Supporting Flexible Processes Through Recommendations Based on History

B. Weber[1], B.F. van Dongen[2], M. Pesic[2], C.W. Guenther[2], and W.M.P. van der Aalst[2]

[1] Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at
[2] Department of Technology Management, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{b.f.v.dongen,m.pesic,c.w.gunther,w.m.p.v.d.aalst}@tue.nl

Abstract. In today's fast changing business environment flexible information systems are required to allow companies to rapidly adjust their business processes to changes in the environment. However, increasing flexibility in large information system 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 guidance, intelligent user assistance is required. In this paper we propose a recommendation service, which, when used in combination with flexible information systems, can guide 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. This paper also describes an implementation of the proposed recommendation service in the context of ProM and the declarative workflow management system DECLARE.

1 Introduction

In today's fast changing business environment, flexible information systems are required to allow companies to rapidly adjust their business processes to changes in the environment. In order to achieve this high degree of flexibility, both academia and industry have recently proposed several paradigms of flexible information systems (e.g., adaptive process management [11], case handling systems [12] and declarative processes [10]). However, in most of these approaches flexibility comes at a cost, i.e. the more flexible a process-aware information system is, the less support it provides to its users and hence the more knowledge these users need to have about the process they are a part of. On the other hand, full support in an information system usually comes at a cost of loosing flexibility (cf. Figure 1).

Although users of flexible systems have the option to make their own decisions while working, a certain level of support is still necessary. Reasons for this can be various: unexperienced users, exceptional situations, personal preferences, etc. Traditionally, this problem is solved by educating workers (e.g. by

making them more process aware), or by restricting the information system by introducing more and more constraints and thus sacrificing flexibility. Obviously, both options are costly. Moreover, they both require a process specialist to gain insights in the process supported by the information system, either to educate the workers about these processes or to change them into more restrictive ones.

The research area of *process mining* has provided many methods and tools to help a process specialist in acquiring such insights. In this paper, we use techniques from the process mining domain and make them directly available to the workers in a process, i.e. we introduce a *recommendation service* that provides more support for users in a flexible environment, without requiring a business specialist to analyze the process in detail.

Process mining addresses the problem that most "process owners" have very limited information about what is actually happening in their organization. In practice, there is often a significant gap between what is prescribed or supposed to happen, and what *actually* happens. Only a concise assessment of the organizational reality, which process mining strives to deliver, can help in verifying process models, and ultimately be used in a process (re)design effort.

The idea of process mining is to discover, monitor and improve *real* processes (i.e., not assumed processes) by extracting knowledge from event logs [1]. Clearly, process mining is most relevant in a setting where much flexibility is allowed or needed, and therefore this is an important topic in this paper. The more ways in which people and organizations can deviate, the more variability and the more interesting it is to observe and analyze processes as they were and are executed.

So far, process mining has always been seen as a stand-alone application, i.e. an event log is taken and used to produce a model (e.g., a Petri net or social network), or to extend an existing model (e.g., building a process model into a simulation model). The recommendation service we introduce in this paper however, more or less applies process mining on-the-fly, i.e. by looking at a log (set of completed executions) and *a partial execution*, predictions can be made about the future of the (partial) case. This logically leads to *recommendations*.

Recommendations can be considered as predictions about a case, conditioned on the next step that has not been performed yet. For example, given the partial execution of the current case and the completed executions of similar cases in the log, it is predicted that this case will take 10 hours if the next step is activity "A". However, if the next step is activity "B", it will last only 4 hours.

In order to recommend to a user what should be the next step in a process, the recommendation service needs to know what the user's target (goal) is, e.g. should the user perform its tasks as soon as possible, or should s/he optimize its outcome in terms of business value. Consider for example an insurance claim handling process. Obviously, users should complete each claim as quickly as possible since that optimizes the throughput and hence the costs of the claim handling department are kept minimal. In a sales department on the other hand, it might be a good idea not to give up after the first rejection by a potential client, i.e. by spending *more* time on that sale, the probability that it is successfully concluded might increase.

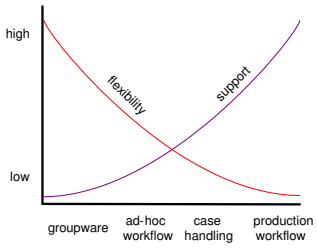

Fig. 1. PAIS trade-offs [3].

