# Contextsensitive Online Adaption of Workflows

Johannes Kretzschmar, Clemens Beckstein

johannes.kretzschmar@uni-jena.de
clemens.beckstein@uni-jena.de
Friedrich Schiller University, Jena, Germany

**Abstract.** A dynamic and complex process environment forces the automated execution and monitoring of processes to face a lot of hard problems. This paper shows how the execution of agile workflows can be assisted by AI planning techniques. In contrast to previous approaches, this method allows for an online adaption of simple process models to changes in the process environment. It can handle a manifold of unexpected events and guarantees the soundness and completeness of the adapted workflow.

## 1 Introduction

Business processes today are commonly automated by formalizing and instantiating them to workflows. Established comprehensive industry standards support modeling, processing and monitoring of these workflow instances by a workflow management system [8]. The inclusion of more dynamic and complex processes and process environments, demands an extended support for agile workflows [17]. Agility allows the adaption of workflows to face a dynamic environment. The classical distinction between modeling and execution of workflows is progressively given up in order to adjust to changes and new demands at runtime. The adjusting procedure should be fully automated, efficient, correct and able to process expected as well as unexpected events. Workflow elements that are not affected by events should be executable during the adaption and not prolong the execution time of the underlaying business process. In order to guarantee the correctness of the adapted workflows a comprehensive formal semantic annotation of the process and its environment is needed. A more expressive model of business processes on the one hand and a formal process semantic on the other one suggest to combine AI planning techniques and workflow modeling for a solution of the adaption problem.

Here, we will present and discuss an approach to workflow adaption via plan repair. First, we will formally define how processes based on a simple plan model can fail, i.e. the corresponding failure model. Next, we will give a sketch and an example for adapting a plan by fragmentation and partial plan repair. Having related our work to previous approaches we will then conclude with a short summary and an outlook on future work.

## 2 Planning and Failing

For our approach, we assume a very basic workflow model. It consists of semantically annotated real world operations and allows parallel and sequential processing of (sub)workflows. This simple model can easily be transformed to a partial order plan and vice versa.

A partial order plan in set theoretically representation is a tuple $\pi = (A_\pi, \prec)$ consisting of a set $A_\pi$ of partially ordered actions [14]. The corresponding planning domain $\Sigma = (S, A, \gamma)$ is defined by a set of states $S \subseteq 2^L$ over a finite set $L$ of propositional symbols (so called fluents), a set of actions $A \supseteq A_\pi$ and a state-transition function $\gamma$, which defines the effect of an applied action. An action $a = (\text{precond(a)}, \text{effect}^+(a), \text{effect}^-(a)) \in A$ is applicable in a state $s \in S$, if precond(a) $\subseteq s$. The application of $a$ in $s$ results in a state $s' = \gamma(a, s) = (s \cup \text{effect}^+(a)) \backslash \text{effect}^-(a))$. In the following $\Gamma^*((A, \prec), s_0)$ denotes the set of states, which result from processing all linearizations of a plan $(A, \prec)$ in $s_0$. A linearization of $(A, \prec)$ is a sequential order of all actions $a \in A$ that is compatible with $\prec$. A partial order plan $\pi$ is a solution for a planning problem $\mathcal{P} = (\Sigma, s_0, g)$ with a set of goal fluents g, if for all $s_{end} \in \Gamma^*(\pi, s_0) : g \subseteq s_{end}$.

Although there is the assumption of a static world in classical planning, we have to handle the dynamics of real life scenarios as they are typical for workflow applications. Plan execution therefore has to deal with a wide range of events, which were not considered at the time of planning. These can affect all aspects of a plan and the planning domain: states, sets of actions and goals. In the following we assume that the actions of a partial order plan $\pi$ are processed one after the other. As a consequence, the execution state $s_{\pi'}$ that results from the linear plan fragment $\pi' \subseteq \pi$ of already executed actions is uniquely defined. An unexpected event can influence this state by implicitly adding or deleting fluents. Therefor we can define an (unexpected) external event as a pseudo action $\epsilon = (\emptyset, \text{effect}^+(\epsilon), \text{effect}^-(\epsilon))$ which transforms $s_{\pi'}$ to the new state $s' = \gamma(\epsilon, s_{\pi'})$. Internal unexpected events (planned actions that failed) can be formally treated in the same way. Process dynamics can also unexpectedly influence the set of actions available for planning thus changing $\Sigma$ to the new domain $\Sigma' = (S, A', \gamma')$. Our approach also allows for (unexpectedly) changing the goals from $g$ to $g'$ during plan execution. We assume, that an event or goal change always occur in between the processing of two actions and that the plan domain is fully observable. In real world scenarios unexpected events of the mentioned types can happen in combination.

