Controlled Flexibility in Workflow Management*

Justus Klingemann

German National Research Center for Information Technology (GMD)
Integrated Publication and Information Systems Institute (GMD-IPSI)
Dolivostraße 15, D-64293 Darmstadt, Germany
`klingem@darmstadt.gmd.de`

Abstract. Traditional workflow models are centered around the structure of a workflow. Other requirements and goals like time and cost can be represented only indirectly. As a result, when the workflow is executed, the WfMS can only enforce the structural constraints. This becomes a problem when the workflow deviates from its normal execution, e.g., due to errors or delays.
In this paper we propose an approach to extend the definition of workflows which allows to make the underlying goals explicit. In addition, we introduce flexible elements into the workflow specification. We make use of these flexible elements in a controlled way to achieve a balanced fulfillment of all goals – structural as well as non-structural – under various runtime conditions.

1 Introduction

Workflow management is a fast evolving technology which is increasingly being exploited by businesses in a variety of industries [1,2,3,4]. Its primary mission is to handle business processes. A *workflow* is the automation of a business process. Such a business process has certain goals. These are often multidimensional: Certain tasks have to be performed, a certain quality has to be achieved, the business process or at least certain activities have to be performed in a certain timeframe and the process shall not exceed a certain cost. To allow the system to support the goal fulfillment, these goals have to be made explicit. In addition, a sufficient degree of freedom is necessary to actively control the workflow execution towards an optimized goal fulfillment. To create this freedom, we integrate execution alternatives in form of flexible elements into the workflow structure. This approach to flexibility is especially useful for production workflows as well as cross-organizational workflows in which manual ad hoc interventions are undesirable as they raise additional organizational problems like who is responsible for a new activity or how they affect the autonomy of the involved organizations. System support for the adaptation at runtime is also beneficial when the effects of possible execution alternatives on the goals are too complex to allow a user to perform these adaptations interactively.

* This work has been partially supported by the ESPRIT project CrossFlow (www.crossflow.org). However, it represents the view of the author.

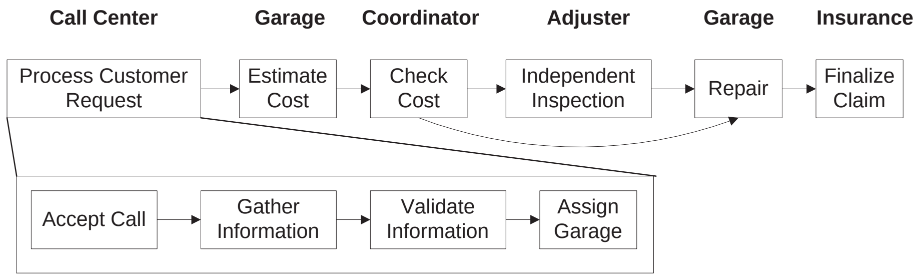

Fig. 1. A cross-organizational workflow to process a motor damage claim

To motivate our approach we will use an example from the insurance domain. This application has been analyzed in the CrossFlow project [5]. Figure 1 shows a simplified workflow used to process a motor damage claim. The workflow is started with the processing of the customer request which is performed by a call center. Like each activity in this cross-organizational workflow, this is a subworkflow on its own. The processing of the customer request for example consists of four activities. In the first activity the call is accepted. Then, the call center gathers information about the accident, the customer and other involved persons. In the next activity, this information is validated. For example, it is checked, whether the customer is properly insured to use this service. In the final activity, the customer is advised about garages which are located nearby. After the processing of the customer request, the garage which has been chosen by the customer becomes active. It has to assess the estimated repair costs. This cost estimate is then checked by an organization which is specialized in the coordination of the processing of motor damage claims. If the estimated costs are above a certain threshold, an independent adjuster is appointed to inspect the damaged car and to re-assess the repair costs. When the independent inspection is finished or it needs not be performed, the garage repairs the car. As a last step in the workflow, the insurance performs some administrative tasks and thus finalizes the claim.

The business goals are not exclusively captured by the sequence of steps represented in Figure 1. In the case of an insurance company an important business goal is to keep the customer satisfied. The processing of an insurance claim is one of the rare occasions where customers are able to judge the performance of their insurance. Therefore, the insurance has to demonstrate that it handles the claim of the customer efficiently. With respect to the workflow execution, this translates to the requirement that the claim processing has to be finished within five days.

Statically, this requirement can be taken into account by estimating the execution times of all activities and making sure that on all execution paths the sum of the estimated execution times is less than five days. However, in this approach the deadline is only represented implicitly. As a result, when unexpected events occur only the structural constraints can be enforced, i.e., a workflow management system (WfMS) makes sure that all activities are executed in the prescribed order even though the deadline may be violated. Therefore, we call a workflow *multidimensional* if it allows to represent explicitly non-structural goals along other dimensions like time or costs.