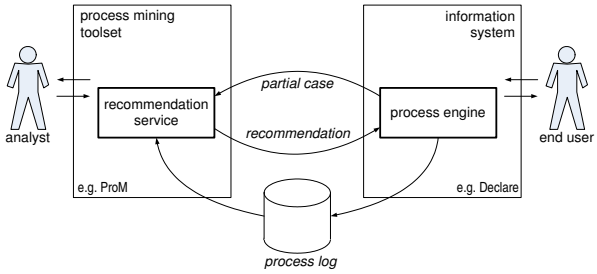

Fig. 2. An overview of the recommendation service.

Figure 2 shows an overview of the recommendation service that we present in this paper. The process engine of some information system (or any other system that supports processes) logs information of some sort. This information is used by the recommendation service as a basis, i.e. it bases its recommendation on the information in this log. Then, a user requests a recommendation for a specific case by sending a partial case, i.e. the user sends a record of all the steps performed in a case up to the point where the recommendation is needed. The recommendation service then answers by sending a recommendation to assist the user in choosing the next step(s). Such a recommendation consists of a list of advised next steps (e.g. "call potential customer again") combined with a number of quality attributes (e.g. this recommendation is based on 100 similar cases, which with a 90% probability lead to a successful sale).

In the remainder of this paper, we first introduce related work in Section 2. Then, in Section 3, we introduce the recommendation service in more detail and we show an implementation thereof in Section 4. We conclude the paper with Section 5 where we provide conclusions and future work.

2 Related Work

Before presenting the recommendation service in more detail, we first provide an overview of the work in the area of run-time flexibility in process enactment as well as user support in flexible systems.

In order to provide flexible information systems both academia and industry recently have proposed several different paradigms for flexible information systems. In all these paradigms exists the described trade-off between flexibility requiring user assistance (e.g., through recommendations).

Adaptive process management systems represent one of these paradigms by enabling users to make instance-specific adaptations and by supporting the evolution of process models (e.g., through version management and instance migration). ADEPT[11] is an example of such an adaptive system, where users can add, delete, and move tasks during the execution. However, users have to rely on their experience and expertise to decide when to add, delete or move a task.

The need to support users when flexible system support is provided has been addressed by the ProCycle approach, which combines ADEPT with case-based reasoning technology (CBRFlow) [17]. ProCylce allows starting with an initial simple process model and learning from the living system. End users are supported by the system to handle exceptional situations through the reuse of deviations. This is achieved through the semantical annotation of changes. In addition, ProCycle allows deriving process improvements when a particular deviation occurs frequently. Unlike in our approach, recommendations are not provided by the system in a proactive way, but user interaction is required for initiating user assistance. In addition, the retrieval of similar deviations for reuse requires user interaction as well. In combination with ProCycle our approach could be used to proactively make suggestions for possible instance-specific process adaptations by taking similar past cases into account. Thereby, not only execution logs, but also change logs can be considered [6]. For example, if the next task according to the process model is "A" but the recommendation is task "D", the user might get the suggestion to insert task "D" before task "A".

Similar to ProCycle, the Uranos project [8] provides authoring support to assist users in the case of process adaptations. Using case-based reasoning techniques similar past workflows are retrieved, which can then be used as a starting point for workflow adaptations. For similarity calculation, the structure of the workflow, its execution state and context information is considered. Like in the ProCyle approach, user interaction is required to get recommendations for process steps. In addition. most of the context information considered in the Uranos project is not automatically available, but has to be manually entered in the system, which might lead to significant knowledge elicitation efforts.

Another paradigm are case-handling systems, which provide flexibility by focusing on the whole case (process instance), not individual tasks [12]. For example, in FLOWer [9], when working on one case, users can: open waiting tasks, execute enabled tasks, skip waiting and enabled tasks, and redo executed and skipped tasks. The topic of user assistance in the context of case-handling systems has not been addressed so far. However, in this context recommendations could, for example, discover that one task is often skipped in a certain type of cases and recommend skipping for new cases of this type. For example, if the next two tasks according to the model are "A" and "B" and recommendation is task "B", this is a signal for the user to skip task "A" in this case.

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 deal with the flexibility problem by defining regions in the process model where changes can be performed during run-time (e.g., through Late Modeling or Late Binding) (for details see [18]). Examples of such approaches are, for instance, Worklets [2] or the Pockets of flexibility [13] approach. Both approaches provide user assistance by providing simple support for the reuse of changes, recommendations as envisioned in our approach are not considered.

In contrast to the approaches described above, the third paradigm addressed in our implementation of the recommendation service, 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 [5, 7, 15]. Constraints describe dependencies between model elements and are specified using pre and post conditions for target task [15], dependencies between states of tasks (enabled, active, ready, etc.) [5] or various model-related concepts [7]. 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. Again, a recommendation system can help users by identifying similar situations from the past and showing how these situations were solved.

