In the following sections we will describe algorithms for inducing the structure of sequential workflow models in detail and give a short outline on the induction of conditions.

Inducing the Structure of Sequential Workflow Models

Within this contribution we restrict ourselves to the induction of the structure of sequential workflow models (models without split and join constructs). Although this is clearly a limitation, because many real world applications require concurrent workflow models, it provided us with a well established theoretical basis for our work and allowed us to evaluate the basic idea of our approach before putting effort in designing advanced induction algorithms, that are able to deal with concurrency, which is one of the main goals of our current work (see Herbst and Karagiannis (1999)).

The Multiple Node Problem

Inducing the structure of a sequential workflow becomes trivial if one assumes, that for each activity there exists one unique activity node within the workflow model. In this case one could create a unique activity node for each distinct activity name observed in the samples. Whenever an activity a_i precedes an activity a_j a transition from a_i to a_j is added to the model. The transition probabilities can be estimated using empirical counts. Finally decision nodes are added whenever a certain activity has more than one possible successor.

To assure that this approach is applicable, one could of course distinguish different occurrences of a certain activity (e.g. “receive order”) within a workflow model by unique names (e.g. “receive order-1” and “receive order-2”). But being able to find unique names for each occurrence of an activity requires that one knows about how many occurrences there are within the workflow model. This would mean, that to

some extent the structure of the workflow needs to be known in advance. Since we would like to provide mechanisms helping to find the structure of a workflow, this assumption is not realistic. So we need to assume that within one workflow model there may be multiple activity nodes assigned to the same activity.

Reduction to Regular Grammar Inference

The structure of sequential workflow models can be - independently of the modeling language used - represented by finite state automata (FSA) and sequential workflow instances can be represented by strings over a finite alphabet. Each symbol in this alphabet corresponds to an activity of the workflow instance. Thus the problem of sequential workflow structure induction can be reduced to the problem of inducing FSAs from a positive sample of strings which is a regular grammar inference problem. Regular grammar inference has already been addressed in the grammatical inference community (see e.g. Parekh and Honavar (1999) for an overview). It has been shown by Gold (see Gold (1967)) that the regular grammar inference problem is hard in the sense that regular languages cannot be correctly identified from positive examples alone. The task of identifying the minimum state deterministic finite automaton, that is consistent with an arbitrary set of positive and negative examples has been shown to be NP-complete Gold (1978). Despite these negative theoretical results there exist a number of heuristic solutions to different regular grammar inference problems, which have been successfully applied in practical applications such as speech recognition or the discovery of patterns in biosequences.

Examples for such heuristic solutions are ALERGIA Carrasco and Oncina (1994) and Bayesian Model Merging Stolcke and Omohundro (1994). Both operate on a positive sample of strings. We have selected Bayesian Model Merging, which is based on hidden markov models (HMMs) and applies a specific-to-general approach, as a basis for our work. After a short introduction into HMMs, we present two different workflow induction algorithms. The first one is based on Bayesian Model Merging, with some minor modifications, and the second one applies a general-to-specific approach

Hidden Markov Models

A HMM (for a good introduction see Rabiner (1989)) can be described basically as a finite state automaton whose transitions have transition probabilities and whose states are associated with a finite set of output symbols having a certain output probability. Formally a HMM can be defined as a tuple $\lambda = (Q, V, A, B, \pi)$ where $Q = \{q_1, q_2, \dots, q_N\}$ is a finite set of states, $V = \{v_1, v_2, \dots, v_M\}$ is a finite set of output symbols, $A = (\alpha_{ij}) = P[S_{t+1} = q_j | S_t = q_i]$ with $1 \leq i, j \leq N$ are transition probabilities, $B = (\beta_i(k)) = P[\text{output of } v_k \text{ at time } t | S_t = q_i]$ with $1 \leq i \leq N$ and $1 \leq k \leq M$ are output probabilities $\pi = \{\pi_i\}$, $\pi_i = P[S_1 = q_i]$ with $1 \leq i \leq N$ are the start probabilities and S_t denotes the state λ is in at time t . A HMM can be visualized as a directed graph, whose nodes describe states or output symbols and whose edges represent state or output symbol transitions. The behavioral aspects of sequential workflow models may be mapped to a HMM, whose output symbols represent activities and whose state transitions describe the control flow. For this purpose we need slightly less representative power than the HMM offers. This is because in a workflow model the relationship between a node and an activity is deterministic. So in the corresponding HMM each state only has one output symbol with an output probability of 1. But to take into account what we called the multiple node problem the same activity may occur more than once within a workflow model. This means that the same output symbol may be produced by several states of the corresponding HMM and one is thus not able to uniquely identify the state that generated this observed output symbol. Therefore the hidden nature of the HMM remains. Some of the other approaches to workflow induction like Agrawal et al. (1998) or Bocionek and Mitchell (1993) ignore this problem.

The problem we want to solve, is to generate a workflow model from workflow traces. This problem corresponds to the problem of finding an appropriate HMM from a number of observed output symbol sequences. For this problem there is a well known algorithm called the Baum-Welch expectation maximization (see for example Rabiner (1989)) also known as the forward-backward algorithm. Given one sequence of output symbols $e = e_1 e_2 \dots e_l$ with $e_i \in V$ and an initial HMM $\lambda_0 = (Q, V, A_0, B_0, \pi_0)$ the

Baum-Welch algorithm iteratively re-estimates the parameters A_i, B_i and π_i until some limiting point is reached. This basic algorithm can be easily extended to support multiple sequences of observations. The formulas for the re-estimation of the parameters are given for example in Rabiner (1989). It was proven that this procedure converges to a local maximum of the likelihood function $P[e|\lambda]$. But Baum-Welch is not very well suited for the problem of finding an appropriate workflow model because it only adapts parameters of a given HMM, with a given number of states. In our case the structure is exactly what we would like to discover.

Model Merging

Stolcke and Omohundro (1994) describe an algorithm which induces the structure as well as the transition and output probabilities of a HMM from observations. To achieve this, the algorithm searches the space of HMMs, beginning with a most specific HMM, which directly encodes the observations. The most specific HMM is iteratively generalized by merging states. The algorithm uses a heuristic based on the bayesian posterior probability $P[\lambda | E]$ to determine which states to merge and when to stop merging states. The input to this algorithm $E = \{e_{11}e_{12} \dots e_{1l_1}, \dots, e_{n1}e_{n2} \dots e_{nl_n}\}$ is a multiset of sequences of output symbols.

The overall algorithm can be characterized as a heuristic hill climbing search over the space of HMM's, starting with a most specific HMM, which is consistent with the observations. The most specific HMM originally proposed by Stolcke and Omohundro (1994) contains exactly one separate path for each observed sequence of symbols. To decrease the size of the search space, we are employing a most specific model with less states, called a prefix HMM (compare Carrasco and Oncina (1994)). This prefix HMM is obtained from the most specific model proposed by Stolcke and Omohundro (1994), by mapping all states sharing a common prefix to one single state. Thus a prefix HMM has the property, that every common prefix of any two observation sequences is generated by the same unique generating path of the prefix HMM. Assume we

observe a set of observations $E = \{abcac, abcc, abcabcc, abcabca\}$ generated from the unknown HMM shown in figure 5. The prefix HMM for E is the HMM M'_1 shown on the top left of figure 7. We have omitted the transition probabilities for those transitions having a probability of 1.

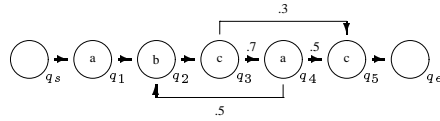


Figure 5: The unknown HMM that generated E

To generalize a HMM, states are iteratively merged using a merge operator, as shown in figure 6. For our purpose we have modified the merge operator originally defined in Stolcke and Omohundro (1994) to allow only the merging of states having the same output symbol. Merging two states q_i and q_j with the same output symbol has the following effects on the HMM:

- q_i and q_j are removed and a new state q'_i is inserted.
- all transitions to and from q_i and q_j are redirected to q'_i .
- the transition probabilities are adjusted to their maximum likelihood estimates. This is calculated as

$$\alpha_{ij} = \frac{c_{ij}}{\sum_{k=1}^N c_{ik}}$$

where c_{ij} are the number of transitions from q_i to q_j needed when generating E .