The motor damage claim process is a typical example. On the one hand, the processing of the claim should be cost efficient. On the other hand, it is important for the insurance to execute the process within five days to keep the customer satisfied. A cost efficient execution implies that in case of a more expensive damage an adjuster is brought into play who inspects the car. Under normal conditions enough time is available for this activity within the desired timeframe of five days. However, if a delay occurs, e.g., due to communication errors among the call center, the insurance and the coordinator, this might not be the case. As a consequence, the runtime priorities can change. To maintain the customer's satisfaction, the insurance might be willing to drop activities like the inspection of the car by an adjuster and instead make sure that the car is repaired in time.

This decision also depends on the parameters. For example, it might be desirable to adapt the workflow if the estimated repair cost is only a little bit in excess of the threshold and on the other hand the risk of failing to repair the car in time is high if the repair is delayed until the inspection is finished. However, if the repair cost is very high and the delay only moderate so that there is a good chance to finish the repair in time, it might be preferable to execute the workflow as planned.

To increase the flexibility and allow an adaptation of the workflow to the conditions at runtime the specification of different execution options is necessary. This includes different aspects. First, alternative execution structures have to be specified. Our approach to do this is described in Section 2. As a second part, the desired quality of service (QoS) goals have to be provided. They allow to compare the utility of the different execution options. The specification of QoS goals and their relationship to QoS parameters is described in Section 3. In Section 4 we give a complete example how our approach is applied to enhance a workflow specification and adapt its execution. Afterwards, we give an overview of the related work and finally some conclusions.

2 Making the Workflow Structure Flexible

To enable different execution options we have to relax the workflow structure. We consider the workflow structure as consisting of two parts. The first part is mandatory. It consists of activities which have to be executed (if the respective enabling conditions evaluate to true) for the workflow to be successful, as well as ordering relationships which have to be obeyed. A second part consists of execution alternatives which can be selected depending on requirements at runtime.

To specify the structure of a workflow, a set of constructors like sequence, or-split, or-join, and-split and and-join has emerged which forms the basis of most workflow models [6]. We use these constructors to specify the overall structure of the workflow and its mandatory parts. To specify execution alternatives, we use so called *flexible elements.* A flexible element (FE) is a construct that represents different execution alternatives, i.e., it is a set of different subworkflows. We identify three kinds of flexible elements namely alternative activities, non-vital activities, and optional execution orders.

Alternative Activities This flexible element allows to represent different trade-offs with respect to the goals of the workflow, e.g., a high-quality option which requires a considerable execution time versus a quicker option which produces results with a lower quality. This provides a way to switch among these alternatives depending on the priorities at runtime. It is represented by a list of activities in square brackets. At runtime exactly one of them will be chosen for execution.

Non-vital Activity This flexible element specifies that an activity can either be executed or omitted, i.e., it is non-vital. Activities are non-vital if they are beneficial under certain conditions but should better be omitted in other situations. Non-vital activities are represented by adding the tag *nv* to the activity. Non-vital activities can be considered as a special case of alternative activities, if they are modelled as the alternative between the activity itself and an empty activity $\emptyset$.

Optional Execution Order This flexible element can be used if activities should be re-ordered to expedite the execution of urgent activities. It allows to specify a sequence of activities in a specific order which can be considered as a preferred or default order. However, this order is not mandatory and the activities can alternatively be executed without any restriction on their order, i.e., in parallel. This flexible element is specified by a list of activities in braces.

We call a traditional activity, i.e., a unit of work, an *elementary activity.* A *flexible workflow (FWF)* is a workflow in which each activity is either an elementary activity or a flexible element. If no ambiguity occurs, we will use the terms activity and elementary activity interchangeably.

Note, that execution alternatives in flexible elements are different from or-split/join structures. The decision which successor of an or-split is executed is made based on the evaluation of a predicate and cannot be influenced. In contrast to this, each choice among the alternatives of a flexible element is considered as correct and the decision can be made based on global goals of the workflow.

To relate a flexible workflow to the possible traditional workflows that can be derived from it, we define some additional terms. For a flexible element fe a *resolution* is a function that maps fe to one specific alternative. For a flexible workflow a resolution r maps each flexible element to a specific alternative, i.e., each flexible element fe is replaced by $r(fe)$. Let fwf be a FWF and r a resolution function for fwf then we call $rfwf = r(fwf)$ a *resolved flexible workflow (RFWF)* of fwf. With $R(fwf)$ we denote the set of all RFWF of fwf.

