# Supporting Flexible Processes Through Recommendations Based on History

Helen Schonenberg, Barbara Weber, Boudewijn van Dongen and Wil van der Aalst

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.h.schonenberg, b.f.v.dongen, w.m.p.v.d.aalst@tue.nl}@tue.nl
Department of Computer Science, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at

**Abstract.** In today's fast changing business environment flexible Process Aware Information Systems (PAISs) are required to allow companies to rapidly adjust their business processes to changes in the environment. However, increasing flexibility in large PAISs usually leads to less guidance for its users and consequently requires more experienced users. In order to allow for flexible systems with a high degree of support, intelligent user assistance is required. In this paper we propose a recommendation service, which, when used in combination with flexible PAISs, can support end users during process execution by giving recommendations on possible next steps. Recommendations are generated based on similar past process executions by considering the specific optimization goals. In this paper we also evaluate the proposed recommendation service, which is implemented in ProM, by means of experiments.

## 1 Introduction

In todays fast changing business environment, flexible Process Aware Information Systems (PAISs) are required to allow companies to rapidly adjust their business processes to changes in the environment [6, 7]. PAISs offer promising perspectives and there are several paradigms, e.g., adaptive process management [13], case handling systems [17] and declarative processes [10, 11] (for an overview see [3,19,21]). In general, in flexible PAIS it occurs frequently that users working on a case have the option to decide between several activities that are enabled for that case. However, for all flexibility approaches, the user support provided by the PAIS decreases with increasing flexibility (cf. Fig. 1), requiring users to have in-depth knowledge about the processes they are working on. Traditionally, this problem is solved by educating users (e.g., by making them more aware of the context in which a case is executed), or by restricting the PAIS by introducing more and more constraints on the order of activities and thus sacrificing flexibility. Both options, however, are not satisfactory and limit the practical application of flexible PAISs. In this paper, we present an approach for intelligent user assistance which allows PAISs to overcome this problem and to provide a better balance between flexibility and support. We use event logs

of PAISs to gain insights into the process being supported without involving a process analyst and we propose a tooling framework to provide continuously improving support for users of flexible PAISs. At the basis of our approach lie so-called recommendations. A recommendation provides a user with information about how he should proceed with a partially completed case (i.e., a case that was started but not finished yet), to achieve a certain goal (e.g., minimizing cycle time, or maximizing profit). In this paper we discuss several methods for calculating log-based recommendations. In addition, we describe the implementation of our approach as recommendation service and its evaluation. The remainder of this paper is structured as follows. In Section 2, we present the requirements and a detailed overview of the recommendation service. Then, in Section 3 we define a log-based recommendation service. In Section 4, we describe the experiment we conducted to evaluate whether recommendations indeed help to achieve a particular goal. Finally, we discuss related work in Section 5 and provide conclusions in Section 6.

## 2 Overview

Fig. 2 illustrates the envisioned support of users of flexible PAISs through a recommendation service. In general, each business process to be supported is described as process model in the respective PAIS. We consider both imperative and declarative process models. In fact, our approach is most useful when the process model provides the user a lot freedom to manoeuvre, i.e., multiple activities are enabled during execution of a case. At run-time, cases are created and executed considering the constraints imposed by the process model. In addition, the PAIS records information about executed activities in event logs. Typically, event logs contain information about start and completion of activities, their ordering, resources which executed them and the case they belong to [1].

As illustrated in Fig. 2, the recommendation service is initiated by a request from the user for recommendations on possible next activities to execute. In this request, the user sends the recommendation service information about the partially executed case, i.e., (1) the currently enabled activities, and (2) the history of executed activities, which we call the partial trace. Information about the partial trace is required because the decision which activities to perform next for a particular case usually depends on the activities already performed for this case. In addition, only enabled activities are considered to ensure that no recommendations are made that violate the constraints imposed by the process model. The recommendation service then provides the PAIS a recommendation result, i.e., an ordering of recommendations where each recommendation refers to one activity and some quality attributes (e.g., expected outcome) explaining the recommendation. Recommendations are ordered such that the first recommendation in the list is most likely to help the user achieving his goal, i.e., optimizing a certain target, such as profit, cost, or cycle time.

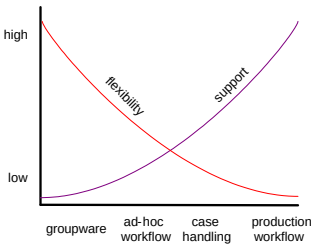


**Fig. 1.** PAIS trade-offs [4].

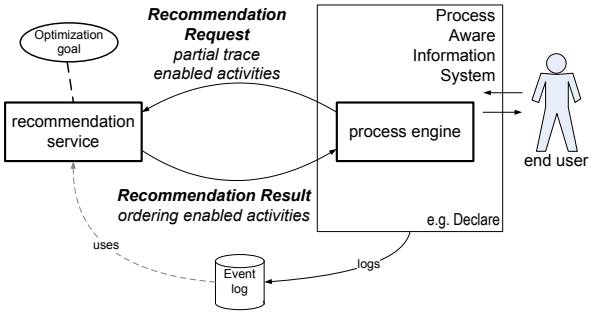


**Fig. 2.** An overview of the recommendation service.

## 3 Log-Based Recommendation Service

In this section, we present a concrete definition of a log-based recommendation service for providing users with recommendations on next possible activities to execute. Recommendations for an enabled activity provide predictive information about the user goal, based on observations from the past, i.e., fully completed traces accompanied by their target value (e.g., cost, cycle time, or profit), that have been stored in an event log. The log-based recommendation service requires the presence of an event log that contains such information about cases that have been executed for a certain process.