Once an unexpected event $e_u$ is recognized in state $s'$ after execution of the plan fragment $\pi'$ its impact on the overall plan $\pi$ has to be assessed. If the remaining plan $\pi_R = \pi \setminus \pi'$ is not a solution for the planning problem $\mathcal{P}' = (\Sigma', s', g')$ the plan $\pi$ *failed*. One way to fix the plan would be to find a solution for $\mathcal{P}'$ that is to fix $\pi$ by re-planning. The fix we propose is to *repair* $\pi$ by isolating and partially replanning only that part of $\pi_R$ which is affected by $e_u$. A plan fragment is called *affected* by $e_u$, if it contains failed actions. An action $a \in \pi_R$ is failed, if $a \notin A'$ or if there is a linearization of $\pi_R$, where $a$ is not applicable.

## 3 Fragmentation and Replanning

In order to affect the plan or workflow execution as little as possible we identify minimal parts of $\pi_R$ that fail and try to find alternatives for them.

For this purpose we first choose an *inner fragment* $\pi_E = (A_E, \prec)$ of $\pi_r$ by selecting a connected plan fragment, which contains all failed actions $A_{\text{fail}} \subseteq A_E$. A plan fragment $\pi_E$ is called *connected*, if for all $a, b \in A_E$ and $c \in A_\pi$, we have that $c \in A_E$ if $a \prec c \prec b$. $A_E$ may also be empty if $\pi_R$ only fails due to an unexpected goal change. The *outer fragment* $\pi_F$ is the plan containing all actions, which are unordered to every action in $\pi_E$ and not contained in $\pi_E$. So $\pi_F$ is connected as well. Finally we define the *remaining fragment* $\pi_P$ as the plan fragment containing all actions $a \in A_R$ which are neither in the inner nor the outer fragment. By definition $A_E, A_F$ and $A_P$ are disjoint, but contain all actions of $\pi_R$ in union. Therefor we call $(\pi_E, \pi_F, \pi_P)$ a *fragmentation* of $\pi_R$ (see fig. 1).

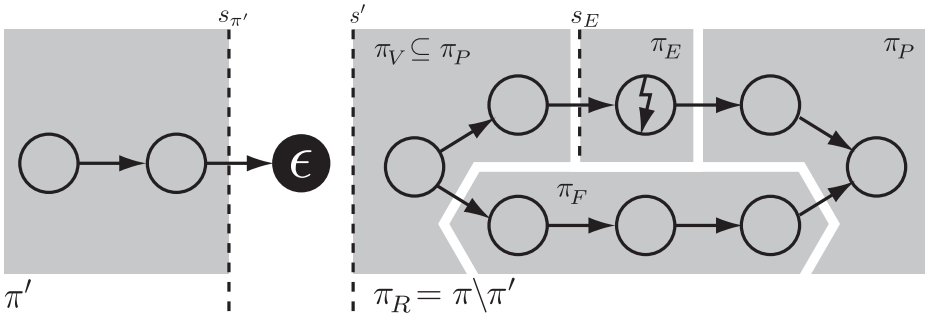


**Fig. 1.** a valid fragmentation of the remaining plan $\pi_R$

The plans $\pi_E$ and $\pi_F$ in a fragmentation $(\pi_E, \pi_F, \pi_P)$ are independent and $\pi_E$ contains all failed actions of $\pi$. The plan repair mechanism we propose searches for an alternative $\pi'_E$ for $\pi_E$ which is similar to the original, can be executed independently of $\pi_F$ and together with $\pi_F$ and $\pi_P$ is a solution of $\mathcal{P}'$. To ensure the independent (unordered) execution of $\pi_F$ in relation to $\pi'_E$, we introduce the so called context of $\pi_F$. The *context* $\mathcal{C}_F$ of $\pi_F$ is a pair $(\mathcal{H}_F, \mathcal{S}_F)$ of two fluent sets: the *soft criteria* $\mathcal{S}_F$ and the *hard criteria* $\mathcal{H}_F$. $\mathcal{S}_F$ contains all fluents which are used in preconditions or added, but not deleted by actions in $\pi_F$.