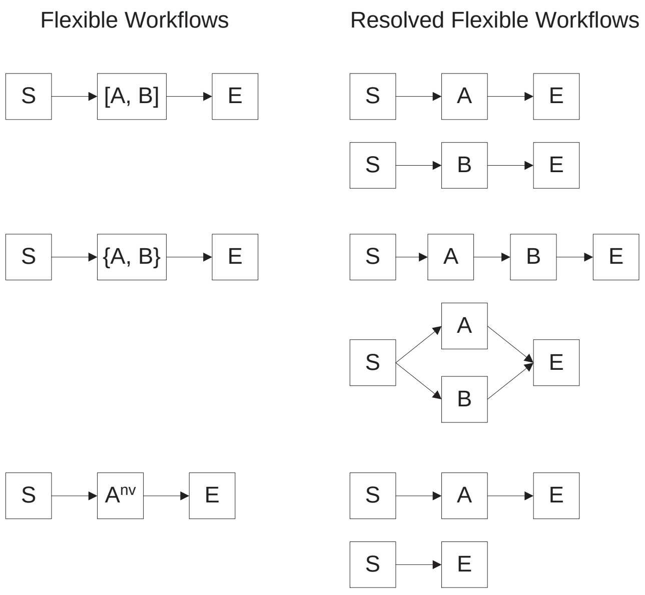

Fig. 2. Examples for flexible workflows and their resolved flexible workflows

Figure 2 shows examples of different flexible workflows and their corresponding RFWFs. In this figure S, E, A, B are elementary activities. The first FWF specifies that all RFWF have to start with S and end with E and in between either A or B is executed. The second example shows an optional execution order, i.e., either A is executed before B or both activities can be executed in parallel. The last example contains the non-vital activity A.

3 Enhancing a Flexible Workflow with QoS Goals

In this section we will explain our approach to model QoS goals and how they are used to enhance a workflow specification. The basic building block of a QoS goal is a QoS parameter. A dimension on which QoS parameters can be defined is called *QoS dimension*. Examples are time, cost or application specific dimensions like the accuracy of a specific result. A *QoS parameter* is a specific quantity defined on a QoS dimension. Examples for QoS parameters on the dimension time are the duration of an activity or the duration of the overall workflow. As QoS parameters we consider any property that can be assigned to or derived from a partial or complete workflow instance. A *QoS goal* is an optimization criterion that is assigned to a QoS parameter. To allow the formulation of QoS goals we will in the following give a classification of QoS parameters.

3.1 Classification of QoS Parameters

Depending on how they are obtained we distinguish three kinds of QoS parameters.

Elementary QoS Parameters As elementary QoS parameters we define parameters which can be directly observed, i.e., monitored, from a running workflow instance. Examples are the output parameters of activities, the current time, and the execution time or cost of terminated activities. Elementary QoS parameters form the basis for the measurement of QoS goals.

Derived QoS Parameters Derived QoS parameters are functions of other QoS parameters, i.e., if qos_0 is a derived QoS parameter, then there exists a user-defined function f and a set of QoS parameters $qos_1, \ldots, qos_n$ with $qos_0 = f(qos_1, \ldots, qos_n)$. An example for a derived QoS parameter in the insurance scenario is the expected cost saving by appointing an assessor depending on the repair cost estimated by the garage. A special case of derived QoS parameters are 0-ary functions, i.e., constants. Constants can be considered as external parameters. An example are the costs of the assessor which are needed to determine the total cost savings by appointing an assessor.

Predicted QoS Parameters Predicted QoS parameters are quantities which make assertions about the future behavior of a workflow. Predicted QoS parameters require a prediction function. Each prediction is related to a resolved flexible workflow, i.e., a workflow for which decisions for each flexible element have been made.

The dependency relationship among QoS parameters can be described as a graph. We require that this graph does not contain any cycles. For example, there must not be the dependency $qos_1 = f_1(qos_2)$ and at the same time $qos_2 = f_2(qos_1)$. Each of the nodes in this graph can be assigned a QoS goal. Usually, a QoS goal will be assigned to the root of a subgraph that forms a tree.

3.2 Specification of QoS Parameters

The necessary specifications depend on the respective class of QoS parameters. For elementary QoS parameters we have to make sure that the relevant parameters are monitored. This is specified by means of a *monitoring condition* that defines the parameter that has to be monitored as well as the activity to which it belongs.

For the specification of derived QoS parameters we have to provide the necessary function as well as the list of parameters.

To allow for predictions, we have to specify a prediction function. The structure, complexity, and specification effort for a prediction function heavily depends on the QoS parameter to be predicted, the complexity of the used prediction model and the available information about the behavior of the workflow. If we want to predict the execution time of a workflow or a part of it, we can model the statistical behavior of the workflow, i.e., the activities and the control flow conditions. This means that we have to provide probabilities for the activation of control flow edges originating from an OR-split and information about the duration of activities. Depending on the desired accuracy, we can model the duration of activities just by means of the expected duration or provide a complete distribution function.