### 3.1 Preliminaries

Let $A$ be a set of activities. $A^*$ denotes a set of finite sequences over $A$. A trace $\sigma \in A^*$ is a finite sequence of activities, given by an injective function $\sigma : \{1, \ldots, n\} \to A$, where $|\sigma| = n$ is the length of the sequence. Sequences are denoted as $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, where $\forall_{1 \leq i \leq n}\ \sigma(i) = a_i$.

On traces, we define the standard set of operators.

**Definition 1 (Trace operators).** *Let* $\sigma : \{1, \ldots, n\} \to A$ *and* $\sigma' : \{1, \ldots, m\} \to B$ *be traces with* $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ *and* $\sigma' = \langle b_1, b_2, \ldots, b_m \rangle$.

| | |
|---|---|
| **Prefix** | $\sigma \leq \sigma' \iff n \leq m \wedge \forall_{1 \leq i \leq n}\ a_i = b_i$ |
| **Concatenation** | $\sigma^\frown\sigma' = \langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m \rangle$ |
| **Membership** | $a \in \sigma \iff \exists_{1 \leq i \leq n}\ a_i = a$ |
| **Parikh vector** | $par(\sigma)(a) = \#_{0 \leq i \leq n}\ a_i = a.$ |

The Parikh vector $par(\sigma)(a)$ denotes the number of occurrences of $a$ in a trace $\sigma$, e.g., $par(\langle a, b, c, a, b, c, d \rangle)(a) = 2$.

For multi-sets (bags), we introduce standard notation to denote the universe of multi-sets over a given set. Let $S$ be a set, then the universe of multi-sets over $S$ is denoted by $\mathcal{B}(S) = S \to \mathbb{N}$, i.e., $X \in \mathcal{B}(S)$ is a multi-set, where for all $s \in S$ holds that $X(s)$ denotes the number of occurrences of $s$ in $X(s)$. We will use $[\![a, b^2, c^3]\!]$ to denote the multi-set of one $a$, two $b$'s and three $c$'s as a shorthand

for the multi-set $X \in \mathcal{B}(A)$ where $A = \{a, b, c\}$, $X(a) = 1, X(b) = c, X(c) = 3$. Furthermore, multi-set operators such as for for union $\uplus$, intersection $\uplus$, and submulti-set $\sqsubseteq$, $\sqsubset$ are defined in a straightforward way and can handle a mixture of sets and multi-sets.

**Definition 2 (Event log).** *Let $A$ be a set of activities. An event log $L \in \mathcal{B}(A^*)$ is a multi-set of traces referring to the activities in $A$.*

Recall that each recommendation contains predictive information regarding the user goal. For now, we assume that this goal can be captured by a function on a trace, i.e., each trace $\sigma$ in an event log has a target value (e.g., cost, cycle time, or profit) attached to it.

**Definition 3 (Target Function).** *Let $A$ be a set of activities and $L \in \mathcal{B}(A^*)$ an event log over $A$. A target function $\tau : L \to \mathbb{R}$ is a function which attaches a target value to each trace in the log.*

### 3.2 Recommendations

A recommendation is initiated by a recommendation request, which consists of a partial trace and a set of enabled activities. Formally, we define a recommendation request as follows.

**Definition 4 (Recommendation request).** *Let $A$ be a set of activities and $\rho \in A^*$ a partial trace. Furthermore, let $E \subseteq A$ be a set of* enabled activities. *We call $r = (\rho, E)$ a recommendation request.*

An activity accompanied by predictive information regarding the user goal is called a recommendation. For each enabled activity, we determine the expected target value when doing this activity (*do*), and the expected target value for alternatives of the enabled activity, i.e., other enabled activities (*dont*). Precise definitions of *do* and *dont* are given in Definitions 10 and 11. A recommendation result is an ordering over recommendations.

**Definition 5 (Recommendations).** *Let $A$ be a set of activities and $L \in \mathcal{B}(A^*)$ an event log over $A$. Furthermore, let $(\rho, E)$ be a recommendation request with $E \subseteq A$, $|E| = n$ and $e \in E$ an enabled activity.*

- $\big(e, do(e, \rho, L), dont(e, \rho, L)\big) \in E \times \mathbb{R} \times \mathbb{R}$ *is a recommendation. We use $\mathfrak{R}$ to denote the universe of recommendations.*
- *A recommendation result $\mathcal{R} = \langle \big(e_1, do(e_1, \rho, L), dont(e_1, \rho, L)\big), \big(e_2, do(e_2, \rho, L), dont(e_2, \rho, L)\big), \ldots, \big(e_n, do(e_n, \rho, L), dont(e_n, \rho, L)\big)\rangle$ is a sequence of recommendations, such that $\mathcal{R} \in \mathfrak{R}^*$ and $\forall_{1 \leq i < j \leq n}\ e_i \neq e_j$.*

The nature of the ordering over recommendations is kept abstract, however, we provide a possible ordering for a recommendation result in Example 1, Section 3.6. In the next section we describe how recommendations are generated by the recommendation service based on an existing event log $L$.