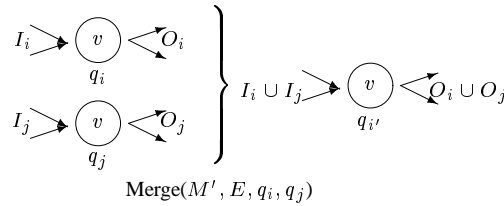


Figure 6: The merge operator

In the last step of the merge operator the transition probabilities are adjusted. This requires that all observations in E are parsed. Using two approximations this step can be simplified. One common approximation is to consider only a most likely path - called a viterbi path - rather than all possible paths that may generate a certain sequence. The second approximation deals with the way these viterbi paths are determined. Although there is an efficient algorithm - known as the viterbi algorithm - for the determination of a viterbi path to a given sequence, applying this algorithm requires that all observations in E are parsed. This step can be approximated by determining viterbi paths only for the most specific model and assuming that a merge operator does not change the viterbi paths. This allows us to maintain frequency counters at each transition. These counters are added whenever two transitions are merged. This approximation seems to work well in practice as shown in Stolcke and Omohundro (1994) and it allows an incremental induction because the approximated merge operator is independent of E .

For our implementation we decided to use a different heuristic than the original bayesian model merging. Stolcke and Omohundru report, that the choice of appropriate priors and parameters, does not greatly affect the course of the search except for deciding when to stop. Their evaluation shows that the likelihood $P[E | \lambda]$ was the determining factor in guiding the search. This was the reason for us to employ a pure log-likelihood heuristic for the search. This means that the parameters for a merge operation are selected in such a way, that the decrease of the log-likelihood is minimized. Merge operations maintaining the log-likelihood of a model are thus preferred over merge operations that lower the log-likelihood. The log-likelihood per sample is calculated as $h(E, \lambda) = \ln(\prod_{i=0}^{N-1} \prod_{j=1}^N \alpha_{ij}^{c_{ij}})$. As a stopping criterion for the hill-climbing search we have defined a parameter `genFactor`, that determines the generalization factor, which is the ratio between the log-likelihood of the model under consideration and the most specific model. As soon as this ratio is larger than `genFactor`, the search terminates returning the current model.

To improve the performance of the merging approach, we allow incremental learning. The number of examples, that are gathered before merging starts is determined by the `mergeStart` parameter. This

parameter allows us to move anywhere in the spectrum from a pure incremental to a pure non-incremental learning algorithm. Setting `mergeStart=1` has the effect that each observed example is immediately merged into the model. A pure non-incremental merging is realized by setting `mergeStart` to the total number of examples observed. The degree of generalization in the first merging phase, which is started when a sufficient number of examples has been observed, is determined by a second generalization factor. This is given by the `preGenFactor` parameter. So the output of a first merging phase with a small generalization factor (`preGenFactor`) can be merged as soon as all examples have been observed in a second phase with a higher generalization factor (`genFactor`).

The Model Merging Algorithm Let λ_0 be the empty HMM, let $i := 0$, let $E := \emptyset$

1. read a set E_i of $|E_i| = \text{mergeStart}$ observations and insert them into λ_i (considering common prefixes) and let $E := E \cup E_i$
2. let $\lambda_{i+1} := \text{merge}(\lambda_i, E, \text{preGenFactor})$, let $i := i + 1$
3. if there are unread observations repeat from step 1
4. let $\lambda_{i+1} := \text{merge}(\lambda_i, E, \text{genFactor})$
5. return λ_{i+1} as the induced HMM

Procedure `merge($\lambda, E, \text{generalizationFactor}$)`

1. determine the log-likelihood $h(E, \lambda)$, let $j := 0$ and let $\lambda_j := \lambda$
2. determine the set of possible merges K for λ_j
3. for each merge $k \in K$ determine the merged model $k(\lambda_j)$ and its log-likelihood $h(E, k(\lambda_j))$
4. let k^* be a merge that maximizes $h(E, k(\lambda_j))$ and let $\lambda_{j+1} := k^*(\lambda_j)$
5. if $\frac{h(E, \lambda_{j+1})}{h(E, \lambda)} \leq \text{generalizationFactor}$ then let $j := j + 1$ and repeat from step 2
6. return λ_j

Example Figure 7 shows the different HMMs on the search path when applying our implementation of model merging (with the parameters `mergeStart=20`, `preGenFactor=0.5` and `genFactor=1.2`) to E . Figure 8 shows the decrease of the log-likelihood for these HMMs. The merge operations are selected in such a way, that the decrease of the log-likelihood is minimized. The returned HMM M'_7 identifies the structure of the unknown HMM correctly.