Instead of explicitly specifying the prediction function, we can alternatively derive a prediction function from the observed behavior of earlier workflow executions. In this case, we assume that the future behavior will be similar to the observed behavior in the past. We call the use of existing workflow logs to derive a prediction function *offline monitoring*. An approach for offline monitoring based on continuous-time Markov chains has been described in [7].

3.3 Specification of QoS Goals

We define a QoS goal as a triple $g = (q, f, w)$ where

1. q is a QoS parameter as defined in the previous section,
2. f is a fulfillment function,
3. w is a weight.

The fulfillment of a goal is specified by means of a function f : QoS parameter $\rightarrow [0, 1]$. The fulfillment describes whether the goal is satisfied or not. For a continuous function it describes the expected degree of fulfillment.

In the following we focus on two classes of QoS goals called *extremity goals* and *range constraints*. A range constraint aims at keeping a QoS parameter within a boundary. For notational simplicity we will interpret a boolean condition containing a QoS parameter as a function of this QoS parameter. In the deterministic case the boolean value *true* is mapped to 1 whereas the boolean value *false* is mapped to 0. Thus, an example of a fulfillment function for the QoS parameter *total execution time* is

$$\text{total execution time} < \text{deadline}.$$

This boolean condition is a shorthand for the function

$$f(\text{total execution time}) = \begin{cases} 1 & \text{if total execution time} < \text{deadline} \\ 0 & \text{else.} \end{cases}$$

In the stochastic case the boolean condition represents the expected value of this function. This is equivalent to the probability that the boolean condition is evaluated to *true*.

An extremity goal aims at maximizing or minimizing the QoS parameter. Usually the fulfillment function will be a linear mapping from the range of values which are expected in real-world applications into the interval $[0, 1]$. For example consider the minimization of the *total execution cost*. If we expect that the costs are always positive and can never exceed 100, then a fulfillment function f could be

$$f(\text{total execution cost}) = 1 - \frac{\text{total execution cost}}{100}.$$

Note, that values closer to 1 always indicate a higher fulfillment of the goal.

The weight of a goal is used to indicate the relative importance of the goals. It can be an arbitrary positive number.

3.4 Specification and Optimization of Multidimensional Workflows

We now have the ingredients to define a multidimensional workflow as a flexible workflow which is enhanced with a set of QoS goals. We formally define a *multi-dimensional workflow (MDWF)* as an ordered pair (fwf, g) where fwf is a flexible workflow and $g = \{g_1, \ldots, g_n\}$ is a set of QoS goals. We extend definitions for FWFs in a straightforward way to MDWF by applying them to the FWF part of a MDWF.

The relevant *runtime conditions* of a workflow execution are represented in our model by the values of monitored elementary QoS parameters and the values of predicted QoS parameters. For each runtime condition and each choice for the different structural alternatives, i.e., each resolved MDWF *rwf* we can calculate the fulfillment of each goal. Using this information we can define the *total utility* as the weighted sum of the goal fulfillment:

$$\text{total utility}(\mathit{rwf}) := \sum_{g_i \in g} \text{weight}(g_i) * \text{fulfillment}(g_i)$$

The total utility is calculated for a resolved MDWF, i.e., a workflow for which decisions for each flexible element have been made. To execute a workflow with the maximal total utility possible for specific runtime conditions, we therefore have to make the appropriate choice among the alternatives of the flexible elements. Thus, to optimize the execution of a MDWF *mdwf* we have to find a resolved MDWF *rwf* with

$$\forall \mathit{rwf}_i \in R(\mathit{mdwf}) : \text{total utility}(\mathit{rwf}_i) \leq \text{total utility}(\mathit{rwf}).$$

A straight forward optimization algorithm is therefore to enumerate all possible resolved FWFs which result from a flexible workflow specification, calculate their total utilities and choose the flexibility alternatives according to the resolved flexible workflow with the highest utility. To adapt to possible changes during the workflow execution, this algorithm is executed each time the execution thread encounters a flexible element. More efficient optimization strategies have to depend on the QoS parameters subject to goals and the structure of the workflows. One example for such an optimization algorithm which is used to optimize a temporal and a utility goal is described in [8].

4 An Example for the Specification and Enforcement of Multidimensional Workflows

In the following, we consider the simplified workflow for the motor damage claim scenario described in Section 1. Using the primitives presented in Section 2, the flexible workflow structure can be specified as shown in Figure 3.