### 3.3 Trace Abstraction

When generating log-based recommendations only those traces from the event log should be considered, which are relevant for determining the predictive information of an enabled activity. From those traces the ones with a high degree of matching with the partial trace execution should be weighted higher than those with small or no match.

To determine which log traces are relevant to provide recommendations for a given partial trace and to weight them according to their degree of matching we need suitable comparison mechanisms for traces. Our recommendation service provides three different trace abstractions based on which traces can be compared, namely, prefix, set and multi-set abstraction. The prefix abstraction basically allows for a direct comparison between the partial trace and a log trace. In practice such a direct comparison is not always relevant, e.g., when the ordering, or frequency of activities is not important. Therefore we provide with set and multi-set two additional abstractions. They are independent of the domain context, e.g., they do not assume the process to be a procurement process or an invoice handling process [16].

**Definition 6 (Trace abstraction).** *Let $A$ be a set of activities, $L \in \mathcal{B}(A^*)$ be an event log and $\sigma \in L$ be a trace. $\sigma_p = \sigma$ denotes the prefix abstraction of $\sigma$, $\sigma_s = \{a \mid a \in \sigma\}$ denotes the set abstraction of $\sigma$ and $\sigma_m = par(\sigma)$ denotes the multi-set abstraction of $\sigma$, i.e., for all $a \in \sigma$ holds that $\sigma_m(a) = par(\sigma)(a)$.*

In Section 3.4 we explain how we determine which log traces are relevant for obtaining predictive information of an enabled activity. In Section 3.5 we describe how we calculate the weighting of log traces.

### 3.4 Support

The relevance of log traces for a recommendation is determined on basis of support. Typically, traces that are relevant are those that support the enabled activity for which the recommendation is computed. What support exactly means here, depends on the trace abstraction used.

For the prefix abstraction, we say that a log trace $\sigma$ supports enabled activity $e$, if and only if e occurs in $\sigma$ at the same index as in the partial trace $\rho$, when this activity is executed. For set abstraction, we consider a log trace $\sigma$ to support the enabled activity $e$ whenever activity $e$ has been observed at least once in the log trace. To support an enabled activity $e$ in multi-set abstraction of trace $\sigma$, the the frequency of activity $e$ in the partial trace $\rho$ must be less than the frequency in the log trace $\sigma$, i.e., by executing $e$ after $\rho$, the total number of $e$'s does not exceed the number of $e$'s in $\sigma$.

**Definition 7 (Activity support functions).** *Let $A$ be a set of activities, $\rho, \sigma \in A^*$ and enabled activity $e \in A$. We use the predicate $s(\rho, \sigma, e)$ to state*

*that log trace $\sigma$ supports the execution of $e$ after partial trace $\rho$. The predicate is defined for the three abstractions by:*

$$\begin{aligned} s_p(\rho, \sigma, e) &\iff \sigma_p(|\rho| + 1) = e \\ s_s(\rho, \sigma, e) &\iff e \in \sigma_s \\ s_m(\rho, \sigma, e) &\iff \rho_m(e) < \sigma_m(e) \end{aligned}$$

The support predicate is used to filter the event log by removing all traces that do not support an enabled activity.

**Definition 8 (Support filtering).** *Let $A$ be a set of activities and $L \in \mathcal{B}(A^*)$ an event log over $A$. Furthermore, let $(\rho, E)$ be a recommendation request with $\rho \in A^*$ and $E \subseteq A$. We define the log filtered on support of enabled activity $e \in E$ and partial trace $\rho$ as $L^s_{(\rho,e)} = [\![\sigma \in L \mid s(\rho, \sigma, e)]\!]$*[1]

Log traces from $L^s_{(\rho,e)}$ support enabled activity $e$ and are used for the recommendation of $e$. Next, we define a weighing function ($\omega$) to express the relative importance of each of these log traces for the recommendation of an enabled activity $e$.

### 3.5 Trace Weight

The support of an enabled activity determines the part of the log that serves as a basis for a recommendation. However, from the traces supporting an enabled activity, not every one is equally important, i.e., some log traces match the partial trace better than others. Hence, we define weighing functions that assign a weight to each log trace. The weight of a trace can be between 1 and 0, where a value of 1 indicates that two traces fully match and 0 that they do not match at all. The calculation of the degree of matching depends on the trace abstraction. For prefixes, the weight of a log trace is 1 if the partial trace is a prefix of the log trace, otherwise, the weight is 0. For the set abstraction, the weight of the log trace is defined as the fraction of distinct partial trace activities that the partial trace abstraction and log trace abstraction have in common. The weight of a trace for the multi-set abstraction is similar to the set-weight, however, the frequency of activities is also considered.