3 The Recommendation Service

As stated before, the recommendation service uses a process log to provide support for users in a flexible environment. To do so, it requires a process log as a basis. This process log is analyzed and when a user requires a recommendation he sends a partial case to the recommendation service. This partial case is compared to the log and a recommendation is generated. In this section, we introduce process logs in more detail, we show how this comparison can be done, and which recommendations result from this comparison. Note that we do not claim that the techniques in this section are exhaustive. Instead, they should be seen as a meaningful instantiation of the ideas presented in Section 1.

Throughout this section, we use an illustrative example which we use to show how a recommendation service can provide the right recommendations at the right time. Consider three friends visiting a karaoke bar to have a good time. However, their ideas of having a good time do not match, i.e. Mary wants to be a singer later, so she wants to practice as much as possible. Bill had a bad day at work, so he just wants to get as drunk as possible and Peter has to stay sober, since he is the driver who takes Mary and Bill home.

At the karaoke bar, there are five things that Mary, Bill and Peter can do (i.e. there are five activities) namely: *sing*, *listen*, *drink beer*, *drink wine* and *drink juice*. Furthermore, there are some rules that should be followed. Obviously, a person cannot sing and listen at the same time (i.e. concurrently). Furthermore, nobody is allowed to drink two drinks concurrently and during singing, drinking is also not allowed.

Mary, Bill and Peter have visited the karaoke bar before, but to no success. Therefore, they asked their friends who go there regularly to "log their activities", i.e. to write down what they did. This time, Mary, Bill and Peter will use this log, of which a fragment is shown in Table 1, to get the best out of their evening.

Table 1. Karaoke bar log.

person	activity	start time	end time
George	sing	20:00	20:04
Susan	listen	20:00	20:04
Susan	drink wine	20:00	20:03
Jerry	listen	20:00	20:04
Jerry	drink juice	20:00	20:05
Susan	drink beer	20:03	20:04
Jerry	listen	20:04	20:09
George	sing	20:04	20:09
Susan	listen	20:04	20:09
Susan	drink beer	20:06	20:07
Jerry	listen	20:09	20:13
George	listen	20:09	20:13
George	drink beer	20:09	20:10
George	drink juice	20:11	20:15
...			

Table 2. Rewritten karaoke bar log.

case	activity	event	time
1: George	sing	start	20:00
1: George	sing	complete	20:04
1: George	sing	start	20:04
1: George	sing	complete	20:09
1: George	listen	start	20:09
1: George	drink beer	start	20:09
1: George	drink beer	complete	20:10
1: George	drink juice	start	20:11
1: George	listen	complete	20:13
1: George	drink juice	complete	20:15
2: Susan	listen	start	20:00
2: Susan	drink wine	start	20:00
2: Susan	drink wine	complete	20:03
2: Susan	drink beer	start	20:03
2: Susan	listen	complete	20:04
2: Susan	drink beer	complete	20:04
2: Susan	listen	start	20:04
2: Susan	drink beer	start	20:06
2: Susan	drink beer	complete	20:07
2: Susan	listen	complete	20:09
...			

3.1 Process Logs

For the purpose of deriving recommendations from a process log, we need logs to satisfy certain requirements. These requirements, which are similar to the requirements presented in [3], are also required most process mining approaches, and state that a process log is a collection of events, such that:

1. Each event should be an event that happened at a given point in time. It should not refer to a period of time. For example, starting to work on some work-item in a workflow system would be an event, as well as finishing the work-item. The process of working on the work-item itself is not. Our example log of Table 1 satisfies this property, as it shows both the beginning and the end of each activity, which can be considered separate events.
2. Each event should refer to one activity only, and activities should be uniquely identifiable. In our example log of Table 1 this requirement is fullfilled.
3. Each event should contain a description of the event that happened with respect to the log, we show the start and the end of each activity, which are the respective descriptions.
4. Each event should refer to a specific case. We need to know, for example, for which invoice the payment activity was started. In our example, the log only contains one visit of three people to the karaoke bar and hence we consider each of these people as individual cases.

Note that our description of a process log is such that it can be considered as a bag of cases, where each case is an ordered list of events (the ordering is induced by the required timestamp). Table 2 shows the same log as Table 1, translated to meet all the requirements.

Before we show how the recommendation service deals with process logs exactly, we first introduce another important part of the recommendation service, the so-called *target function*. When a user asks for a recommendation, then he does so with certain goals in mind. Such a goal is formalized using a target function.