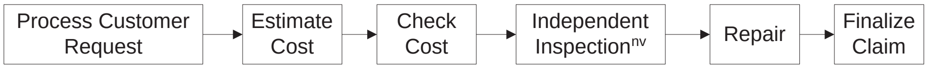

Fig. 3. The motor damage claim process as a flexible workflow

This workflow has exactly one flexible element, namely the non-vital activity Independent Inspection. Therefore, we have to execute the decision algorithm exactly once, namely immediately after the activity Check Cost has terminated. In the remainder of this section we will describe how the goals of this workflow can be modelled and how these goals influence the execution of the workflow.

4.1 Use of Deterministic Predictions

Informally, the first goal of the workflow is to minimize the cost. For the motor damage claim, this translates to the requirement to appoint an assessor to perform an independent inspection if this activity can save cost. The overall cost savings of the independent inspection again depend on the expected reduction of the repair cost as a result of the inspection, and the cost of the inspection itself. The calculation of the total savings is therefore performed as follows. As elementary QoS parameter we need the estimated repair cost provided by the garage as a result of the activity Check Cost. Hence, we have to define an appropriate monitoring condition:

$$\text{Monitor(Check Cost, estimated repair cost)}$$

To derive the reduction of the repair cost due to the inspection, we specify an appropriate function. In our example we use

$$\text{repair savings(estimated repair cost)} := \frac{\text{estimated repair cost}}{10}.$$

To calculate the total savings we again specify a derivation function and in addition to this provide an external parameter which describes the cost of the assessment itself:

$\text{cost assessment} := 50$

$$\text{total savings(estimated repair cost)} :=$$
$$\begin{cases} \text{repair savings(estimated repair cost)} \\ -\text{cost assessment} & \text{if Independent Inspection executed} \\ 0 & \text{else} \end{cases}$$

Since we want to specify a goal on the parameter total savings, we have to define an appropriate fulfillment function. Assume that all repair costs are in [0, 10000]. Then we can define

$$\text{fulfillment}_1(\text{total savings}) := \frac{\text{total savings} + \text{cost assessment}}{1000}.$$

Note, that total savings could be negative, if the cost of the assessment exceeds the savings in repair cost. To map the fulfillment into the interval [0, 1], we therefore add the cost assessment. We further would like to give this goal a weight of 10. This allows us to define the first goal as

$$\text{goal}_1 := (\text{total savings}, \text{fulfillment}_1, 10).$$

Our second goal is to execute the workflow within five days. To check this range constraint we need information about the execution time that has already been elapsed since the start of the workflow. To obtain this information we specify the following monitoring condition:

$$\text{Monitor(Check Cost, time until end of)}$$

To get information about the remaining time until the end of the workflow, we need a prediction function. The signature of such a function for the prediction of the length of the time interval between the end of the activity Check Cost and the end of the workflow would look as follows:

$$\text{remaining time} := \text{Predict(resolved workflow, end, Check Cost, end, workflow)}$$

Assume that we have provided a prediction function which uses a deterministic model for the activity duration. For this example we further specify the execution time of activities used by the prediction function as duration(Independent Inspection) = 1 day, duration(Repair) = 2 days, and duration(Finalize Claim) = 0.2 days. As a result, the prediction function would return a duration of 1 + 2 + 0.2 days = 3.2 days for the execution option including the inspection activity and 2 + 0.2 days = 2.2 days for the execution option without the inspection activity.

The monitored information about the elapsed execution time together with the predicted information about the remaining execution time allows us to specify the derived QoS parameter total execution time as the sum of remaining time and the time until end of the activity Check Cost. We can now specify the fulfillment function as

$$\text{fulfillment}_2(\text{total execution time}) := \text{total execution time} < 5 \text{ days}$$

and the second goal (using a weight of 2) as

$$\text{goal}_2 := (\text{total execution time}, \text{fulfillment}_2, 2).$$

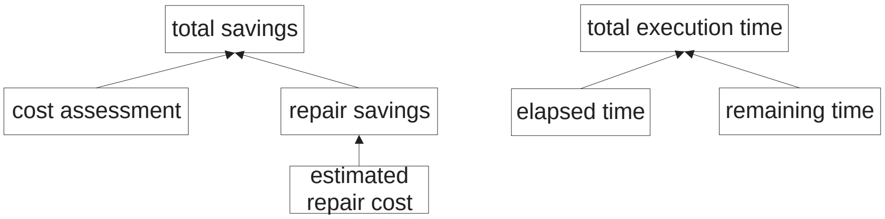

Fig. 4. The relationship of QoS parameters

Figure 4 shows the two trees that result from the dependency relationship of the QoS parameters.

Now we will consider the different decisions depending on the parameters. First, consider the total savings. For a workflow without an inspection, the fulfillment of this QoS parameter always has the value of 0.05. For a workflow which performs an independent inspection, the value of this QoS parameter linearly increases with the repair cost. For a repair cost below 500, its fulfillment value is below 0.05. In this case, the inspection will always be omitted as its contribution to the total utility is inferior to the option to omit it. For repair costs above 500, we have to make a trade-off with the second goal.