**Definition 9 (Weight functions).** *Let $A$ be a set of activities and $\sigma, \rho \in A^*$. We define $\omega(\rho, \sigma)$,i.e., the relative importance of a log trace $\sigma$ when considering the partial trace $\rho$ as follows:*

$$\omega_p(\rho, \sigma) = \begin{cases} 1 \text{ , } if \quad \rho_p \leq \sigma_p \\ 0 \text{ , } otherwise \end{cases} \text{ , } \omega_s(\rho, \sigma) = \frac{|\rho_s \cap \sigma_s|}{|\rho_s|} \text{ , } \omega_m(\rho, \sigma) = \frac{|\rho_m \uplus \sigma_m|}{|\rho_m|}$$

[1] Note that $\sigma$ ranges over a multi-set traces.

### 3.6 Expected Outcome

Definition 5 states that a recommendation for enabled activity $e$ contains predictive information about the target value. We define the expected outcome of the target value (*do* value), when $e$ is executed in the next step, as a weighted average over target values of log traces from $L^s_{(\rho,e)}$, the log filtered on support of $e$. The target value of each trace from $L^s_{(\rho,e)}$ is weighted ($\omega$) on basis of the degree of matching with the partial trace.

**Definition 10 (*do* calculation).** *Let $A$ be a set of activities, $\tau$ a target function, $\rho, \sigma \in A^*$, $L \in \mathcal{B}(A^*)$ and $e \in E \subseteq A$ an enabled activity. The expected target value when $\rho$ is completed by the user after performing activity $e$ next is defined as:*

$$do(e, \rho, L) = \frac{\sum_{\sigma \in L^s_{(\rho,e)}} \omega(\rho, \sigma) \cdot \tau(\sigma)}{\sum_{\sigma \in L^s_{(\rho,e)}} \omega(\rho, \sigma)}$$

Similarly, we define the expected target value of not doing an enabled activity $e$[2]. The *dont* function determines the weighted average over *all* alternatives of $e$, i.e., all traces that do not support the execution of $e$ after $\rho$, but do support any of the alternatives $e'$ after $\rho$.

**Definition 11 (*dont* calculation).** *Let $A$ be a set of activities, $\tau$ a target function, $\rho, \sigma \in A^*$, $L \in \mathcal{B}(A^*)$ and $e, e' \in E \subseteq A$ enabled activities. The expected target value when $\rho$ is completed by the user after not performing activity $e$ next is defined as:*

$$dont(e, \rho, L) = \frac{\sum_{e' \in E \setminus \{e\}} \sum_{\sigma \in L^s_{(\rho,e)} \setminus L^s_{(\rho,e')}} \omega(\rho, \sigma) \cdot \tau(\sigma)}{\sum_{e' \in E \setminus \{e\}} \sum_{\sigma \in L^s_{(\rho,e')} \setminus L^s_{(\rho,e)}} \omega(\rho, \sigma)}$$

Next, we provide an example calculation for a recommendation, based on a concrete partial trace, a set of enabled events and a log.

*Example 1 (Recommendation).* Suppose $\rho = \langle D, F \rangle$ is a partial trace and $E = \{A, B, C\}$ is the set of enabled activities. Together, they form a recommendation request $(\rho, E)$. The log is given by $L = [\![\langle A, B, C \rangle, \langle D, B, C \rangle, \ldots]\!]$, with $\tau(\langle A, B, C \rangle) = 900$, $\tau(\langle D, B, C \rangle) = 500$, etc. (cf. Fig. 3). For convenience, we also provide the values for support ($s_s(\rho, \sigma, e)$) and trace weight ($\omega_s(\rho, \sigma)$). For each log trace, support is denoted by $\top$. The user wants to minimize the cost and uses set abstraction. The *do* and *dont* values for the recommendation are calculated as follows.

[2] Note that in both *do* and *dont* $\sigma$ ranges over a multi-set of traces

| Log | | Weight and support | | | |
|---|---|---|---|---|---|
| $\sigma$ | cost | $\omega_s(\rho,\sigma)$ | $s_s(\rho,\sigma,e)$ | | |
| | | | $e=A$ | $e=B$ | $e=C$ |
| ABC | 900 | 0 | $\top$ | $\top$ | $\top$ |
| DBC | 500 | 0.5 | | $\top$ | $\top$ |
| FBC | 500 | 0.5 | | $\top$ | $\top$ |
| DFA | 1000 | 1 | $\top$ | | |
| DFB | 1500 | 1 | | $\top$ | |
| DFC | 2000 | 1 | | | $\top$ |
| DFH | 1260 | 1 | | | |
| CCA | 1680 | 0 | $\top$ | | $\top$ |

**Fig. 3.** Example log, with weight and support values.

$$do(A,\langle D,F\rangle,L) = \frac{0\cdot 900 + 1\cdot 1000 + 0\cdot 1680}{0+1+0} = 1000$$

$$do(B,\langle D,F\rangle,L) = \frac{0\cdot 900 + 0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 1500}{0+0.5+0.5+1} = 1000$$

$$do(C,\langle D,F\rangle,L) = \frac{0\cdot 900 + 0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 2000 + 0\cdot 1680}{0+0.5+0.5+1+0} = 1250$$

$$dont(A,\langle D,F\rangle,L) = \frac{(0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 1500)}{(0.5+0.5+1)+(0.5+0.5+1)} + \frac{(0.5\cdot 500 + 0.5\cdot 500 + 1\cdot 2000)}{(0.5+0.5+1)+(0.5+0.5+1)} = 1125$$

$$dont(B,\langle D,F\rangle,L) = \frac{(1\cdot 1000 + 0\cdot 1680) + (1\cdot 2000 + 0\cdot 1680)}{(1+0.5)+(1+0)} = 1500$$

$$dont(C,\langle D,F\rangle,L) = \frac{(1\cdot 1000) + (1\cdot 1500)}{(1)+(1)} = 1250$$

The implementation of our recommendation service orders the enabled activities on the difference between *do* and *dont*, i.e., the bigger the difference, the more attractive the activity is. The recommendations for the enabled activities are (A, 1000, 1125),(B, 1000, 1500) and (C, 1250, 1250), with the differences of -125, -500 and 0 respectively. Thus, the recommendation result is $\langle (B,1000,1500),(A,1000,1125),(C,1250,1250)\rangle$. If the user goal would be to maximize costs, the order will be reversed.