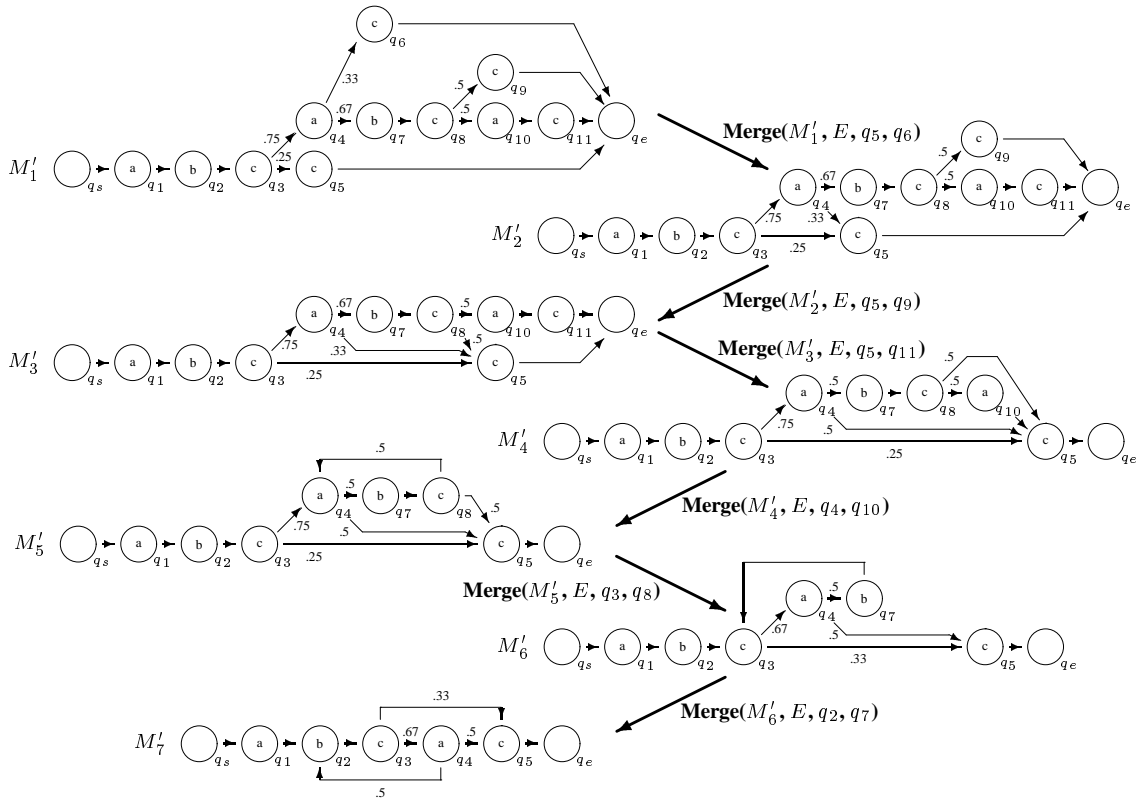


Figure 7: Model merging applied to E

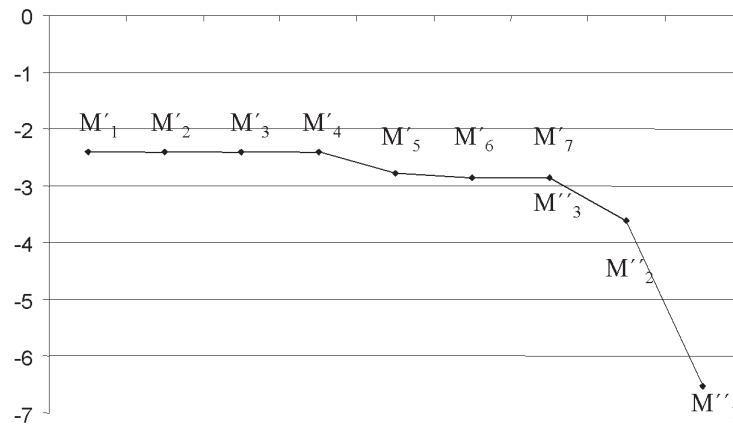


Figure 8: Log-Likelihood of the models

Model Splitting

When applying the merging approach to repetitive business processes we encountered a problem resulting from the direction of the search strategy. In the merging approach one starts with the most specific model, which has a high number of states being associated with the same symbol. In the workflow models we have acquired manually for workflow projects over the last few years we have observed at most three or four nodes associated with the same activity within one workflow model. In the space of HMM's, the distance from the most specific model to a target model having only a few states associated with the same symbol is usually very large. We measure distance in terms of the number of merge operations one has to apply to transform the most specific model to the target model. We found that due to the hill-climbing search the chance of taking a suboptimal merge decision is pretty high. This especially true when loops are present in the target model. For this reason we decided to try the reverse search strategy, which means that the induction algorithm performs specialization steps starting with the most general HMM. In our sense - considering the restriction that each state is associated with exactly one output symbol - the most general HMM is the HMM that

- contains exactly one state for each observed output symbol and
- incorporates exactly the transitions observed in the examples.

The transition probabilities are assigned to their maximum likelihood estimates as in the merging approach. Consider again the examples $E = \{abcac, abcc, abcabcc, abcabcac\}$. The corresponding most general HMM covering these samples is the HMM M_1'' shown at the top of figure 10.

As it can be easily verified for the given examples the most general HMM M_1'' covers all observations. But in contrast to the most specific HMM it also covers many unseen sequences. Examples for sequences which are covered but not contained in the set of observations are $acacacac$, $abcccccc$ and abc . In order to receive a more appropriate HMM we must specialize. Specialization is done using a split operator, that splits one state into two states with different incoming transitions as shown in figure 9. This introduces two states producing the same output symbol in two different contexts. Splitting may be applied to any state q_i with more than one incoming transition. The split operator requires a set of parameters. Let I be the set of states $q_j, j \neq i$ with outgoing transitions to q_i and let $\{I_1, I_2\}$ be a partition of I . Let O be the set of states $q_j, j \neq i$ which have an incoming transition from q_i . Let $\sigma \in \{1, 2, 3, 4\}$. $\text{Split}(\lambda, E, q_i, I_1, I_2, \sigma)$ has the following effects on λ

- state q_i and all its incoming and outgoing transitions are removed and new states q_i' and q_i'' are inserted
- q_i' obtains incoming transitions from all states in I_1 and q_i'' receives incoming transitions from all states in I_2
- both q_i' and q_i'' receive outgoing edges to all states in O
- if q_i originally had a transition to itself then
 - if $\sigma = 1$ then both q_i' and q_i'' receive transitions to themselves
 - if $\sigma = 2$ then q_i' receives a transition to itself and q_i'' obtains a transition to q_i'

- if $\sigma = 3$ then q_i'' receives a transition to itself and q_i' obtains a transition to q_i''
- if $\sigma = 4$ then both q_i' and q_i'' receive a transition to the other
- the transition probabilities are adjusted to their maximum likelihood estimates
- transitions having a probability of zero are removed

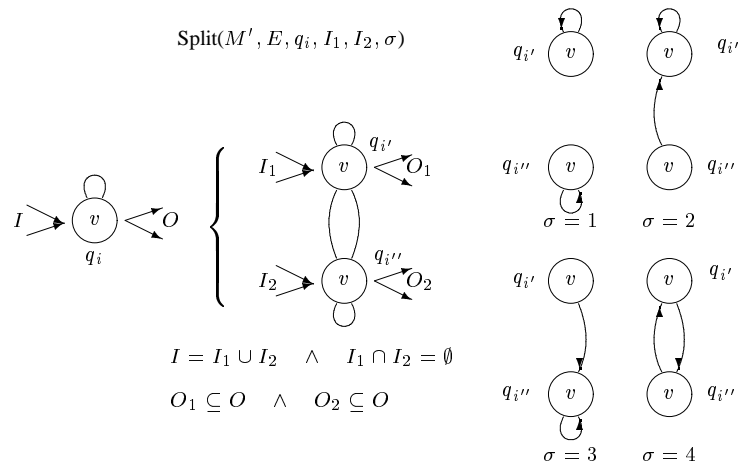


Figure 9: The split operator

The specialization actually occurs in the last step when unnecessary transitions are removed. A HMM resulting from a split operation has some important properties. If the HMM covers all the observations before the split it covers all observations after the split. Depending on the choice of the parameters the HMM is specialized, which means that some sequences (not in the set of observations) it covered before the split are no longer covered. If the path that generates an arbitrary output sequence is unique before the split there will be a unique path generating the same output sequence after the split. This last property is important for the computation of the likelihood. Instead of the pure log-likelihood based search presented in (Herbst and Karagiannis (1998)), we have implemented a variant of the splitting algorithm, which uses a minimum improvement parameter `minImprove` as a stopping criterion and the log-likelihood per sample

$\frac{h(E,\lambda)}{|E|}$ as the heuristic to guide the search. The split operator resulting in the highest value of $\frac{h(E,\lambda)}{|E|}$ is chosen for specialization. The parameter `minImprove` defines how much a split operator must improve the log-likelihood per sample in order to be accepted. If in a given situation no split operation may improve the log-likelihood per sample of the current model by at least `minImprove`, the search terminates and the current model is returned as the result. As the log-likelihood of a model depends very much on the number of examples, we are using the log-likelihood per sample as the heuristic. This does not make any difference for the search, but it makes the `minImprove` parameter less sensitive to changes in the number of examples read. The overall algorithm can be outlined as follows.

The Model Splitting Algorithm Let λ_0 be the most general HMM for the set E , let $i = 0$

1. read the complete set of observations E
2. determine the set of possible splits K for λ_i
3. for each split $k \in K$ determine the resulting model $k(\lambda_i)$ and its log-likelihood per sample $\frac{h(E,k(\lambda_i))}{|E|}$
4. let k^* be a split maximizing $\frac{h(E,k^*(\lambda_i))}{|E|}$ and let $\lambda_{i+1} = k^*(\lambda_i)$
5. if $\frac{h(E,\lambda_{i+1})}{|E|} - \frac{h(E,\lambda_i)}{|E|} \geq \text{minImprove}$ then let $i := i + 1$ and repeat from step 2 else terminate and return λ_i as the induced HMM

At a first glance model splitting has some disadvantages compared to the merging approach. First of all the number of possible splits for each state is exponential in the number of incoming edges. Another drawback is the fact that the approximation concerning the detection of viterbi paths explained in the previous section is not applicable here. So incremental learning is not possible. But as our experiences in applying both algorithms to workflow traces show, the splitting approach also has many advantages (see below).

Example The result of applying model splitting (with parameter `minImprove=0.01`) to E is shown in figure 10. The log-likelihood of the HMMs visited during the search is shown in figure 8. Model splitting

stops after just two search steps and returns the HMM M_3'' , which describes the structure of the unknown model correctly. Any further split operations do not increase the log-likelihood significantly.