3.2 Target Functions

The goal of the recommendation system is to support users in their decisions, such that a certain target value is optimized. These targets can be anything, from the duration of a case to whether or not a sale is successful. To determine how successful each case in the log was, we introduce *target functions*. These functions map a given case to a number, e.g. in case of duration, it maps the case to the number of minutes or seconds that case lasted. However, any data value can be used, if it is stored somewhere in the event log or can be derived from it.

Recall that the three people involved in our karaoke process have three different goals (target functions), i.e. Mary wants to be a singer, Bill wants to get as drunk as possible and Peter wants to stay sober, so he can drive. To capture the fact that different processes (or even different users within the same process) can have different objectives, many different target function can be thought of, but we consider the following to be the most relevant:

1. The *duration* of a case. When a user is trying to finish cases as soon as possible, the target for the recommendation service should be to minimize the duration of a case (hence to maximize one divided by the duration of a case).
2. The *business value* of a case. When trying to sell insurances for example, the user would like to get recommendations that lead to a sale in the end, hence the target function should express this. Since process logs usually contain data, this target function typically optimizes for a certain data element expressing the business value of that case.
3. The *learning curve* of a person. Since the recommendation service has access to the entire log, it is able to determine which people performed which activities. Therefore, a manager can decide to influence the recommendation service, such that each user performs each activity with the same frequency. This way, the users are constantly learning and developing their skills.
4. The *value for money* of a case, i.e. the business value per time unit. If cases that last longer are unreasonable more expensive, then it is logical to try to optimize for earning the most money per time unit. However, if a certain activity is expected to take long, but with a high cost/benefit ratio, then it might be a good idea to still perform this activity.

Table 3. Evaluation of karaoke target functions.

Case	Mary duration of singing in minutes	Bill drinks per hour	Peter non-alcoholic
1: George	9	1/15	0
2: Susan	0	3/9	0
3: Jerry	0	0	1

When we consider our karaoke example and we look into the log of Table 1 in more detail, then we see that drinking wine always takes considerably longer than drinking beer. Therefore, when trying to get as drunk as possible, the target function should be such that it counts the number of drinks per hour (i.e. the value for money category). However, when staying sober, the target function should be such that it should only provide a positive outcome for cases where no alcohol was consumed and zero value otherwise. Furthermore, when trying to become a singer, only the duration of all singing activities combined is important. Table 3 shows the results of the three target functions on the log of Table 2.

From Table 3, it is easy to see that Bill should best follow Susan when he wants to get drunk, i.e. Susan drank the most drinks per hour. Furthermore, following George is also possible (although this is not the best option), whereas following Jerry would not make sense. The latter statement shows the idea of the recommendation service. In this case it should lead Bill to "follow" the same steps as Susan did. However, for this, we need to formalize what this "following" means, i.e. we need a means to compare cases.

The recommendation service we present in this paper uses process logs to base recommendations on. It does so by comparing the partial instance a user sends to the service with the cases already present in the process log. However, if we would not abstract from the log at all, then it is very unlikely that two cases can be compared to each other, i.e. in real life, no two cases are dealing with exactly the same data, involving exactly the same people at the same time.

Again consider our karaoke example. Since the recommendation service has no knowledge about Mary, but only about George, Susan and Jerry, there should be some abstraction from the log. If Mary, at the beginning of her evening asks the recommendation service what to do next, the recommendation service should ignore the specific persons that performed the logged activities and should tell her to follow Georges lead.

Abstracting from the identities of people is not the only useful abstraction and therefore, in Subsection 3.3, we introduce several abstraction mechanisms and we show how cases can be compared to obtain a degree of similarity between them.

3.3 Comparison Mechanisms

Most existing process mining algorithms use a multi step approach when considering event logs. In the first step, an event log is typically analyzed to come

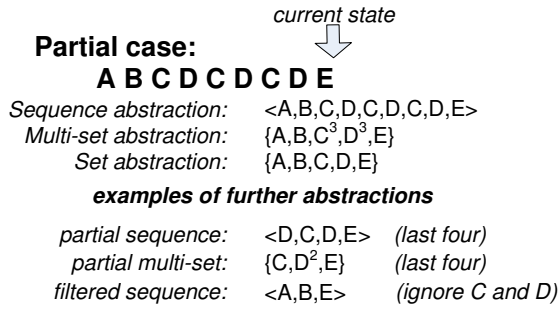

Fig. 3. Possible abstractions of cases.

to some abstraction thereof and in the second step the respective algorithm uses this abstraction as a starting point.