$$\mathcal{S}_F := \left\{ m : \bigvee_{a \in A_F} \bigwedge_{b \in A_F} \left(m \in \text{effect}^+(a) \cup \text{precond}(a)\right) \wedge \neg \left(m \in \text{effect}^-(b)\right) \right\}$$

The fluents in $\mathcal{S}_F$ can also be used, added but not deleted in $\pi'_E$ without resulting in plan flaws. On the other hand, fluents are not allowed to be added or used if they are deleted in $\pi_F$. These fluents make up $\mathcal{H}_F := \{m : \bigvee_{a \in A_F} m \in \text{effects}^-(a)\}$.

With the help of the context, it is possible to formulate an extended planning problem $\mathcal{P}_E = (\Sigma', s_E, g_E, \mathcal{C}_F)$. Compared to a classical planning problem like $\mathcal{P}$ an extended planning problem also takes into account the dynamic environment described by the context $\mathcal{C}_F$. As described in section 2, a failure may occur after the state where the unexpected event took place that triggered the plan repair. This happens for example, if a state change event affects the applicability of prospective actions in the plan. We therefore identify a fragment $\pi_V$ of $\pi_P$ that contains all actions $a \in A_P$ with $a \prec a_E$ for all $a_E \in A_E$. The plan fragment $\pi_V$ is used to produce the starting state $s_E$ for the extended planning problem $\mathcal{P}_E$ from the state $s'$ which contains those fluents that for sure hold once $\pi_V$ is completely executed: $s_E := \bigcap\{s_i : s_i \in \Gamma^*(\pi_V, s')\}$. Thus our method is able to repair parts of the plan which are far ahead of the current execution state without expanding the inner fragment. Finally, the goal $g_E$ is constructed from open preconditions in $\pi_P$ and the goal set $g'$. To simplify this procedure, the goal is represented by an implicit goal action $a_{goal} = (g', \emptyset, \emptyset)$ which is inserted into $\pi_P$ after all other actions $a \in A_P$. The goal $g_E$ for $\mathcal{P}_E$ is then chosen to:

$$\left\{ m \in L : \bigvee_{a \in A_P} m \in \text{precond}(a) \text{ and } \bigwedge_{b \in A_P \cup A_F} \left(b \prec_\pi a \rightarrow m \notin \text{effects}^+(b)\right) \right\}.$$

In order to handle extended plan problems the planning procedure uses a generalized definition of applicability which uses the context $\mathcal{C}_F = (\mathcal{H}_F, \mathcal{S}_F)$ to avoid flaws between the computed plan solution and $\pi_F$. An action a is applicable in a state s wrt. a context $\mathcal{C}_F$, if

$$\text{precond}(a) \subseteq s \text{ and } \bigwedge_{m \in \text{precond}(a)} m \notin \text{H}_F \text{ and } \bigwedge_{n \in \text{effects}^-(a)} n \notin \text{S}_F.$$

The parameters for the extended planning problem and the resulting applicability of actions are all determined by the selection of the inner plan fragment of $\pi_E$: the smaller $\pi_E$, the bigger $C_F$ and the more constrained is the planning process. For a good choice of $\pi_E$ we propose to first attempt the repair process with the smallest $\pi_E$ possible followed by iterative attempts with bigger fragments until $\pi_E = \pi_R$ which amount to classical replanning. It can be shown, that the solution $\pi'_E$ of $\mathcal{P}_E$ can replace the defective $\pi_E$ without producing flaws wrt. $\pi_F$. By processing $\pi'_E$ all open preconditions of $\pi_P$ and all (maybe changed) goal fluents will be satisfied: the repaired plan $\pi'_R$ is a sound and correct solution for the plan problem $\mathcal{P}'$ that summarizes the failure of the original plan.