Considering the second goal, we see that the goal is always fulfilled, if the elapsed execution time at the end of the activity Check Cost is less than 1.8 days. In this case, both execution options make sure that the workflow is finished in time. On the other hand, if the elapsed time is more than 2.8 days, none of the execution options will allow the workflow to be finished in time. As a result, for these two cases, the fulfillment of the second goal is independent from the chosen structural alternative. The interesting case occurs if the elapsed time is between 1.8 and 2.8 days. In this case, the execution of the inspection results in missing the deadline whereas omitting the inspection allows to keep the deadline and thus, increases the total utility by 2. Therefore, the decision depends on the estimated repair cost. From the calculation of $goal_1$ we see that the increase of the contribution to the total utility resulting from the execution of the inspection is equal to 2 for an estimated repair cost of 2500. Hence, if the elapsed time is between 1.8 and 2.8 we will execute the inspection if the estimated repair cost is above 2500. If it is below, we will skip the inspection. In case the estimated repair cost is exactly 2500, both options have the same utility and a random choice can be made.

4.2 Use of Stochastic Predictions

Let us now consider a slightly modified example. The modification consists of the statistical model for the prediction function. In the following, we will examine the case where the prediction function provides us with a probability distribution for the remaining execution time. Therefore, if the elapsed time when the activity Check Cost is terminated is x, the prediction function provides us with the conditional probabilities

$$p_1 = P(\text{remaining time} < 5 - x \mid \text{Independent Inspection executed}),$$
$$p_2 = P(\text{remaining time} < 5 - x \mid \text{Independent Inspection skipped}).$$

These probabilities constitute the fulfillment values of the second goal for the different execution options depending on the time x at the point of decision. As a result, the decision procedure has to be generalized. Due to the use of probabilities, the expected increase in the utility of the second goal by means of skipping the inspection is $(p_2 - p_1) * 2$, where $(p_2 - p_1)$ is the increase in fulfillment and 2 the weight-factor. Note, that the technique described in Section 4.1 can be considered as a special case. If we have the same fixed execution times for activities as in Section 4.1, we would have $p_2 = p_1 = 0$ for $x > 2.8$ and $p_2 = p_1 = 1$ for $x < 1.8$ and thus, a utility of 0. For $1.8 \leq x \leq 2.8$ we have $p_2 = 1$ and $p_1 = 0$ and thus, a utility of 2. However, this generalization allows us to get more precise results in case of non-deterministic activity durations.

To make a decision, we have to compare the gain in utility of the second goal with a possible reduction in utility of the first goal if we skip the inspection. As discussed in Section 4.1, the fulfillment value of the first goal is 0.05 if the inspection is skipped and $\frac{\text{estimated repair cost}}{10000}$ if the inspection is performed. Therefore, the decrease in utility resulting from skipping the inspection is $(\frac{\text{estimated repair cost}}{10000} - 0.05) * 10$. If we combine these two results, we see that it is beneficial to skip the inspection if

$$(p_2 - p_1) * 2 > \left(\frac{\text{estimated repair cost}}{10000} - 0.05\right) * 10. \tag{1}$$

To make the example more concrete let us assume that the durations of the activities a_i follow a normal distribution $N(\mu_i, \sigma_i^2)$ with mean μ_i and variance σ_i^2. We still use the same expected values for the duration as in Section 4.1 but we a add a positive variance. For this example we specify the execution times as duration(Independent Inspection) = $N(1, 0.25)$, duration(Repair) = $N(2, 0.25)$, and duration(Finalize Claim) = $N(0.2, 0.01)$, e.g., the duration of the Independent Inspection has as normal distribution with mean 1 and variance 0.25 and thus, a standard deviation of 0.5.

Since the remaining workflow is a sequence, we know that the duration of the remaining workflow is the sum of the individual durations of the activities. Therefore, we can make use of the summation property of normal distributions (see for example [9]):

Theorem 1. *Let $X_1, X_2, \ldots, X_n$ be n independent random variables having normal $N(\mu_1, \sigma_1^2)$, $N(\mu_2, \sigma_2^2), \ldots, N(\mu_n, \sigma_n^2)$ distributions, respectively. Then $Y = X_1 + X_2 + \cdots + X_n$ is normally distributed with mean $\mu_1 + \mu_2 + \cdots + \mu_n$ and variance $\sigma_1^2 + \sigma_2^2 + \cdots + \sigma_n^2$.*

Therefore, we know that the remaining execution time of the workflow after the activity Check Cost has terminated is distributed $N(3.2, 0.51)$ if the inspection is performed and distributed $N(2.2, 0.26)$ if the inspection is omitted.