The α-algorithm is a typical example of a multi step algorithm applying process discovery from a process perspective (i.e. discovering the control flow of the underlying process). This algorithm first abstracts from the log by deriving dependencies between the activities in the log (e.g. causal dependencies). These dependencies are then used to construct a Workflow net describing the control flow of the process. Another example is the construction of a social network from an event log, i.e. process discovery from an organizational perspective. In order to do this, the log is first analyzed to derive relations between the people involved in the process and these relations are then used to construct a social network.

Abstracting from the information in the log typically serves two purposes. First, it hides the complexity of the respective mining algorithm and second, it helps to focuss on relevant information. Consider an event log with thousands of cases concerning ten activities and ten people. The complexity of the α-algorithm or an algorithm for social network analysis for example, results from the analysis of the dependencies that are contained in the abstraction. Since deriving the abstraction from the actual event log can be done in polynomial time, the mining algorithms become more or less independent of the size of the log.

The second purpose of abstracting from a log is to focus on relevant information. Again, as an example, consider a conference management system. Logs produced by such a system might show events relating to all people who submitted a paper. However, when trying to analyze the social network of the conference visitors, all events relating to those people who did submit a paper, but did not visit the conference should be removed. This process of removing events from (but also changing events or adding events to) a log when certain criteria are met is called filtering and can be considered as a special case of abstraction.

The recommendation service's main goal is to decide what the next step in an partial case should be. To do so, this *partial case* is compared to the event log containing many completed cases and it is decided which of those completed cases are most similar to the incomplete one. In the remainder of this section,

we introduce several abstraction mechanisms and we show how this comparison should be performed for each of them. Figure 3 shows some examples of how a partial case (and hence a complete case) can be abstracted from.

Finite vs. infinite horizons. First of all, the base for similarity calculation may be using finite or infinite horizons. In some scenarios the entire case has to be considered in order to make meaningful recommendations (hence the horizon is infinite), while in other scenarios it is sufficient to consider the h last events of a case (i.e., postfix of a case). For example, instead of using a complete case $\langle A, B, C, D, C, D, C, E\rangle$ only a limited horizon of 4 activities (h=4) is considered: $\langle D, C, D, E\rangle$. This abstraction is especially useful when the recent past is more relevant. Using this abstraction, deviations which lay far behind are not considered in the similarity calculation. Obviously, the horizon can also be expressed in terms of time (e.g. consider the activities of last week) or other data objects (e.g. consider the activities performed after the sale was made).

Depending on whether a finite or an infinite horizon is considered, similarity is calculated differently. Let us assume that we have a case $\langle A, A, C, D, E, A, F\rangle$ and a partial case $\langle X, X, A, A, C, D, E\rangle$. When considering infinite horizons, the similarity is 1 if the partial case is a prefix of the first case, otherwise 0 (like in our example). When considering a finite horizon of length h, the similarity is 1 if the last h events of the partial case make up an infix of length h of the first case, otherwise 0. Given the above example and a horizon of 3, the similarity between the first case and the partial case is 1 as the last three events of the partial case $\langle C, D, E\rangle$ make up an infix of the first case.

When we consider our karaoke example, then the recommendation service which Mary uses should consider an infinite horizon, since her goal is to optimize the time spent singing during the *entire* evening. Also for Peter, the goal is not to drink the *entire* evening and hence the horizon should be infinite. For Bill however, the idea of a finite horizon is more intuitive, i.e. he does not care *when* the most drinks per hour were consumed, so he considers an horizon of, for example, 5.

Filtering of events. Second, only a selected set of events may be considered, i.e., the log is filtered and only the remaining events are used as an input for the recommendation engine. This abstraction is orthogonal to taking a partial case. Filtering may be used to remove certain activities or certain event types. For instance, if there are "start" and "complete" events for all activities in the event log ("A start" and "A complete"), then it is possible to only consider "start" events by filtering out other non relevant events.

When calculating the similarity between cases, events which have been filtered out are not considered, only the remaining events are compared. Let us assume that we have a case $\langle A_{start}, A_{complete}, B_{start}, B_{complete}\rangle$ and a partial case $\langle A_{complete}, B_{complete}\rangle$. Without removing the "start" events from the the first case a low similarity value results. However, by filtering out all "start" events, both cases are perfectly similar.

In our karaoke example, the log that is used by the recommendation service should be filtered to remove the "complete" events from the log. It would make no sense if a recommendation service suggests to Mary to stop singing. Hence, the recommendation service should only consider the starting of an activity when generating a recommendation.