## 4 A Real World Usecase

The generic repair method, introduced in this paper, was developed in the context of the MOPS project[1]. The target of this joint research project is adaptive

[1] MOPS is funded by the European Union (European Regional Development Fund) and the Federal State of Thuringia of Germany.

planning and secure execution of mobile processes for human agents in dynamic scenarios [1]. Because the processes in those scenarios typically are long living, a comprehensive event and failure model as well as support for process adaption during runtime is needed. The workflow technology of MOPS is based on BPEL descriptions. In order to bridge the expressive gap between the BPEL model and the plan model, the plan activities are encapsuled as semantically annotated services. As a consequence, workflows can contain complex control flow structures and detailled local event handling mechanisms, and still can be planned and manipulated automatically on an abstract level.

The application of MOPS is based on a generic scenario of service staff completing missions (on site repair jobs, delivery tasks, facility management). In order to exemplify our repair method, let us assume an extremely reduced scenario with 3 service technicians and 3 missions that can be represented by the following simple set-theoretic planning domain with the fluent set $L = \{a1, ..., a3, done1, ..., done3, 1doing1, ..., 3doing3\}$. For every technician, there is a fluent $aX$, which means "technician $X$ is available". Likewise there is a fluent $doneX$ for every job, which means "job $X$ is successfully accomplished." For every job and every technician there is a fluent $XdoingY$, which means "technician $X$ is assigned to job $Y$". The planning domain further contains two types of actions. The first one actually assigns a job to a technician, if the technician is available. $assign1to2 = (\{a1\}, \{1doing2\}, \{a1\})$, e.g., allocates job2 to technician1. The second type of action initializes the execution of a job by its assigned technician. $1do2 = (\{1doing2\}, \{done2, a1\}, \{1doing2\})$, e.g., means "technician 1 is doing job 2". The solution of the planning problem $\mathcal{P} = (\Sigma, s_0, \{done1, done2, done3\})$ with $s_0 = \{a1, a2, a3\}$ then results in a plan like $\pi$ as shown in fig. 2.

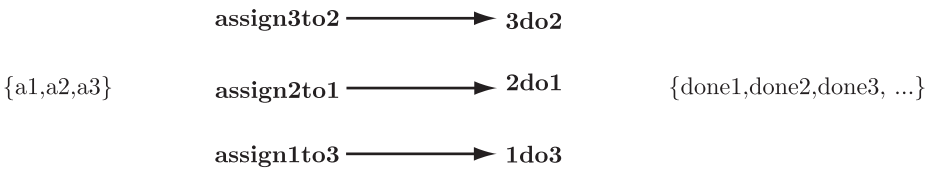


**Fig. 2.** a sound and correct solution $\pi$ for $\mathcal{P}$

Let us now assume that the current execution state $s_\pi = s_0$ and that in this state a technician reports to be sick resulting in the new actual state $s' = \gamma((\emptyset, \emptyset, \{a3\}), s_\pi) = \{a1, a2\}$. This event leads to a failure of the remaining plan $\pi'$, because action *assign3to2* is no longer applicable. Because this action is the only action affected by the failure we can reasonably set $A_E = \{assign3to2\}$, which implies $A_F = \{assign1to3, 1do3, assign2to1, 2do1\}$ and $A_P = \{3do2\}$. This fragmentation leads to a context $\mathcal{C}_F$ with $\mathcal{H}_F = \{a1, a2, 1doing3, 2doing1\}$ and $\mathcal{S}_F = \{done3, done1\}$) and the extended planning problem $\mathcal{P}_E = (\Sigma, \emptyset, g_E, \mathcal{C}_F)$ with the open precondition of *3do2* as goal $g_E = \{3doing2\}$ It is impossible to find a solution for this problem because job