## 4 Evaluation Based on a Controlled Experiment

To evaluate the effectiveness of our recommendation service we conducted a controlled experiment. Section 4.1 describes the design underlying our experiment and Section 4.2 describes the preparatory steps we conducted. Section 4.3 explains the experiment procedure including data analysis. The results of our experiment are presented in Section 4.4. Factors threatening the validity of our experiment are discussed in 4.5.

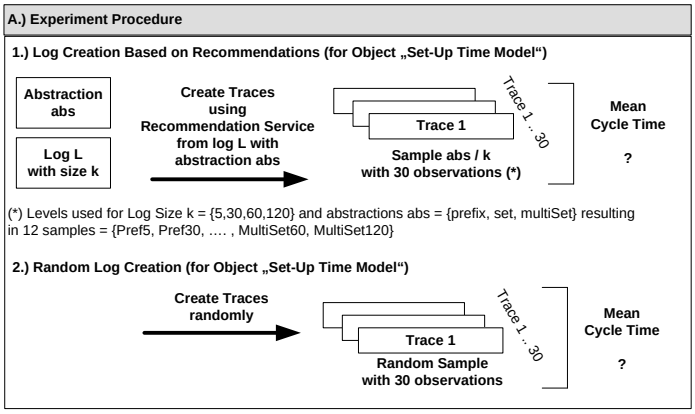


**Fig. 4.** The experiment design.

In our experiment we use the recommendation service to support a business process, for which the set-up time plays an important role. Five activities $(A, B, C, D, E)$ have to be executed exactly once and can be executed in any order. Each activity has a cycle time of 10 time units, however, if $C$ is directly executed after $B$, then there is no closing-time for activity $B$ and no set-up time for activity $C$ leading to a cycle time reduction. In this case only 35 time units are needed for the entire trace instead of 50. During execution, only activities that have not been executed before, are enabled. For the experiment we assume that the user goal is to minimize the cycle time for cases of this process and that the recommendation service is used for support.

### 4.1 Experiment Design

This section describes the design underlying our experiment.

- **Object:** The object to be studied in our experiment are the traces created for the *set-up time model* with the help of our recommendation service.
- **Independent Variables:** In our experiment we consider the log abstraction and the log size as independent variables. For variable *log abstraction* we consider levels $abs \in \{prefix, set, multiset\}$ (cf Section 3.3). Variable *Log size k* represents the number of instances in the event log, i.e., the amount of learning material based on which recommendations are made. As levels $k \in \{5, 30, 60, 120\}$ are considered.
- **Response Variable:** The response variable in our experiment is the cycle time of a trace created by the recommendation service using a log of a given size and a given abstraction.
- **Experiment Goal:** The main goal of our experiment is to investigate whether changes in the log significantly effect the cycle time[3] of the cre-

[3] Note that our approach can also be used for costs, quality, utilization, etc. However, for simplicity we focus on the cycle time only.

ated traces given an abstraction. Another goal is to investigate whether the traces created by our recommendation service yield significantly better results than randomly created traces.

### 4.2 Experiment Preparation

This section describes the preparatory steps we conducted for the experiment.

- **Implementing the Recommendation Service in ProM.** As a preparation for our experiment we implemented the recommendation service described in Section 3 as a plug-in for the (Pro)cess (M)ining framework ProM[4]. ProM is a pluggable framework that provides a wide variety of plug-ins to extract information about a process from event logs [20], e.g., a process model, an organizational model or decision point information can be discovered. To implement the recommendation service we had to make several extensions to ProM as the recommendation service, in contrast to other plug-ins, is not a posteriori mining technique, but recommendations are provided in real-time during process execution. The implementation of our recommendation service is able to provide a process engine with recommendations on possible next steps knowing the enabled activities and the partial trace. In addition, the recommendation service provides means to add finished cases to the event log to make them available for recommendations in future executions.
- **Implementing a Log Creator and Log Simulator.** In addition to the recommendation service we implemented a log creator and log simulator. While the log creator allows us to randomly create logs of size $k$ for a given process model, the log simulator can be used to create traces using the recommendation service with a log of size $k$ and an abstraction $abs$. The log simulator takes the constraints imposed by the process model into consideration and ensures that no constraint violations can occur. Thus, the log simulator can be seen as a simulation of a process engine. Both the log creator and the log simulator have been implemented in Java using Fitnesse[5] as user interface. This allows us to configure our experiments in a fast and efficient way using a WIKI and to fully automate their execution.