Removing order and/or frequency. The third abstraction mechanism removes the order and/or frequency of activities from a case. For instance, in particular scenarios it might only be relevant to know which activities occurred, but not their ordering or multiplicity. In other cases, it might be relevant to know how many times activity A was executed or that A has been executed before B. Thus, this abstraction mechanism suggests the following three ways of representing cases:

- *sequence*, i.e., the order of events is considered for a particular case
- *multi-set of events*, i.e., the number of times each events is executed is considered, order is ignored
- *set of events*, i.e., the mere presence of events is taken into consideration

As the sequential approach allows to consider the ordering of activities this abstraction is most appropriate if a process is highly structured, i.e. the number of possible sequences is limited. However, since we are considering flexible systems where the number of possible sequences is typically extremely large, the multi set or set approach seems to be more promising. The main difference between these two is that loops (i.e. multiple executions of the same part of a process) are only considered in the multi set, but not in the set approach. Which abstraction to choose depends on the specific process that is being considered. In a hospital for example, no two patients ever follow the same path and therefore, the abstraction which considers the complete ordering of the complete case is not meaningful. For example an abstraction which shows the collection of treatments which a patient received in the last year would be more applicable.

Whether the ordering of activities has been removed or not has a direct impact on the similarity calculation. Assume that the similarity between a case $\langle A, A, C, D, E, A, F\rangle$ and a partial case $\langle A, A, D, G, H, I\rangle$ should be calculated. Taking the sequential approach the ordering of the activities is taken into account and the similarity value would only be zero or one. As in our example the partial case is not an exact prefix of the first case, the similarity is 0. Taking the multi-set or set approach the ordering of activities is not considered. The similarity is calculated as the number of activities which are contained in both cases divided by the number of activities in the partial case send to the recommendation service by the user. For instance, for the multi-set approach the similarity between the case $\{A, A, C, D, E, A, F\}$ and the partial case $\{A, A, D, G, H, I\}$ returns $\frac{3}{6}$ as a similarity value as the partial case contains 3 activities which are also contained in the first case. However, for the set approach the similarity is $\frac{2}{5}$, because each activity is only considered once.

Table 4. Comparison mechanisms for the karaoke bar.

Person	Infinite vs. Finite	Filtering	Order Frequency
Mary	Infinite	remove "complete" events	sequence
Bill	Finite (h=5)	remove "complete" events	multi-set
Peter	Infinite	remove "complete" events	set

In our karaoke example, the ordering of activities might only be relevant for Mary, e.g. there could be a policy that the same person cannot sing more than three songs in a row, in which case the recommendation service should take this into account for Mary. For Bill, the multi-set abstraction would work just as well, i.e. it is not important in which order drinks were consumed, only the multiplicity is. For Peter the set abstraction seems perfect, since he is only interested in the absence of any activity relating to drinking alcohol, hence he does not care if alcohol was drank once or more than once.

Table 4 summarizes the different comparison mechanisms that suit our karaoke example best, e.g. for Mary, recommendations should be based on similar cases, such that these similar cases are complete, with full ordering and without "complete" events.

In this section, we presented all the ingredients for the recommendation service, i.e. we presented (1) process logs, (2) comparison mechanisms and (3) target functions. In Section 4, we show a concrete implementation of the recommendation service in the process mining framework ProM. Furthermore, we show how access to the recommendation service is included in the flexible information system Declare. However, before we do so, we first revisit our karaoke example and show which recommendations are given to Mary, Bill and Peter when they step into the karaoke bar. For this, we use the settings of Table 4, the target functions of Table 3, and the log of Table 2.

When Mary enters the bar and asks for a recommendation for her first step, her history (which is empty) is a perfect match to all three cases. However, case 1 has a higher outcome for the target function and hance it is preferred over the other cases. The recommendation for Mary will therefore be to "start singing", like George did. For the others however, the recommendation is rather different.

For Peter (trying to stay sober) the set abstraction leads to the recommendation to either start listening or drinking juice, since that is what Jerry did and Jerry's example is the only one where the goal of staying sober is reached. Which activity to start first however is not of important in this context.

For Bill, the resulting recommendation is less straightforward. Since the target function eliminates the example set by Jerry, we focus on the other two. In George's case, he sang twice, listened, drank beer and drank juice. Susan listened twice, drank beer twice and drank wine. In both cases, the activities "drink beer" and "listen" are equally frequent. Therefore, the recommendation service cannot decide which one of these two is more relevant, thus both are proposed as the first step. The other activities are also proposed, but they are less frequent and/or their cases have a lower target function result and therefore

the quality attributes sent along with the recommendation show that they are probably not a good start.