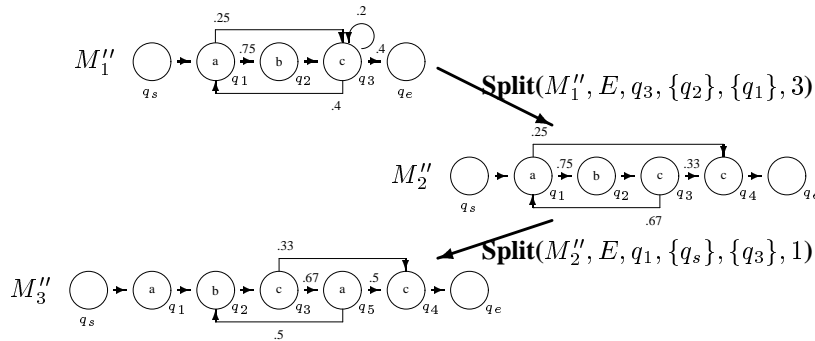


Figure 10: Model splitting applied to E

Induction of Conditions

The first induction phase returns a compact model for the structure of the observed workflow instances. In this structure all non-deterministic transitions are described by probabilities. These probabilistic process models may be used for simulation (compare e.g. Herbst et al. (1997)). In this case the distribution described by the transition probabilities would be modeled by random generators. This of course is no adequate model for a WFMS. For the selection of the next transition most current WFMS allow the modeler to state simple boolean conditions based on decision relevant attributes of the workflow instance. These attributes may be for example the attributes of the documents which are passed between the actors of the activities. Many systems also offer manual decisions. These allow the actors decide which path to follow. This is useful when complex human reasoning processes are necessary for the decision. In case conditions are applicable, methods from machine learning may help finding these conditions. A method which - where possible - induces transition conditions from the observed manually enacted workflows is outlined in the

following.

The learning task for this second induction phase is to find rules, stated in terms of conditions on a given set of attributes, replacing the transition probabilities. This task can be solved efficiently by using standard decision rule induction algorithms such as for example C4.5 (see Quinlan (1993)). These algorithms, which have also been successfully applied in many data mining applications, are given a set of examples, each consisting of the values of a fixed set of attributes and a classification. Consider for example the structure given on the left side of figure 11. Let's assume we are given values for three attributes customer, price and review result and we observe two workflow instances. The first one has a transition from q_1 to q_2 and the values of the attributes are (Jones, 20, ok) and the second has a transition from q_1 to q_3 and the value tuple (Smith, 300, no). This yields two examples for each transition shown in table 3.

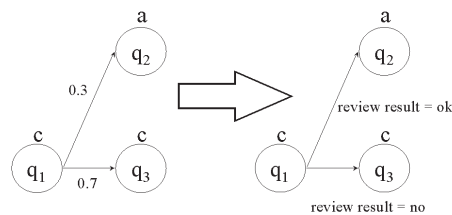


Figure 11: Induction of conditions

| Transition $q_1 \rightarrow q_2$ | Transition $q_1 \rightarrow q_3$ |
|----------------------------------|----------------------------------|
| (Jones, 20, ok, TRUE) | (Jones, 20, ok, FALSE) |
| (Smith, 300, no, FALSE) | (Smith, 300, no, TRUE) |

Table 3: Two sets of examples

Examples like the ones in table 3 can be directly processed by a decision rule induction algorithm to produce conditions as the ones shown on the right side of figure 11. This of course requires a much greater number of examples than shown here and the quality of the induced condition depends on the provided attributes. If these are not relevant for the decision, either no condition or a useless condition with a low

predictive accuracy will be found.

As the values of the decision relevant attributes are subject to change during the observation sequence, it is important to remember the values at those points in time a certain decision is taken. A method for dealing with this problem is outlined in Herbst and Karagiannis (1998).

Implementation and Evaluation

Prototype

To evaluate the described concepts, we have developed a research prototype. Both the model merging and the model splitting algorithms are included in this prototype. The induction of transition conditions has not yet been implemented. The prototype is able to read either a text file containing sample strings as well as workflow traces. For the evaluation we are currently using artificial workflow traces produced by a simulation rather than real world workflow traces produced by a WFMS. This allows us to evaluate our prototype much more easily than applying a WFMS. By using simulation we can generate a wide variety of workflow traces from workflow models of different sizes and structural complexities. The simulation component (see Herbst et al. (1997)) of the business process management system ADONIS (see Karagiannis et al. (1996)) is employed for this purpose. It could easily be replaced by any commercial adaptive WFMS such as InConcert (Sarin (1996)) or Teamware Flow (Teamware (1997)), that is able to collect workflow traces. For the integration into our prototype an interface for transforming the trace to our description language needs to be provided.

The workflow models induced by the machine learning algorithms are transformed to an ADL (ADONIS Definition Language) file, which can be directly imported by ADONIS. A function for the generation of the graphical layout, which is provided by ADONIS, allows us to compare the input workflow model given to the simulation with the workflow model that was induced from the workflow trace. The analysis

and optimization of the induced model can be done using different analysis, evaluation or simulation functionalities of ADONIS. ADONIS already provides a customizable exporting component that may generate workflow descriptions for different commercial WFMS.

Figure 12 provides an overview of the architecture our prototype. We are using the standard ADONIS application library for a visualization of the workflow models. Alternatively a workflow model can be visualized as a hidden markov model, using a special application library, which we have developed for this purpose. We are making use of this feature, when applying the prototype to regular languages.

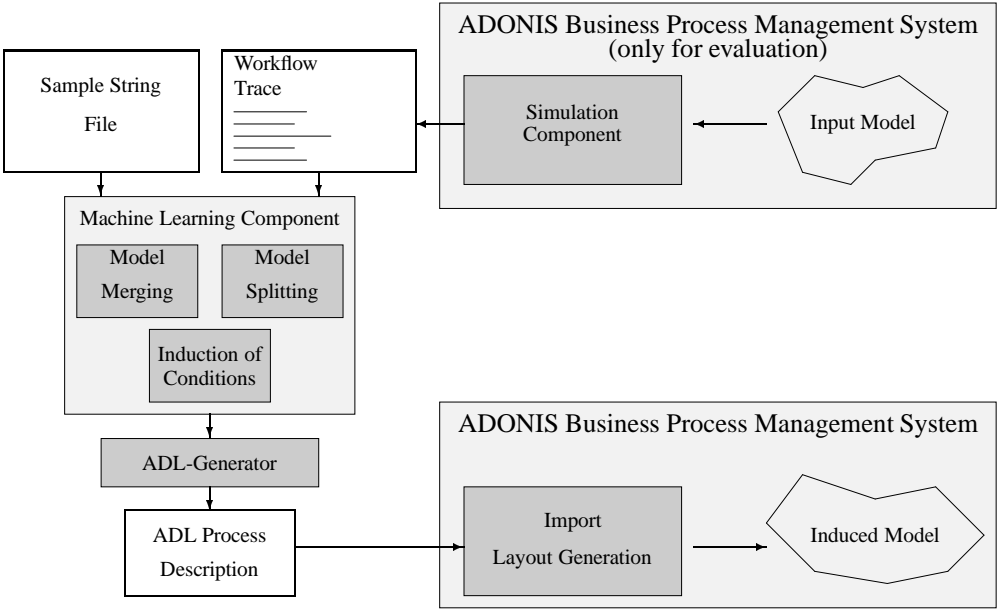


Figure 12: The prototype architecture

Experiments

We have evaluated model merging and model splitting experimentally in a series of applications. Such an evaluation is essential for a number of reasons:

- to determine the quality of the results, to prove the viability of the developed concepts
- to learn about the performance of the learning algorithms in different settings
- to compare the split and the merge approaches in terms of results and performance
- to compare our likelihood-based approach with bayesian model merging

Some of the main results of our evaluation are presented in the following two subsections. We proceed in two stages. First of all example sets containing a very small number of examples from simple regular languages are used for a first basic evaluation of the concept, this allows the reader to validate the results induced by the prototype. In the second stage we apply the learning algorithms to workflow traces, generated using simulation on a provided input workflow model.