### 4.3 Experiment Execution and Data Analysis

The experiment procedure including the analysis of the collected data is described in this section.

- **Generation of Data.** As illustrated in Fig. 4 our experiment design comprises two independent variables (i.e., log abstraction $abs$ and log size $k$). As a first step a log of size $k$ is randomly created using the log creator, which is then - in a second step - taken by the log simulator as input to create

[4] The ProM framework can be downloaded from `www.processmining.org`.
[5] Fitness Acceptance Testing Framework `fitnesse.org`

traces for each combination of abstractions and log sizes. Traces are created based on the recommendations provided by the recommendation service described in Section 3. The recommendations given by the recommendation service are used throughout the entire execution of the case whereby the best recommendation (i.e., the one with the highest difference of *do* and *dont* values, see. Section 3.6) is taken. For each completed trace the log simulator records the cycle time. We repeated (n=30) the process of producing a log and creating a trace using the log simulator with this log as input. In total we obtained 12 samples covering all combinations of log size levels and abstraction levels. For example, sample `PREF5` represents the sample with $abs = prefix$ and $k = 5$.
In addition to the 12 samples which are created using recommendations, we created one sample with 30 randomly created traces to compare this sample with the ones created using the recommendation service.

- **Effects of Changes in Log Size and Abstraction.** To analyse the effects of changes in the log size and the selection of a particular abstraction on the cycle time of the created traces we calculated 95% confidence intervals (CI 95%) on the mean cycle time for each sample. Doing so, we can say with 95% probability, for a given log size and a given abstraction, that the cycle time of a created trace will be within the calculated confidence interval.
If we then compare the confidence intervals of two samples of a given abstraction (e.g., `PREF5` and `PREF10`) and these intervals do not overlap, we can assume that the two samples have statistically different cycle times.
- **Effectiveness of Abstractions.** To investigate whether the traces created by our recommendation service yield significantly better results than randomly created traces we compared the confidence interval of the random sample with the confidence intervals of each of the other 12 samples.

### 4.4 Experiment Results

The results of our experiment are summarized in Figures 5-10. The results of our experiments are summarized in Figures 5-10. Figures 5-7 depict the effect of the log size on the mean cycle time for the 12 samples created using recommendations. In Figures 8-10 we compare the different abstractions and the random strategy for a fixed log size.

- **Increasing the Log Size.** Figure 5 clearly shows the impact of increasing the log size on the cycle time for prefix abstraction. The mean cycle time for sample `PREF5` is given by a CI 95% [38.68,44.32] and for samples `PREF30`, `PREF60` and `PREF120` the mean cycle time is 35 (with a standard deviation of 0), which is also the optimum cycle time. When studying the results of Fig. 5 changing the log size from $k = 5$ to $k = 30$ yields a significant decrease of the cycle time. As the confidence intervals of samples `PREF5` and `PREF30` are not overlapping the difference in their cycle times is statistically significantly different. Further increases in the log size have no effect since the optimum cycle time has already been found for sample `PREF30`. Figure 6

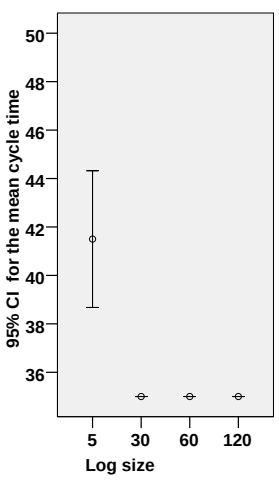


**Fig. 5.** Prefix.

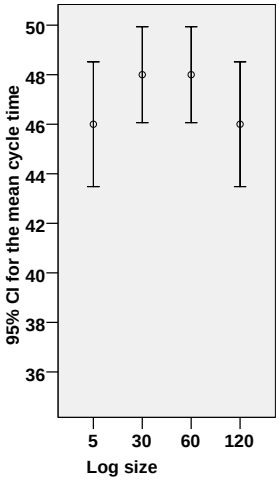


**Fig. 6.** Set.

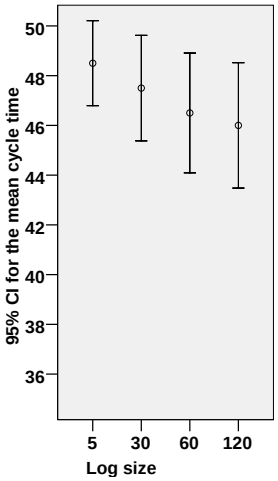


**Fig. 7.** Multi-set

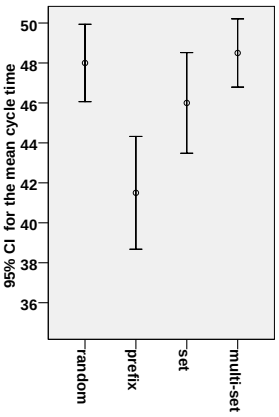


**Fig. 8.** Log size = 5.

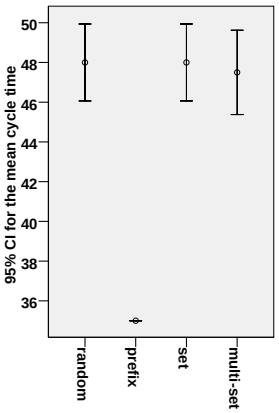


**Fig. 9.** Log size = 30.

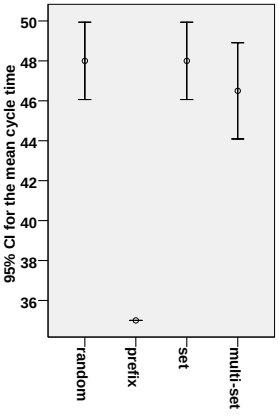


**Fig. 10.** Log size = 60.

and 7 shows the results for the set and multi-set abstraction. As all intervals are overlapping we can conclude that there is no significant improvement in the cycle time for the set and multi-set abstraction.