4 Implementation

The recommendation service has been implemented in the process mining framework, ProM, i.e. ProM contains an analysis plugin that uses a (filtered) process log, a specific comparison mechanism and a specific target function. Note that the philosophy of ProM is to keep all components extendable with little effort. Therefore, we implemented the target functions and comparison mechanisms as plugins inside a plugin. To test the recommendation service implemented in ProM, we included the possibility to ask for a recommendations in the flexible system Declare [10].

We decided to use Declare as an information system and not for example YAWL, since Declare is based on a *declarative language*. Unlike imperative languages, declarative languages specify the "what" without determining of the "how". When working with such a model, the users are guided by the system to produce required results, while the manner in which the results are produced depends on the preferences of users. Therefore, declarative languages are more flexible by nature, but it is more likely that users working in such an environment need support.

4.1 Declare

Declare is a declarative workflow management system [10]. A process model in Declare consists of: 1) a set of activities and 2) *constraints*, which define the forbidden behaviour of the process. These constraints are provided to the user in an intuitive graphical notation and therefore these users do not have to be process experts to work with Declare models.

We will use an illustrative example to show how recommendations are used in Declare system. Consider a process model for the general practitioner medical process in Figure 4. It consists of five activities: 1) "check patient", i.e. the doctor examines the patient, 2) "prescribe medication", i.e. an appropriate medication is prescribed , 3) "check allergy", i.e. the existence of an allergy to a specific medication is checked, 4) "archive data", i.e. patient information is stored, and 5) "consult external", i.e. an external professional is consulted about exceptional situations. For details about these constraints, we refer to [10] and the annotations in Figure 4.

Figure 5 shows the Declare work-list with one open case of the doctor model from Figure 4. On the left hand side of the screen a case is selected, i.e. the active case. In the middle of the screen the whole process model for the active case is presented to the user. On the right side of the screen we can see that the system recommends to start activity "check allergy". This figure shows that recommendations are presented to users as additional information that increases system support during their work. By no means users are forced to follow

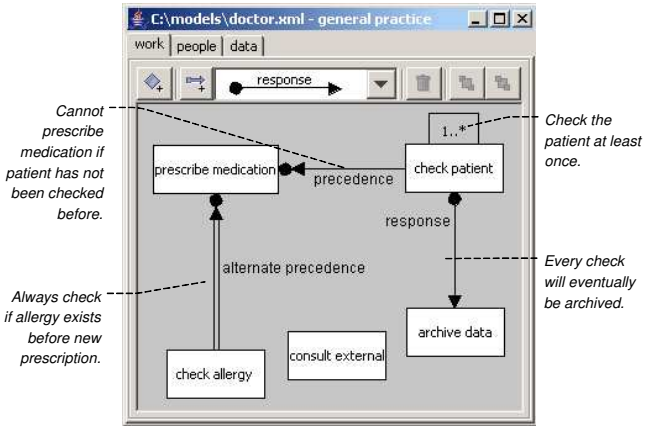

Fig. 4. General practitioner model in Declare.

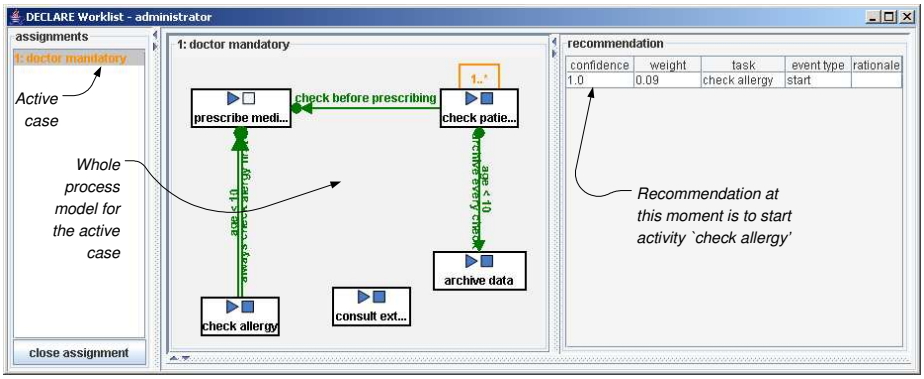

Fig. 5. General practitioner execution with recommendations in Declare.

recommendations. The user can freely chose the next action regardless of the recommendation, i.e., not following the recommendation is not considered to be a mistake or an error. Furthermore, Declare checks the incoming recommendations for consistency, i.e. if a certain activity is recommended but violates constraints imposed by the process model and consequently cannot be executed, it is not shown to the user.