For an application of our prototype to a real existing process we refer to Herbst (1999), where we apply our approach to a simplified release process of the Mercedes Benz passenger car development division.

Regular Languages

First of all we evaluate the prototype in three scenarios using three example sets from simple regular languages. These three example sets are given in table 4. The first example set is identical to the set that was used throughout this paper to explain model merging and splitting. The second and third example set are taken from Stolcke and Omohundro (1994). This allows us to compare our algorithms directly with the bayesian model merging approach.

| Example set 1 | Example set 2 | Example set 3 |
|---------------|---------------|---------------|
| abcac | aa | abab |
| abcc | bb | aabab |
| abcabcc | aca | abbab |
| abcabcac | bcb | abaab |
| | acca | ababb |
| | bccb | aaabab |
| | accca | abbbab |
| | bcccb | abaaab |
| | | ababbb |

Table 4: Examples for three regular languages

To each of these example sets we applied the model merging and the model splitting approach. We tested each approach using three different parameter settings. The results of these experiments are documented in table 5. Due to space limitations, we can not show every HMM induced within the experiments. For each scenario we show the HMM, that should have been found - and that was indeed found for at least one parameter setting in every scenario. Additionally we show the resulting HMMs for some selected experiments, where learning revealed interesting results or failed completely. The result table further contains the number of nodes and edges of the correct HMM and of the resulting HMM for each testcase. You can use these numbers for a quick evaluation of the testcases. This is especially important, for those experiments for which we do not provide a visualization of the resulting HMM. If the number of nodes and edges are printed in bold font, this indicates that the result HMM of the experiment is equivalent to the correct HMM. Here “equivalence” refers to structural equivalence, as the probabilities, which are approximated by empirical counts, may differ depending on the example sets.

Results for example set 1

In five of six testcases of the first scenario the learning algorithms terminated returning the correct HMM given in figure 5. In the last testcase, the `minImprove` parameter for the model splitting algorithm was obviously too high, so model splitting failed. It terminated with an overly general HMM, which could have

| | Merge | Merge | Merge | Split | Split | Split |
|--|--------------|--------------|--------------|--------------|--------------|-------------|
| mergeStart/minImprove | 20 | 20 | 20 | 0.01 | 0.08 | 0.32 |
| preGenFactor | 1.1 | 1.1 | 1.2 | | | |
| genFactor | 1.2 | 1.3 | 1.4 | | | |
| Example set 1 | | | | | | |
| Result model nodes/edges (target model: 7/8) | 7/8 | 7/8 | 7/8 | 7/8 | 7/8 | 6/7 |
| Search steps/Time spent | 6/<1s | 6/<1s | 6/<1s | 2/<1s | 2/<1s | 1/<1s |
| Result shown in Figure | Figure 7 | Figure 7 | Figure 7 | Figure 10 | Figure 10 | - |
| Example set 2 | | | | | | |
| Result model nodes/edges (target model: 8/12) | 8/12 | 8/12 | 7/11 | 12/18 | 8/12 | 5/11 |
| Search steps/Time spent | 10/<1s | 10/<1s | 11/<1s | 7/2s | 3/1s | 0/<1s |
| Result shown in Figure | Figure 13 | Figure 13 | - | - | Figure 13 | - |
| Example set 3 | | | | | | |
| Result model nodes/edges (target model: 6/9) | 10/16 | 9/15 | 6/9 | 8/12 | 4/6 | 4/6 |
| Search steps/Time spent | 15/1s | 16/1s | 19/2s | 4/1s | 0/<1s | 0/<1s |
| Result shown in Figure | - | - | Figure 14 | Figure 15 | Figure - | Figure - |

Table 5: Results for the regular languages

been transformed into the correct model by applying one further split operation.

Results for example set 2

The results for the second scenario show, that in one half of the testcases the correct resulting HMM as shown in figure 13 was found. The experiments on the model splitting indicate, that this success depends on the correct parameter setting. In the first testcase ($\text{minImprove}=0.01$) for the model splitting we received a too specific HMM, in the second case ($\text{minImprove}=0.08$) we got the correct result and in

the last case ($\text{minImprove}=0.32$) an overly general HMM was returned. These results are comparable to the results published in Stolcke and Omohundro (1994) for bayesian model merging, which were also very strongly dependent on the selection of the right prior probability.

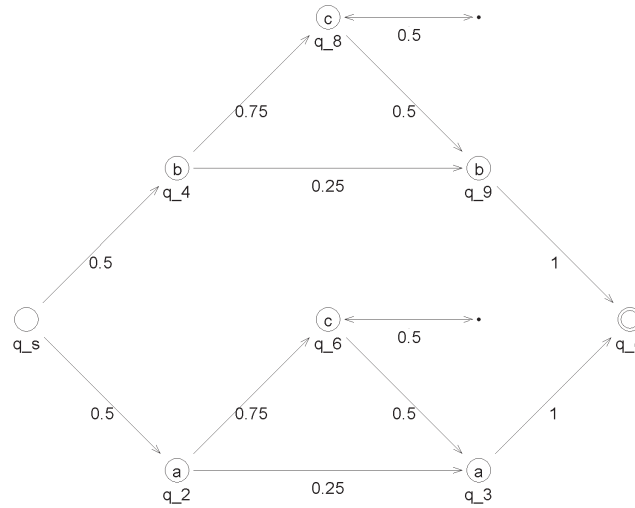


Figure 13: Correct HMM for example set 2

Results for example set 3

In the third scenario only the merging approach succeeded for one parameter setting. Model splitting failed completely in all three test cases. We analyzed multiple search traces and discovered, that the first splitting step is the cause for the failure. The problem is, that after generating the most general HMM, the algorithm decides to split the node associated to the b before the node associated to the a is split. To find the correct result, the a node should be split first, to provide the context for correctly splitting the b. So in this case the heuristic is misleading the search into a suboptimal split decision, which leads to an overly specific model as shown in figure 15. Splitting can only be prevented by selecting a high minImprove parameter, which causes the learner to terminate with the most general HMM. The model merging algorithm does not run

into these problems. Selecting the right parameters allows it to find the correct result as shown in figure 14. After selecting the right parameters the same results are returned by bayesian model merging as reported in Stolcke and Omohundro (1994).

