Adjustment of Process Transactional Properties for Flexible Workflow Specification and Management

Jelena Zdravkovic

Department of Computer Science,
University of Gävle,
Kungsbäcksvägen 47, 80 277 Gävle, Sweden
jzc@hig.se

1 Introduction

Workflow management systems have emerged as powerful technologies for automating business processes. Besides the basic functionality that a process must satisfy, requests for quality of service (QoS) are to be set. A workflow system has to be able to fulfill established QoS goals, namely to satisfy requested cost limits, deliver results within desired timeframes, and attain some quality aspects, etc, etc. This leads to a need for a flexible workflow specification that will enable the necessary adaptations during run-time according to varying conditions and circumstances.

There have been significant works that explore the abilities by which to improve flexibility at the core workflow structure level. In this paper, we argue for the increase of workflow adaptability by introducing flexibility at the level of process transactional semantics. We propose an approach that leads to different execution structures, based on different settings for process transactional properties.

We will illustrate the motivation for our approach through a simplified example from the construction domain. The workflow depicted in Figure 1, starts when a customer comes to the construction company and makes an order for a farm and its accessories. After the order is confirmed, the site conditions are estimated – based on the farm design sketch and the nature of the terrain. The "value" of the site is reported and then revised by an expert team. On the basis of this report, the company hires an adequate labor force, including a manager. The next step involves work with the water pipes, including the entrenching, connecting with the main aqueduct and testing. In the meantime, the site is cleared, the terrain is prepared and material is gathered. Following this, construction of the farmhouse is initiated. When the construction work is completed, inspection of all the completed objects is performed. If necessary, additional corrections are carried out. Finally, the construction company contacts the customer to collect the farm key.

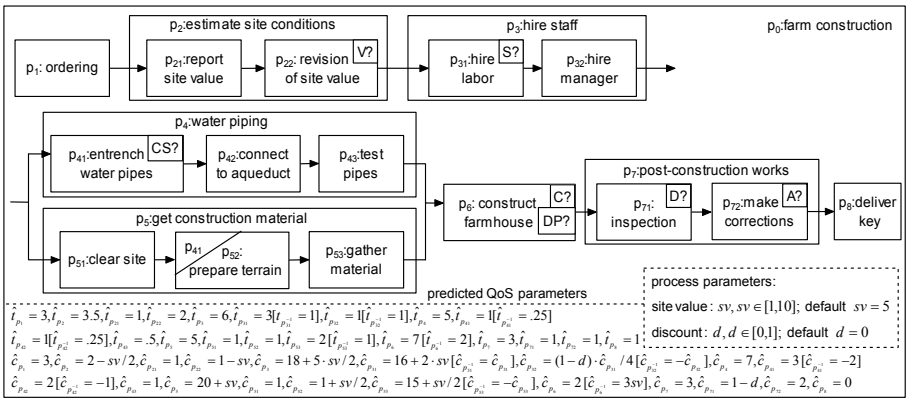

Figure 1. A Farm Construction Workflow

The most important goal of the construction company is to satisfy the customer. This is transformed into the following QoS requirements: the farm has to be constructed with minimal costs, within thirty days.

It is up to the company to complete the job in a way that achieves established goals. However, final QoS results can be only estimated, since some process parameters are calculated late in run-time and a number of undesired events may occur. The company knows for example, that revision of the site value process saves construction costs in proportion to the estimated site value. However, if the revision fails for some reason, the company has to decide whether it is better to fail the whole process of estimating site conditions or to go proceed without having the revision results. Furthermore, the company is aware that there is a discount when hiring the construction manager from the same source as the rest of labor force. If there is no available manager, a relevant question is whether it is better to re-hire all the staff from another source, or to search only for a manager. The company also has to know, based on a particular site conditions, whether it is better to allow for the preparation of the terrain right entrenching pipes, or to delay the process until the testing is successfully completed. Time and cost of farmhouse construction are around 10% lower if it is dry weather. The company prefers therefore to wait for a non-raining period if possible. Once the farmhouse is constructed, the company has to decide whether or not it can afford compensation for that process, due to some requirements. If it cannot do so then the main process is treated as being in the completing state. Furthermore, the company needs to decide whether inspection of the objects should be done without delay, or if it is possible to wait for experts and obtain their service for a lower fee. Finally, a decision needs to be made regarding corrections. Should corrections to work be canceled and redone if the quality of work is low, or should the corrections be completed with moderate results?

We consider the main process (p_0) as well as its sub-processes (p_2, p_3, p_4, p_5, p_7) from the Figure 1 as *complex processes*, with relaxed transactional semantics [1]. The bottom processes are considered as *basic processes*, with strict ACID features.

In what follows, we suggest a solution to improve QoS results by adjusting process transactional properties. We first define a collection of process transactional properties relevant for flexible workflow management. We then formalize QoS parameters and goals as well as a criterion for maximal fulfillment of goals. Following this, we formulate the process QoS values for possible execution structures, regarding different transactional settings. Finally, we show how the concept may be applied to the described scenario in order to obtain optimal QoS results.