– **Comparing the Abstractions.** Figure 8-10 compares randomly created traces with the samples for prefix, set and multi-set. It can be observed that the prefix abstraction (i.e., samples `PREF5`, `PREF30` and `PREF60`) has significantly better cycle times compared to the random sample and thus outperforms the random selection strategy. As illustrated in Figure 8-10 the difference between random selection and prefix abstraction becomes bigger with increasing log size. The prefix abstraction also outperforms the multi-set abstraction for all considered log sizes. A comparison of the set and prefix

abstraction reveals that no significant differences exist between PREF5 and SET5, while `PREF30` and `SET30` as well as `PREF60` and `SET60` significantly differ. Finally, between the samples of the set and multi-set abstraction no significant differences can be observed.

In summary, our results show that an increase of the log size does effect the mean cycle time for the prefix abstraction and that this abstraction significantly outperforms the random selection strategy. For the set and multi-set abstraction changes in the log size do not significantly effect the mean cycle time. As these abstractions cannot exploit the order characteristics of the traces in the log they do not outperform the random selection strategy.

### 4.5 Risk Analysis

In the following we discuss factors potentially threatening the validity of our experiment. In general, it can be differentiated between threats to the internal validity (*Are the claims we made about our measurements correct?*) and threats to the external validity (*Can we generalize the claims we made?*) [9]. For our experiment most relevant threats affect the external validity:

- **Selection of Process Model.** In the selection of the business process for our experiment constitutes a threat to the external validity of our experiment. Given the properties of the chosen process model, a particular order of executing activities yields a benefit. As the prefix abstraction considers the exact ordering of activities, while the set and multi-set abstractions disregard this information, the chosen process model is favouring the prefix abstraction. For process models with different characteristics other abstractions might be more favourable. Therefore it cannot be generalized that the prefix abstraction is always better than the set and multi-set abstraction. A family of experiments using process models with different characteristics is needed for generalization. Initial investigations with a business process which is not order-oriented show that set and multi-set abstraction can perform significantly better than random selection.
- **Method of Log Creation.** For our experiment the method we used for log creation might constitute another threat to the external validity. We assume a log that only contains randomly created traces as the input for the log simulator. Using the simulator we then create, based on this log, an additional trace considering recommendations. In practice such an assumption might not always be realistic as a real-life log will most probably contain a mixture of randomly created traces and traces created using recommendations., i.e., by random/explorative and experience-based ways of working. First experiments indicate that the degree to which a log contains random traces compared to traces created based on recommendations also influences the cycle time. However, like for completely random logs an increase of the log size has led to decreases in the cycle time, but the slope of the decrease tends to be steeper for higher ratios of random behaviour in the log. An

extensive investigation of logs with different ratios of random traces will be subject of further studies.

## 5 Related Work

The need for flexible PAISs has been recognized and several competing paradigms (e.g., adaptive process management [8, 13, 23], case-handling systems [17] and declarative processes [11]) have been proposed by both academia and industry (for an overview see [21, 22]). In all these approaches the described trade-off between flexibility requiring user assistance can be observed.

Adaptive PAIS represent one of these paradigms by enabling users to make structural process changes to support both the handling of exceptions and the evolution of business processes. Related work in the context of adaptive PAISs addresses user support in exceptional situations. Both ProCycle [12, 14] and CAKE2 [8] support users to conduct instance specific changes through change reuse. While their focus is on process changes, our recommendation service assists users in selecting among enabled activities. ProCycle and CAKE2 use case-based reasoning techniques to support change reuse. Therefore suggestions to the users are based on single experiences, (i.e., the most similar case from the past), while in our approach recommendations are based on the entire log.

In [18] recommendations are used to select the step, which meets the performance goals of the process best. Like in our approach selection strategies (e.g., lowest cost, shortest remaining processing time) are used. However, the recommendations are not based on a log, but on a product data model.

Except for adaptive process management technology, which allows for structural changes of the process model and the case-handling paradigm, which provides flexibility by focusing on the whole case, many approaches support flexibility by allowing the design of a process with regions (placeholders) whose contents is unknown at design-time and whose content is specified (Late Modeling) or selected (Late Binding) during execution of the tasks (for details see [21, 22]). Examples of such approaches are, for instance, Worklets [2] or the Pockets of Flexibility [15] approach. Both approaches provide user assistance by providing simple support for the reuse of previously selected or defined strategies, recommendations as envisioned in our approach are not considered.