2 can not be assigned to any other technician due to $\{a1, a2\} \subset \mathcal{H}_F$ (an assignment action deletes the availability fluent for its agent and therefore is not applicable as long as the fluent is contained in the context). If we now widen the inner fragment to $A_E = \{assign3to2, 3do2\}$ (hence $A_P = \emptyset$) then the corresponding new extended planning problem $\mathcal{P}_E = (\Sigma, \emptyset, \{done2\}, \mathcal{C}_F)$ still has no solution because $A_F$ and $\mathcal{C}_F$ remained the same. A further widening of the inner fragment to $A_E = \{assign3to2, 3do2, 2do1\}$ ($A_P = A_V = \{assign2to1\}$) diminishes the outer fragment to $A_F = \{assign1to3, 1do3\}$ and the context to $\mathcal{S}_F = \{done3\}$ and $\mathcal{H}_F = \{a1, 1doing3\}$. The resulting extended planning problem $\mathcal{P}_E = (\Sigma, \{2doing1\}, \{done1, done2\}, \mathcal{C}_F)$ is now solvable with the plan alternative $\pi'_R$ as shown in fig. 3

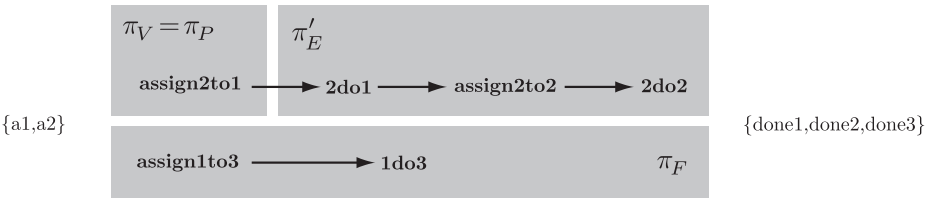


**Fig. 3.** the adapted plan $\pi'_R$ with alternative $\pi'_E$ and corresponding fragmentation

This little example already shows that the choice of the inner fragment $A_E$ significantly influences the context and therefore the degrees of freedom for the replanning problem. For this choice we propose to use the control-flow structure, respectively the partial plan order: in order to maximize the part of the workflow that can be executed concurrently to the adaption process for the failed part, the outer fragment $A_F$ and $A_V$ of the remaining fragment should stay as big as possible.

## 5 Related Work

A lot of workflow related approaches for adaption while runtime are based on the reusability of pre-modeled workflow fragments. AdaptFlow [6], which is based on ADEPT$_{flex}$ [16], uses explicit event-condition-action (ECA) rules to change a workflow. A similar, and widely-established approach, based on cased-based reasoning (CBR) [12], is implemented in CBRflow [18], Phala [10] and CAKE II [13]. In case of a change request, a pre-modeled workflow fragment is chosen from a case repository and integrated into the workflow. The retrieval is based on a similarity measure of the structural, procedural and declarative knowledge. The CBR related methods differ primarily in the type and complexity of the underlying knowledge model. There are also approaches using AI planning techniques like the traditional cased-based planner CHEF [7], which can be applied to simple workflow models. CODAW [11], e.g., is using a case repository implemented as a hierarchical task network. Other workflow specific approaches like

BPEL'n'Aspects [9] or StPowla [5] use policy descriptions for implementing adaptive process logic. These descriptions are similar to the pre-modeled knowledge of CBR cases and likewise triggered by ECA rules. In contrast to our approach, all these approaches lack of a specified failure model and a goal oriented adaption. The identification of events and the adaption is restricted to pre-modeled use cases which do not cover unexpected events. Further, it is impossible to recognize and handle any flaws which may occur by the adaption.

Besides case based approaches there are two strategies in AI planning for performing a plan adaption: re-planning from scratch or plan repair. Nebel and Koehler [15] show, that the local adaption and reuse of plans suffers from the inherent structural worst case complexity of planning: failed plans should be re-planned from scratch because in the worst case, the whole plan has to be restructured anyway. First, this consideration ignores the necessity of concurrent plan execution and plan adaption as it is typical for, e.g., business processes. Further, one of the few general domain independent approaches for plan repair, as discussed by Arangu et. al. in [2] and used in the planner MIPS-XXL [3], performs great using local adaption techniques: an iterative two-step procedure identifies the defective part of the plan and tries to find an alternative by gradually expanding the planning problem, especially the set of planning operators. But in contrast to our approach the scope of plan repair is fixed.