In the following, we will determine the advantageous execution structure for the situation that exactly two days have elapsed when we have to make the decision whether to execute the activity Independent Inspection or not. By using the cumulative distribution function for normally distributed random variables we can calculate that

$$p_1 = P(\text{remaining time} < 5 - 2 \mid \text{Independent Inspection executed}) = 0.390,$$
$$p_2 = P(\text{remaining time} < 5 - 2 \mid \text{Independent Inspection skipped}) = 0.942.$$

This means that we will keep the deadline with a probability of 39.0% if we perform the inspection and with a probability of 94.2% if we skip the inspection. Therefore, the expected increase in the utility of the second goal by means of skipping the inspection is $(0.942 - 0.390) * 2 = 1.104$. From (1) we can calculate that in this case it is beneficial to skip the activity Independent Inspection exactly if the estimated repair costs are below 1604.

5 Related Work

The idea to incorporate additional constraints into workflow specifications has especially drawn attention with respect to temporal goals. However, a common feature of all these approaches is that they mainly check whether a temporal constraint is fulfilled or not but do not actively apply this knowledge to adapt the workflow. In addition to this, all deadlines are considered as hard, i.e., there is no possibility violate them in certain situations.

Closest to our approach is the work of Panagos and Rabinovich [10,11,12]. They impose a deadline on a workflow execution and check at runtime, whether the deadline will be kept or not. They transform the overall deadline into individual deadlines for each activity and propose different criteria to decide when a workflow is considered as late. Depending on the delay of a workflow and the cost to abort the workflow at different stages, the workflow instance is either allowed to continue or aborted. The approach is similar to our work in the sense that they check the expected goal fulfillment at runtime and react on this. However, the only reaction policy in case of a late workflow is to abort the execution, whereas we aim at a graceful service degradation by adapting the workflow.

Marjanovic and Orlowska [13,14] propose an algorithm which is a generalization of a shortest path algorithm and the critical path method to calculate the maximal and minimal duration of a workflow. This algorithm is applied for verification purposes to find out whether deadlines are kept for all possible workflow instances, which can occur for different decisions in or-splits. They further extend this approach to check whether different temporal constraints are inconsistent.

Other approaches for calculating temporal properties and specifying temporal constraints of workflows can be found in [15,16]. In [15] a strategy is proposed to ensure at runtime that a specific kind of temporal constraint, namely upper and lower bounds on the temporal distance of two activities, is fulfilled. In case of an upper bound, the first activity involved in the constraint is delayed and in case of a lower bound, the second activity is delayed. However, this approach does not help in situations where it is necessary to accelerate the execution instead of delaying it.

Flexible workflows are a very active research area and since the seminal work of Ellis et al. [17] numerous approaches have been developed. See for example [18,19] for an overview. However, the goal of these approaches is to allow manual changes to the workflow structure. These can be either ad hoc changes to react to problems at runtime or evolutionary changes which are usually the result of a business process reengineering effort. Problems considered in this research area are for example the transition of running workflow instances to the new structure [17] and techniques to ensure the consistency of the resulting workflow, especially with respect to dataflow dependencies [20]. The only other approach known to the author that automatically derives the necessary changes from the runtime conditions is the work of Müller and Rahm [21]. They describe an approach using logic-based rules. These rules decide when an adaptation to running workflow instances is necessary and how the workflow structure has to be adapted. Hence, the resulting specification can be viewed as a rule-based workflow specification in which the possible execution paths and the conditions under which they are taken are represented as rules. In contrast to this, in our approach the workflow is adapted to a set of goals.

An approach to optimize and speed up the execution of workflows is described in [22]. The idea of this approach is to separate the preconditions for the start of an activity into data flow and control flow parts. Whereas an activity is *enabled* to start when the corresponding control flow condition is fulfilled, it is *ready* to start when the necessary input data is available. This makes it possible to start activities that have no side effects immediately after all input data is available. This can be the case before the control flow condition can be evaluated. If the control flow condition eventually turns out to be false, the activity is simply aborted. Although this is an interesting approach to speed up a workflow execution, it is limited to special cases and in many situations more severe modifications are necessary to ensure that a deadline is kept.

Other optimization techniques are more oriented towards the appropriate assignment of resources and tuning the underlying system components to the application requirements. An example can be found in [23].

6 Conclusion

In this paper we have developed an approach towards multidimensional workflows. This approach makes the workflow structure flexible in a controlled way. It further allows to model the underlying goals of a workflow and provides a way to systematically make use of the flexibility to adapt the workflow execution for an optimized goal fulfillment. As an initial step towards optimized decision making we have presented an optimization algorithm which compares all possible execution alternatives. We are currently working on more efficient optimization strategies. These depend on the QoS parameters subject to goals and the structure of the workflows. One example for such an optimization algorithm which is used to optimize a temporal and a utility goal is described in [8]. In addition, we have started to implement our approach for controlled flexibility within the CrossFlow demonstrator. The demonstrator system uses MQSeries Workflow from IBM [24] (the former FlowMark) as its underlying Workflow Management System. An overview about the components necessary to realize multidimensional workflows as well as how flexible elements are mapped to the process model offered by MQSeries Workflow can be found in [8].