In addition to the approaches described above there is a third paradigm for flexible workflows, which relies on a declarative way of modeling business processes. As opposed to imperative languages that "procedurally describe sequences of action", declarative languages "describe the dependency relationships between tasks" [5]. Generally, declarative languages propose modeling constraints that drive the model enactment [11]. When working with these systems, users have the freedom to choose between a variety of possibilities as long as they remain within the boundaries set by the constraints [10, 11]. In the context of declarative workflows user assistance has not been addressed so far.

## 6 Conclusion

Existing PAISs are struggling to balance support and flexibility. Classical workflow systems provide process support by dictating the control-flow while groupware-like systems offer flexibility but provide hardly any process support. By using recommendations, we aim at offering support based on earlier experiences but not limit the user by imposing rigid control-flow structures. In this paper we presented an approach based on recommendations, i.e., based on a process model providing a lot of flexibility the set of possible activities is ranked based on *do* and *dont* values. The recommendation is based on (1) a configurable abstraction mechanism to compare the current partial case with earlier cases and (2) a target function (e.g., to minimize costs or cycle time). The whole approach has been implemented by extending ProM and can be combined with any PAIS that records events and offers work through worklists.

The experimental results in this paper show the *value of information*, i.e., the more historic information is used, the better the quality of the recommendation. We experimented with different abstractions and log sizes. Clearly, the performance depends on the characteristics of the process and the abstraction. However, the experiments show that traces executed by support of recommendations often outperform traces executed without such support. This is illustrated by the difference in performance between the random selection and appropriate guided selection.

Future work will aim at characterizing the suitability of the various abstraction notions. Through a large number of real-life and simulated experiments we aim at providing insights into the expected performance of the recommendation service. Moreover, we aim at an intelligent selection of the horizon for the log used to "train" the recommendation service. Our experiments show that it is not wise to provide recommendations based on just a few observations. However, if there are many observations, it makes sense to only use a subset (e.g., only the most recent traces). This way the performance of the approach improves for long-running processes and the recommendation strategy can easily adapt to changing circumstances.

**Acknowledgements** The authors would like to thank Maja Pesic and Christian Günther, Isaac Corro Ramos, Christian Haisjackl for their help and contributions.

## References

1. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
2. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *Coopis'06*, 2006.
3. S. Carlsen, J. Krogstie, A. Slvberg, and O.I. Lindland. Evaluating flexible workflow systems. In *Proc. HICSS-30)*, 1997.
4. B.F. van Dongen and W.M.P. van der Aalst. A meta model for process mining data. In *EMOI-INTEROP*, 2005.

5. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: mediating between representation and action in workflow systems. In *Proc. CSCW '96*, pages 190–198, 1996.
6. M. Hammer and S.A. Stanton. *The Reengineering Revolution – The Handbook*. Harper Collins Publ., 1995.
7. P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A Comprehensive Approach to Flexibility in Workflow Management Systems. In *Proc. WACC '99*, pages 79–88, New York, NY, USA, 1999. ACM.
8. M. Minor, A. Tartakovski, D. Schmalen, and R. Bergmann. Agile Workflow Technology and Case-Based Change Reuse for Long-Term Processes. *International Journal of Intelligent Information Technologies*, 1(4):80–98, 2008.
9. B. Mutschler, B. Weber, and M. U. Reichert. Workflow management versus case handling: Results from a controlled software experiment. In *Proc. SAC'08*. ACM Press, 2008.
10. M. Pesic and W.M.P van der Aalst. A Declarative Approach for Flexible Business Processes. In *Proc. DPM'06*, pages 169–180, 2006.
11. M. Pesic, M.H. Schonenberg, N. Sidorova, and W.M.P v. d. Aalst. Constraint-Based Workflow Models: Change Made Easy. In *CoopIS'07*, 2007.
12. J. Pinggera, S. Zugal, B. Weber, W. Wild, and M. Reichert. Integrating Case-Based Reasoning with Adaptive Process Management. Technical report, CTIT, University of Twente, Enschede, 2008.
13. M. Reichert and P. Dadam. ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. *JIIS*, 10(2):93–129, 1998.
14. S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution - a semantics based approach. In *BPM 2005*, pages 252–267, 2005.
15. S. Sadiq, W. Sadiq, and M. Orlowska. A Framework for Constraint Specification and Validation in Flexible Workflows. *Information Systems*, 30(5):349 – 378, 2005.
16. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, , and C.W. Günther. Process mining: A two-step approach using transition systems and regions. Technical Report BPM-06-30, BPMcenter.org, 2006.
17. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering.*, 53(2):129–162, 2005.
18. I. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product based workflow support: A recommendation service for dynamic workflow execution. Technical Report BPM-08-03, BPMcenter.org, 2008.
19. W.M.P. v.d. Aalst and S. Jablonski. Dealing with workflow change: Identification of issues an solutions. *Int'l Journal of Comp. Systems, Science and Engineering*, 15(5):267–276, 2000.
20. W.M.P v.d. Aalst, B.F van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 27(2):237–267, 2003.
21. B. Weber, S. Rinderle, and M. Reichert. Change Patterns and Change Support Features in Process-Aware Information Systems. In *Proc. CAiSE'07*, pages 574–588, 2007.
22. B. Weber, S. Rinderle, and M. Reichert. Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical Report TR-CTIT-07-76, University of Twente, 2007.
23. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In *Proc. HICSS-34*, 2001.