Usually there is no unique solution for a planning problem, thats why alternatives have to be considered. Fox et. al, e.g., discuss in [4] a measure for plan-similarity like those used in CBR: a heuristic assures that the alternative plan is as close as possible to the original failed plan. This is justified by the assumption that the original plan was modeled by or with the help of domain experts. Especially when using plans as workflow models, this measure also permits maintaining commitments between process partners. Our context description of process (fragments) could supply useful informations for a heuristic or similarity measure in first principle planners as well as CBR or CBP approaches.

## 6 Conclusion and Future Work

We have introduced a domain independent method for adapting partial order plans. In contrast to present approaches, this method follows a formally defined complex failure model that captures the relevant characteristics of the underlying process model. The approach is able to adapt processes at runtime, i.e. concurrently to process execution, and is guaranteed to produce sound and correct solutions.

At the moment there still is a big gap between the expressive power of the plan representation we used and that of state of the art workflow models but we are working hard to generalize our method to more complex representations, which are able to represent additional aspects of todays workflow languages — among them complex control structures as well as explicitly coded data flow, organizational and security aspects.

## References


1. MOPS — project description (2011), http://mops.uni-jena.de/us/Homepage-page-.html, [Online; accessed 08-February-2012]
2. Arangu, M., Garrido, A., Onaindia, E.: A general technique for plan repair. In: Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on. vol. 1, pp. 515 –518 (nov 2008)
3. Edelkamp, S., Jabbar, S., Nazih, M.: Large-scale optimal pddl3 planning with mips-xxl. In: 5th International Planning Competition Booklet (IPC-2006) (2006)
4. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: In Proc. ICAPS. pp. 212–221. AAAI Press (2006)
5. Gorton, S., Montangero, C., Reiff-Marganiec, S., Semini, L.: Service-oriented computing - icsoc 2007 workshops. chap. StPowla: SOA, Policies and Workflows, pp. 351–362. Springer-Verlag, Berlin, Heidelberg (2009)
6. Greiner, U., Ramsch, J., Heller, B., Löffler, M., Müller, R., Rahm, E.: Adaptive guideline-based treatment workflows with adaptflow. In: Proceedings of the Symposium on Computerized Guidelines and Protocols (CGP 2004), Computer-based Support for Clinical Guidelines and Protocols. pp. 113–117. IOS Press (2004)
7. Hammond, K.: Case-based planning: A framework for planning from experience. Cognitive Science 14, 385–443 (1990)
8. Jablonski, S. (ed.): Workflow-Management. dpunkt-Verl., Heidelberg (1997)
9. Karastoyanova, D., Leymann, F.: BPEL'n'Aspects: Adapting Service Orchestration Logic. In: Proceedings of 7th International Conference on Web Services ICWS 2009. pp. 222–229. IEEE Computer Society (2009)
10. Leake, D., Kendall-Morwick, J.: Towards case-based support for e-science workflow generation by mining provenance. In: Althoff, K.D., Bergmann, R., Minor, M., Hanft, A. (eds.) Advances in Case-Based Reasoning, Lecture Notes in Computer Science, vol. 5239, pp. 269–283. Springer Berlin / Heidelberg (2008)
11. Madhusudan, T., Zhao, J.L., Marshall, B.: A case-based reasoning framework for workflow model management. Data Knowl. Eng. 50, 87–115 (July 2004)
12. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: Montani, S., Bichindaritz, I. (eds.) Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010, Alessandria, Italy, July 19-22, 2010. Proceedings. pp. 421–435. LNAI 6176, Springer (2010)
13. Minor, M., Schmalen, D., Kempin, S.: Demonstration of the agile workflow management system cake ii based on long-term office workflows. In: BPM (Demos)'09 (2009)
14. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
15. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. Artificial Intelligence 76, 427–454 (1995)
16. Reichert, M., Dadam, P.: Adept flex - supporting dynamic changes of workflows without loosing control. Journal of Intelligent Information Systems 10, 93–129 (1998)
17. Weber, B., Wild, W.: Towards the agile management of business processes. In: Althoff, K.D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T. (eds.) Professional Knowledge Management, Lecture Notes in Computer Science, vol. 3782, pp. 409–419. Springer Berlin / Heidelberg (2005)
18. Weber, B., Wild, W., Breu, R.: Cbrflow: Enabling adaptive workflow management through conversational case-based reasoning. In: ECCBR. pp. 434–448 (2004)