Acknowledgements

The author would like to thank Karl Aberer, Thomas Tesch, Ralph Busse, Yigal Hoffner and Paul Grefen for their helpful comments.

References

1. D. Georgakopoulos, M. Hornick, and A. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2), April 1995.
2. A. Sheth, D. Georgakopoulos, S. Joosten, M. Rusinkiewicz, W. Scacchi, J. Wileden, and A. Wolf. Report from the NSF workshop on workflow and process automation in information systems. *ACM SIGMOD Record*, 25(4):55–67, 1996.
3. A. Cichocki, S. Helal, M. Rusinkiewicz, and D. Woelk. *Workflow and Process Automation – Concepts and Technology.* Kluwer Academic Publishers, 1998.
4. F. Leymann and D. Roller. *Production Workflow – Concepts and Techniques.* Prentice-Hall, 2000.
5. S. Browne and M. Kellet. CrossFlow Deliverable D1a: Insurance (Motor Damage Claims) Scenario. Report, Esprit Project No. 28635, January 1999.
6. WfMC. Workflow standard – Interface 1: Process definition interchange process model. Technical Report WFMC-TC-1016-P, Workflow Management Coalition, November 1998. Version 7.04.
7. J. Klingemann, J. Wäsch, and K. Aberer. Deriving service models in cross-organizational workflows. In *Proc. of RIDE – Information Technology for Virtual Enterprises*, pages 100–107, Sydney, Australia, March 1999.
8. J. Klingemann. CrossFlow Deliverable D8a: Flexible Change Control. Report, Esprit Project No. 28635, January 2000.

9. A. O. Allen. *Probability, Statistics and Queueing Theory.* Academic Press Inc., second edition, 1990.
10. E. Panagos and M. Rabinovich. Reducing escalation-related costs in WFMSs. In Dogac et al. [25].
11. E. Panagos and M. Rabinovich. Predictive workflow management. In *Proc. of the third Int. Workshop on Next Generation Information Technologies and Systems (NGITS)*, Neve Ilan, Israel, June 1997.
12. E. Panagos and M. Rabinovich. Escalations in workflow management systems. In *Proc. of the Workshop on Databases: Active and Real-Time (DART)*, Rockville, Maryland, November 1996.
13. O. Marjanovic and M. E. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Systems*, 1(2):157–192, May 1999.
14. O. Marjanovic and M. E. Orlowska. Time management in dynamic workflows. In *Proc. of the second Int. Symposium on Cooperative Databases and Applications (CODAS'99)*, Wollongong, Australia, March 1999.
15. J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proc. of the 11th Int. Conference on Advanced Information Systems Engineering*, pages 286–300, Heidelberg, Germany, June 1999.
16. H. Pozewaunig, J. Eder, and W. Liebhart. ePERT: Extending PERT for workflow management systems. In *Proc. of the first East-European Symposium on Advances in Database and Information Systems (ADBIS)*, pages 217–224, St. Petersburg, Russia, 1997.
17. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proc. of the Conf. on Organizational Computing Systems*, pages 10–21, Milpitas, California, August 1995.
18. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, Seattle, Washington, November 1998. http://ccs.mit.edu/klein/cscw98.
19. A. Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *Proc. of the DEXA Workshop on Workflow Management in Scientific and Engineering Applications*, pages 24–27, Toulouse, France, September 1997.
20. M. Reichert and P. Dadam. ADEPT$_{\text{flex}}$ – supporting dynamic changes in workflows without loosing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
21. R. Müller and E. Rahm. Rule-based dynamic modification of workflows in a medical domain. In *Proc. of the GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 429–448, Freiburg, Germany, March 1999.
22. R. Hull, F. Llirbat, J. Su, G. Dong, B. Kumar, and G. Zhou. Adaptive execution of workflow: Analysis and optimization. Working paper, Bell Labs, 1999.
23. D. Roller. Performance prediction and optimization in workflow-based applications. In *Proc. of the sixth Int. Workshop on High Performance Transaction Systems*, Asilomar, California, September 1995.
24. IBM. *MQSeries Workflow: Concepts and Architecture*, 1999. Version 3.2, GH12-6285.
25. A. Dogac, L. Kalinichenko, T. Özsu, and A. Sheth, editors. *Workflow Management Systems and Interoperability.* NATO-ASI Series. Springer Verlag, 1998.