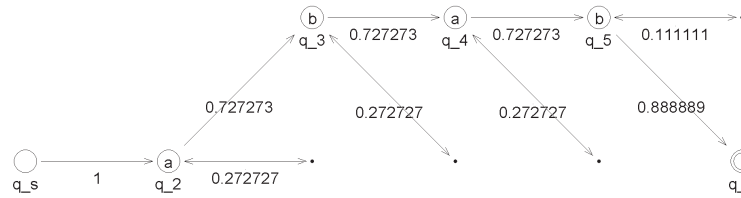


Figure 14: Correct HMM for example set 3

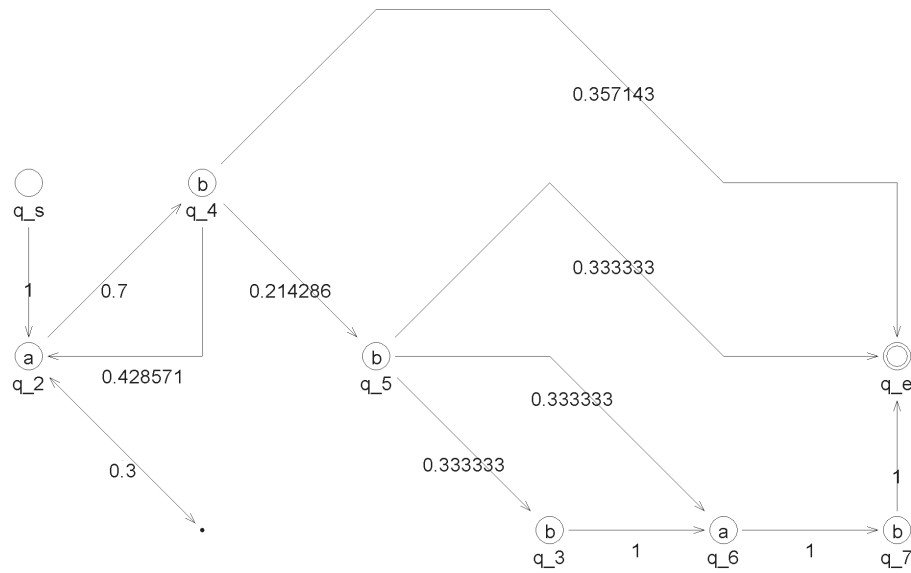


Figure 15: Specialization failed, because of a suboptimal split operation

Workflow Traces

The scenarios described in this section are based on workflow traces generated by the simulation of an input workflow model in ADONIS. Each of the three workflow traces contains approximately 300 examples. To each of these traces we applied model merging and model splitting using three different parameter settings. To determine the quality of the induced model we compared it with the input model. As the input model was discovered in at least one experiment for each scenario, we do not provide separate figures showing the input workflow models. The input models differ from the correct result models only in their transition probabilities, which of course may differ due to the approximation using empirical counts. So if in the following we talk about an input model shown in a certain figure, this only refers to the structure of the model not it's probabilities.

Table 6 summarizes the results for the experiments using workflow traces. For an explanation of the rows and columns we refer to the regular languages section. It must be mentioned though, that the number of nodes and edges mentioned in table 6 do not reflect the number of nodes and edges of the workflow model, but of it's equivalent HMM representation. These numbers differ because HMM's do not contain separate decision and activity nodes. You can verify these numbers if you consider an activity node of a workflow model and it's associated outgoing decision node as one node.

Results for Workflow Trace 1

In the first scenario we generated the trace from the workflow model given in figure 16. This workflow model contains exactly one node for each distinct activity. This is the simplest possible task for the learning component. Given good parameters neither the merging nor the splitting may fail. Any sequence of merge operations applied to the most specific model will sometime lead to the correct result. Success can only be prevented by choosing too small values for the generalization factors. This obviously occurred in the first testcase. For splitting the situation is similar. Because we are using only a small example set of an infinite

| | Merge | Merge | Merge | Split | Split | Split |
|---|--------------|--------------|--------------|--------------|--------------|--------------|
| mergeStart/minImprove | 20 | 20 | 20 | 0.01 | 0.08 | 0.32 |
| preGenFactor | 1.02 | 1.1 | 1.3 | | | |
| genFactor | 1.05 | 1.2 | 1.6 | | | |
| Workflow trace 1 | | | | | | |
| Result model nodes/edges (target model: 7/9) | 10/14 | 7/9 | 7/9 | 7/9 | 7/9 | 7/9 |
| Search steps/Time spent | 60/7s | 44/4s | 40/3s | 0/<1s | 0/<1s | 0/<1s |
| Result shown in Figure | - | Figure 16 | Figure 16 | Figure 16 | Figure 16 | Figure 16 |
| Workflow trace 2 | | | | | | |
| Result model nodes/edges (target model: 7/9) | 7/9 | 7/11 | 6/10 | 7/9 | 7/9 | 7/9 |
| Search steps/Time spent | 56/10s | 46/6s | 43/4s | 2/1s | 2/1s | 2/1s |
| Result shown in Figure | Figure 17 | Figure 18 | - | Figure 17 | Figure 17 | Figure 17 |
| Workflow trace 3 | | | | | | |
| Result model nodes/edges (target model: 11/17) | -/- | 9/22 | 7/17 | 11/17 | 11/17 | 10/17 |
| Search steps/Time spent | -/>8h | 154/217s | 103/48s | 4/2s | 4/2s | 3/2s |
| Result shown in Figure | - | - | - | Figure 19 | Figure 19 | - |

Table 6: Results for workflow traces

set of possible examples, splitting states may improve the likelihood of the model. The correct result can only be found, if the value of the `minImprove` parameter is big enough to prevent specialization. In all three testcases of the splitting approach this parameter was obviously chosen right.

Results for Workflow Trace 2

The workflow model (see figure 17) used to generate the second workflow trace is slightly more complicated. At a first glance it looks the same as the first workflow, but looking at the activities reveals that the a

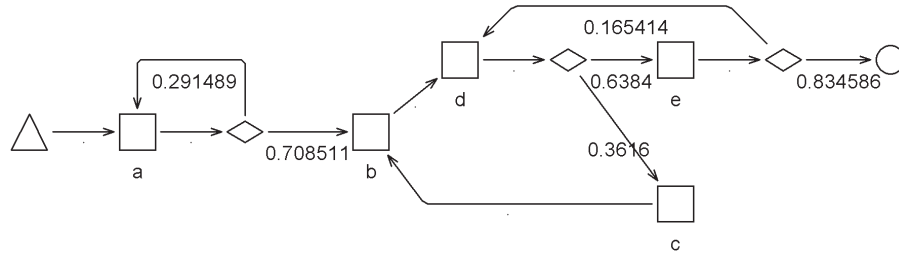


Figure 16: Correct result and input model for workflow trace 1

and the b activity are referenced by two activity nodes.

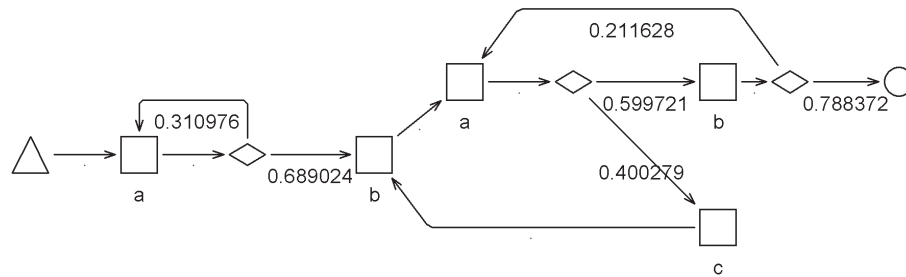


Figure 17: Correct result and input model for workflow trace 2

While splitting is successful for all three parameter settings, merging only returns the correct result for one testcase. Figure 18 shows an interesting case. While the correct number of activity nodes for each of the activities has been identified, the model contains some links which are not present in the input model of figure 17. These links, which can easily be identified by their low probabilities, are due to suboptimal merging decisions.

What also becomes obvious, when looking at the result table for this scenario are the differences in the performance of merging and splitting. Splitting requires less search steps and less time than merging.

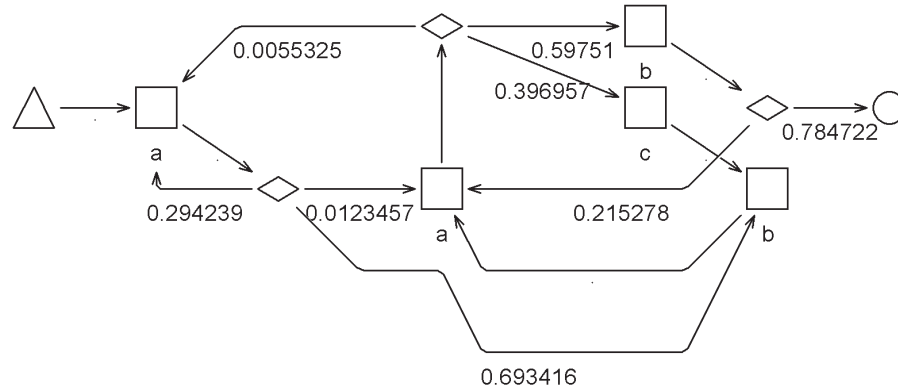


Figure 18: Suboptimal generalization containing unnecessary edges

Results for Workflow Trace 3

The last workflow trace is even more complicated. It is generated by a workflow model (see figure 19) containing many loops and many nodes referring to identical activities.