4.2 ProM

The ProM framework has been developed as a workbench for process mining and related subjects, such as verification and analysis. In [3], the MXML log format was presented, which lies at the basis of the process mining algorithms contained in ProM. ProM however is not just a collection of process mining algorithms. Instead, it contains implementations of a large number of related subjects, such

as analysis or conversion algorithms. For more information about ProM, we refer to [4, 14] and the website http://www.processmining.org/.

An important feature of the ProM framework is that it allows for interaction between a large number of so-called plug-ins. A plug-in is basically the implementation of an algorithm that is of some use in the process mining area, where the implementation agrees with the framework. Such plug-ins can be added to the entire framework with relative ease: Once the plug-in is ready it can easily be added to the framework without the need for recompiling the framework.Note that there is no need to modify the ProM framework (e.g., recompiling the code) when adding new plug-ins, i.e., it is a truly "plug-able" environment.

For the implementation of our recommendation work, we extended ProM with an analysis plugin that in itself is extensible via plug-ins. This analysis plugin lets the user select the type of abstraction that needs to be used, as well as the specific implementation of the target function. Currently, several implementations of both are available.

When the right abstraction and target function have been selected, the analysis plugin opens a network connection for communication with a process engine, such as Declare. This enables the service to operate completely disconnected from any process engine, furthermore it allows for multiple process engines to connect at once. [3]

5 Conclusions and Future Work

The recommendation service presented in this paper touches on a whole new area of user support in flexible environments. Whereas traditionally, user support is negatively correlated to flexibility, we gave a first step to solve this problem. In our approach, users of a flexible information system can send their current execution history to the recommendation service. This service then considers all past executions of the same process and generates a recommendation to the user about the next action to take, taking into account the specific goals of that user.

Currently we are working on the experimental evaluation of the proposed recommendation service. We are conducting simulations evaluating the different comparison mechanisms and target functions. In addition, we plan to perform experiments with real users. Letting different groups of users work with the system with and without recommendations will allow us to make clear statements about the impact of recommendations on process performance (e.g., duration or business value). In addition, we plan to evaluate the effects of recommendations in respect to collaborative learning and examine whether users adjust their way of working when recommendations about better ways of working are presented to them.

[3] For more information about the implementation details, how to add abstractions or target functions and which messages are passed between ProM and Declare, we refer to [16]. This report contains all information needed to use ProM with any other flexible system, or to use Declare with any other implementation of a recommendation service.

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. Michael Adams, Arthur H. M. ter Hofstede, David Edmond, and Wil M.P. v. d. Aalst. A service-oriented implementation of dynamic flexibility in workflows. In *Coopis'06*, 2006.
3. B.F. van Dongen and W.M.P. van der Aalst. A meta model for process mining data. In Michele Missikoff and Antonio De Nicola, editors, *EMOI-INTEROP*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
4. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
5. P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: mediating between representation and action in workflow systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 190–198, New York, NY, USA, 1996. ACM Press.
6. C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. Change mining in adaptive process management systems. In *CoopIS'06*, pages 309–326, 2006.
7. P. Mangan and S. Sadiq. On building workflow models for flexible processes. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 103–109, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.
8. Mirjam Minor, Alexander Tartakovski, , and Ralph Bergmann. Representation and structure-based similarity assessment for agile workflows. In *ICCBR07*, 2007.
9. Pallas Athena. *Flower User Manual*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.
10. M Pesic and W.M.P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180, 2006.
11. M. Reichert and P. Dadam. ADEPT$_{flex}$ – supporting dynamic changes of workflows without losing control. *JIIS*, 10(2):93–129, 1998.
12. H. Reijers, J. Rigter, and W.M.P. van der Aalst. The Case Handling Case. *International Journal of Cooperative Information Systems*, 12(3):365–391, 2003.
13. Shazia Sadiq, Wasim Sadiq, and Maria Orlowska. A framework for constraint specification and validation in flexible workflows. *Information Systems*, 30(5):349 – 378, 2005.
14. H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM Framework. In *EMOI-INTEROP*, 2006.
15. J. Wainer and F. de Lima Bezerra. *Groupware: Design, Implementation, and Use*, volume 2806, chapter Constraint-Based Flexible Workflows, pages 151 – 158. Springer Berlin / Heidelberg, 2003.
16. B. Weber, B.F. van Dongen, M. Pesic, C.W. Guenther, and W.M.P. van der Aalst. The recommendation service interface description. Technical Report To Appear, Eindhoven University of Technology, 2007.
17. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling adaptive workflow management through conversational cbr. In *ECCBR'04*, pages 434–448, Madrid, 2004.
18. Barbara Weber, Stefanie Rinderle, and Manfred Reichert. Change patterns and change support features in process-aware information systems. In *Caise07*, 2007.