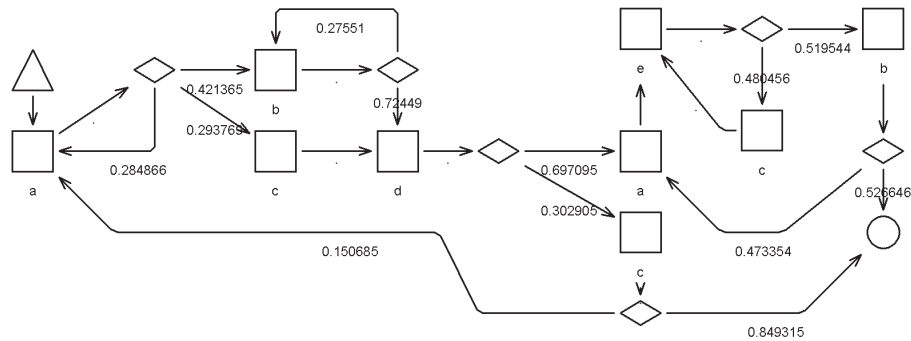


Figure 19: Correct result and input model for workflow trace 3

This scenario reveals the deficits of the merging approach. If the parameters for the first merging phase (`mergeStart` and `preGenFactor`) do not allow enough merging, the search space explodes and performance becomes unacceptable. We aborted the first testcase after 8 hours. A slight change of parameters allowing more merging in the first phase, leads to a drastic improvement of the performance but

also to suboptimal merge decisions, which prevent the algorithm from finding the correct result. We have tried over 100 parameter settings for model merging and in none of these experiments the correct model was found within 8 hours, after which we aborted each trial.

Even in this scenario the splitting approach performs very well and given a right parameter it will find the correct result. It is by far less sensitive to parameter variations than model merging.

Evaluation and Discussion

In all testcases the machine learning component found the right results for at least one parameter setting. The results of our merging algorithm using a pure likelihood heuristic were as good as those of the bayesian model merging algorithm presented in Stolcke and Omohundro (1994) for the small example sets from the regular languages. Due to a suboptimal split decision the model splitting approach failed for one testcase.

When applied to workflow traces with many examples, model splitting produced superior results at a better performance than the model merging algorithm. It was also less sensitive to changes in the parameters. Although generally there is no reason to favor a general-to-specific learner. In the domain of workflow models having only a few nodes associated to the same activity it seems reasonable to start with a most general model. Because fewer search operations (split/merge) lead to the target model, the chance of getting stuck in a local optimum of the heuristic function is less than in the merging approach and the performance is better.

Due to the use of hill-climbing search, the success of both algorithms is of course dependent on the structural complexity of the target model. For some target models it may be necessary to perform a series of merge or split operations, that do not improve the heuristic function, to provide the context for performing some other really important merge or split operations, which improve the heuristic function.

The current performance of the splitting algorithm was fine, even for large example sets. We have even tried example sets with more than 20000 examples and still received an answer within 10 seconds.

The underlying structure of the target model plays a much greater role for the performance as it directly determines the size of the search space. The experiments show, that in the workflow domain the current performance of the model splitting approach leaves enough time to employ a better search algorithm, that will prevent getting stuck in a local optimum of the heuristic function for most practically relevant cases.

When a good model is already provided as prior knowledge, it may make sense to prefer model merging. It should be more efficient, since it may apply an incremental approach. So it might be a good solution to apply model merging in the adaptation phase, when a good model is available as background knowledge and model splitting in the acquisition phase, when there is little knowledge about the structure of the workflow at hand.

The experiments show, that finding the correct model depends on choosing the parameters of the induction algorithms correctly. In practical applications the correct model of course is unknown, which makes it difficult to decide on the parameters. Especially the parameters that determine the degree of specialization/generalization (like `minImprove` and `genFactor`) are critical. Our experiences also show that a process engineer should be able to select good parameters by looking at the log-likelihood curve (like the one shown in figure 8). Providing that enough examples have been observed, structurally significant operations that have major effects on the set of workflow instances covered by the model usually result in large changes of the log-likelihood, while less significant operations like unrolling a loop of a model result in only minor (some orders of magnitude lower) changes of the log-likelihood. With this observation it should be possible to provide the process engineer with additional assistance in finding good parameters.

At the same time for practical applications it makes sense to relax the requirement that the correct model is found. First of all it is questionable whether such a correct model exists at all, as workflows may change as business evolves over time. Furthermore, even if we assume that the workflow model is constant over a certain period of time, there are often several alternative ways for modeling a certain business process. Some process engineers may want to make certain dependencies explicit within the structure of the model and

other process engineers may prefer a structurally overly general model and handle additional dependencies within transition conditions. In any case the models induced by our induction algorithms are more useful for a process engineer than a large collection of unrelated workflow instances. At the same time a slight overgeneralization that can be compensated by appropriate transition conditions or manual user decisions seems to be more useful than an overspecialization. For example a workflow model having a loop that iterates over a certain sequence of activities seems to be more useful than a model having three separate paths with one, two or three repetitions of this sequence. In the first case a transition condition or a manual decision can stop any further iteration after each completion of the sequence. In case separate paths are used, one would have to decide on the number of iterations in advance, which is usually not possible. As workflow splitting has some tendency to overgeneralization this is another reason to prefer the splitting approach over the merging approach.

Related Work

In Agrawal et al. (1998) an approach for mining process models from workflow logs is presented. This approach is based on the induction of directed graphs. In contrast to our approach it is able to deal with concurrency. One major restriction of this approach, which is not present in ours, is that each distinct activity must be associated to one unique node within the workflow model. In terms of markov models this means that there are no hidden states. Applying this method to the examples considered here would thus generate overly general models having only one node for each activity.

In Cook and Wolf (1996) three different methods for the induction of formal process models are presented and evaluated. The first one called RNET is based on a recurrent neural network. It shows some limitations concerning accuracy, performance and usability. The authors consider RNET as inferior to the other two methods they present. The second method is a purely algorithmic method called KTAIL. It is

based on a merging method similar to ours. In contrast to our log-likelihood based heuristic, this method merges states whenever they have the following k symbols in common, where k is a parameter to the algorithm. As this basic method tends to produce overly complex automata due to the fixed length context, Cook and Wolf describe additional mechanisms that reduce the complexity of the model by further state merging, based on the predecessor state and the output behavior of the states considered for merging. Nevertheless Cook and Wolf observed that this method leads to overly complex models when a large k is selected and the accuracy suffers if k is too small. Both our merging and our splitting approach do not suffer from this problem, because they selectively consider a large context of a certain state only where necessary for improving the log-likelihood. The third method presented in Cook and Wolf (1996) is called MARKOV. It is similar to our splitting approach. To split states this method counts occurrences of two and three symbol length sequences and splits states that enable the generation of certain three symbol length sequences, which were not observed in the samples. While this approach does a complete search in a very limited search space (a state that is result of a split is never split again) our approach does an incomplete search in an infinite search space. It is not clear how the limited search space of MARKOV could be extended because there are exponentially many possible k -length sequences and keeping a table of these seems intractable. In contrast the fairly simple search method of our approach can be very easily replaced by an enhanced search method like e.g. beam-search or simulated annealing. Some of the more recent work Cook and Wolf (1998) by the same authors also deals with concurrent process models. It uses three different metrics for the number, frequency and regularity of event sequences to estimate a model of the concurrent process. But this approach is restricted to process models having unique activity nodes and would thus overgeneralize the examples we have considered here.

Another approach called RAP is presented in Bocionek and Mitchell (1993). RAP is a system which supports the users in reserving meeting and lecture rooms. It uses a technique called dialog-based learning to acquire a workflow model represented as a finite state machine by observing the transfer of structured

e-mails. RAP has some limitations concerning inconsistent user behavior (compare Bocionek and Mitchell (1993)), which are not present in our approach.

The idea of merging technologies for organizational learning and workflow management is presented in Wargitsch (1997). The developed system uses completed business cases stored in an organizational memory to configure new workflows. The selection of an appropriate historical case is supported by a case-based reasoning component. While we focus on repetitive processes this approach was designed for more unstructured, project-like processes.

Similar approaches have also been considered in the area of design process management. Some early work reported in Casotto et al. (1990) describes a design process management system called VOV, which is based on design traces. This system records activities and the data dependencies between these activities in a so-called design trace. This design trace is used for the coordination of team design, consistency maintenance and documentation. While the basic idea of defining process models by executions is the same as ours, the methods used in VOV are rather limited. VOV simply records one instance without applying any generalization.

Summary, Conclusions and Future Work

We described a machine learning component, which is able to induce a sequential workflow model from traces of manually enacted workflow instances. This component is realized by the composition of two machine learning algorithms, one for the induction of the structure and one for the induction of transition conditions. While we have presented two different alternative solutions for the induction of the structure in detail, we have outlined only the basic idea for the induction of the conditions. The two alternative algorithms for the induction of the structure - model merging and model splitting - have been implemented in a research prototype. Our experimental evaluation shows that both algorithms provide useful results. In

the workflow domain, model splitting produced superior results at a better performance.

The are convinced that the integration of machine learning algorithms will provide a number of significant improvements to WFMS. We are thus doing further research in this area. The main goal of our ongoing work is extending the learning algorithms towards being able to deal with concurrent workflow models. If we applied the presented approach to workflow traces generated by a concurrent model, an overly complex representation of the workflow, containing all possible orderings of each concurrent set of activities would be induced. This of course is not optimal, but it nevertheless is a useful input for a process engineer.

We are currently working on an extension of our induction algorithm, which in contrast to the other learning algorithms presented in (Agrawal et al. (1998)) and (Cook and Wolf (1998)), is not restricted to workflow models with unique activity nodes. The main challenge in inducing concurrent workflow models lies in the detection of dependencies between activities. Once dependencies have been discovered an enhanced version of the model splitting algorithm described here can be applied. This basic idea is outlined in Herbst and Karagiannis (1999). Further work must also be done for dealing with noise, caused for example by erroneous workflow instances. Finally we need to integrate our prototype into an adaptive WFMS for an evaluation using real world workflow instances.

Acknowledgements

We would like to thank the development team at the BOC GmbH for providing the interface to ADONIS.

References

- K. R. Abbott and S. K. Sarin. Experiences with Workflow Management: Issues for the Next Generation. In *Proceedings of the Conference on Computer Supported Cooperative Work CSCW94*, pages 113–120. ACM Press, 1994.

- R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Proceedings of the sixth International Conference on Extending Database Technology (EDBT)*, 1998.
- C. Agyris and D.A. Schoen, editors. *Organizational Learning: A Theory of Action Perspective*. Addison-Wesley Publishing Company, Inc., 1978.
- BOC. *ADONIS Version 3.0 - Users Guide*. BOC GmbH, Vienna, 1999.
- S. Bocionek and T. M. Mitchell. Office Automation Systems that are Programmed by their Users. In *23. Jahrestagung der Gesellschaft für Informatik*, pages 214–219, Berlin, Germany, 1993. Springer-Verlag.
- R. C. Carrasco and J. Oncina. Learning Stochastic Regular Grammars by Means of a State Merging Method. In *Second International Colloquium on Grammatical Inference and Applications (ICGI94)*, volume 862 of *Lecture Notes on Artificial Intelligence*, pages 139–152, Berlin, Germany, 1994. Springer-Verlag.
- A. Casotto, A. R. Newton, and A. Sagiovanni-Vincentelli. Design Management based on Design Traces. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 136–141, 1990.
- J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. Technical Report, CU-CS-819-96, Department of Computer Science, University of Colorado, 1996.
- J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. Technical Report, CU-CS-860-98, Department of Computer Science, University of Colorado, 1998.
- B. Curtis, M.I. Kellner, and J. Over. Process Modeling. *Communications of the ACM*, pages 75–90, September 1992.
- C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic Change Within Workflow Systems. In *Proceedings of the Conference on Organizational Computing Systems*, pages 10–21. ACM Press, 1995.
- E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

- E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3): 302–320, 1978.
- P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. DYNAMITE: Dynamic Task Nets for Software Process Management. In *Proceedings of the 18th International Conference on Software Engineering*, pages 331–341, 1996.
- J. Herbst. Inducing Workflow Models from Workflow Instances. In *Proceedings of the 6th European Concurrent Engineering Conference*, pages 175–182. Society for Computer Simulation (SCS), 1999.
- J. Herbst and J. Bumiller. Towards Engineering Process Management Systems. In *Proceedings of the Concurrent Engineering Europe Conference*, pages 109–115. Society for Computer Simulation (SCS), 1997.
- J. Herbst, S. Junginger, and H. Kühn. Simulation in Financial Services with the Business Process Management System Adonis. In *Proceedings of the European Simulation Symposium 1997 (ESS97)*. Society for Computer Simulation (SCS), 1997.
- J. Herbst and D. Karagiannis. Integrating Machine Learning and Workflow Management to Support Acquisition and Adaptation of Workflow Models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE, 1998.
- J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In *Proceedings of the IJCAI 99 Workshop for Intelligent Workflow and Process Management: The New Frontier for AI in Business*. IJCAI, 1999.
- C. Hewitt. Offices Are Open Systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

- D. Karagiannis, S. Junginger, and R. Strobl. Introduction to Business Process Management Systems. In B. Scholz-Reiter and E. Stickel, editors, *Business Process Modeling*, pages 81–106. Springer-Verlag, 1996.
- F. Leymann and D. Roller. *Production Workflows: Concepts and Techniques*. Prentice Hall, New Jersey, 1999.
- R. Parekh and V. Honavar. Automata Induction, Grammar Inference, and Language Acquisition. In Dale, Moisl, and Somers, editors, *Handbook of Natural Language Processing*. New York: Marcel Dekker, 1999.
- R. Pareschi, G. de Michelis, and S. Sarin. Introduction to the Workshop on Adaptive Workflow. In *Proceedings of the First International Conference on Practical Aspects of Knowledge Management PAKM96*, pages 1–4, 1996.
- J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.
- M. Reichert and P. Dadam. Adept_{flex} - Supporting Dynamic Changes of Workflows Without Loosing Control. Ulmer Informatik Berichte Nr. 97-07, University of Ulm, 1997.
- S. K. Sarin. Workflow and Data Management in InConcert. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, pages 497–499. IEEE Computer Society, 1996.
- A. Stolcke and S. Omohundro. Best-First Model Merging for Hidden Markov Model Induction. Technical Report, TR-94-003, International Computer Science Institute (ICSI), 1994.
- Teamware. *TeamWARE Flow 2.0 - Work Management Software for the Way People Work Document number: PT00401E*. TeamWARE Group Oy, Helsinki, Finland, 1997.

- C. Wargitsch. WorkBrain: Merging Organizational Memory and Workflow Management Systems. In *Workshop of Knowledge-Based Systems for Knowledge Management in Enterprises at the 21st annual German Conference on AI (KI-97)*, pages 214–219, Kaiserslautern, Germany, 1997. Deutsches Forschungszentrum für Künstliche Intelligenz.
- C. Wargitsch. *Ein Beitrag zur Integration von Workflow- und Wissensmanagement unter besonderer Berücksichtigung komplexer Geschäftsprozesse*. PhD thesis, Universität Erlangen-Nürnberg, November 1998.

Notes

¹This is a revised and extended version of Herbst and Karagiannis (1998), that was presented at the Ninth International Workshop on Database and Expert Systems Applications 1998

²ADONIS is a registered trademark by BOC GmbH