2 Flexible Transactional Semantics

In this section, we first outline a set of process properties, which extend core transaction semantics:

- **Vital (V).** The property is used to determine the importance of a transactional sub-process. If a *vital* process fails, the parent process fails. Conversely, if a *non-vital* process fails, its parent process forwards execution, thus avoiding the recovery procedure.
- **Safepoint (S).** The property is used to enable a partial rollback of a process, in order to recover the process in a more efficient way. A process backward recovery is performed up to a sub-process marked as a *safepoint*, from which a forward execution may continue.
- **Commit Scope (CS)**. The property is used to avoid nonprotection of commit results given by the open nested model, which may lead to cascading aborts, if the "nonprotected" process is compensated. A process may commit its result either globally (*open*) or to one of its ancestor processes (*in-process*).
- **Compensatable (C)**. The property is used to determine how the effects of a completed process may be annulated. A process is *compensatable* if a compensation process is defined for it. A process is *pivot* if it is not possible to compensate it, upon its successful completion.
- **Dependency (DP).** The property is used to specify process execution constraints in contrast to other processes. In general, we express a dependency condition as: *change of a process p_i state is dependent on an event of process p_j.*

The outlined properties are chosen from related work as in [2]. In addition, we introduce:

- **May Be Aborted (A).** The property is used to specify whether it is allowed to abort a process or not. A process that can be aborted has to be recovered. Conversely, a process that cannot be aborted is completed without interruptions.
- **May Be Delayed (D).** The property is used to specify whether it is allowed to delay a process or not. A process that can be delayed, may wait for its invocation. Conversely, a process that cannot be delayed starts immediately upon a request for its execution.

The chosen properties present a set of process transactional features that we found relevant to flexible workflow specification and adaptable execution. In the "construction" scenario, explored uncertainties for optimal execution structures are transformed into flexible transactional specifications for processes (denoted on the Figure 1 as V?, S?, etc.). The final settings are determined during run-time according to varying conditions (late-constituted parameters, errors, etc.), based on a criterion for maximal fulfillment of QoS goals. As described earlier, process properties are related to exceptional situations. Their final settings are determined when these situations occur (for properties: *vital*, *safepoint*) or at time when a "flexible" process is requested for execution (for properties: *commit scope*, *compensatable*, *dependency*, *may be aborted*, *may be delayed*). Possible determination strategies are further discussed in [2].

3 Influence of Flexible Transactional Semantics on Workflow Management

Different QoS dimensions may be established for different business domains. Apart from common ones, such as time and costs, other examples of QoS dimensions are quite diverse and specific to particular business domains for instance, network throughput in Internet providing services.

A QoS parameter q_x^i is an occurrence of its dimension i, for a particular process p_x. QoS parameters that correspond to *basic* processes are observed either directly, or indirectly, as functions of other QoS parameters. For example, the cost q_x^1 of a *basic* process p_x may depend on its execution time q_x^2, as well as on its termination time q_y^3 of a process p_y, i.e. $q_x^1 = \varphi(q_x^2, q_y^3)$.

A QoS parameter q_x^i of a *complex* process p_x is determined from QoS parameters of its sub-processes, according to the execution structure (which may include both "original" and compensational sub-processes). In general,

$$q_x^i = \xi^i(q_1^i,..,q_n^i). \tag{1}$$

where n denotes the index of last sub-process of the process p_x. For instance, the cost of a process is found as the sum of the costs of the executed sub-processes:

$$q_x^i = q_1^i + .. + q_n^i. \tag{2}$$

The function ξ^i is the same for each complex process in the process hierarchy. Therefore, it is resolved recursively and automatically, starting from the *basic* process level and according to execution structures. This is discussed in more detail in [2].

During run-time, those QoS parameters that correspond to *basic* processes that have not been enacted can be predicted by using specified functions. A QoS parameter q_x^i that corresponds to a *complex* process p_x may be predicted as $\hat{q}_x^i = \xi^i(q_1^i,..,q_m,\hat{q}_{m+1},..,\hat{q}_n^i)$, where q^i corresponds to the QoS parameter of a completed process and $\hat{q}^i$ corresponds to the predicted QoS parameter of a non-completed process.

A QoS parameter q_x^i represents a QoS goal g if a limitation criterion is defined for it. For example, if a process must be completed within 7 days, then we set the goal for the process execution time. In [3], a method for finding the *total fulfillment* (tf) of workflow goals $(g_1, g_2 ... g_t)$ is defined, as:

$$tf = \sum_{j=1}^{t} w(g_j) \cdot f(g_j). \tag{3}$$

where w is a weight, which defines relative importance of the goal; f is a goal fulfillment function which describes in what extent the goal is satisfied.

During run-time, when a process with a flexible transactional property is encountered (according to the discussion from the end of Section 2), the system should be able to enumerate the setting that maximizes fulfillment of the goals, for the rest of execution. Thereafter, we formulate possible execution structures for each of the transactional properties settings to predict QoS parameters values of the affected processes:

– **Vital.** If a failed basic process p_x is set as *vital*, the parent process p_{par_x} will enter a recovery procedure up to the *safepoint* sub-process, and therefore its final QoS parameter will be found as:

$$\hat{q}_{par\_x}^i = \xi^i(q_1^i,..,q_{x-1}^i,\hat{q}_{(x-1)^{-1}}^i,\hat{q}_{(x-2)^{-1}}^i,..,\hat{q}_{(s+1)^{-1}}^i,\hat{q}_{(s+1)}^i,..,\hat{q}_x^i,..,\hat{q}_n^i).^* \tag{4}$$

If the process p_x is set as *non-vital*, the parent process p_{par_x} will resume with the forward execution:

$$\hat{q}^i_{par\_x} = \xi^i(q^i_1,..,q^i_{(x-1)},\hat{q}^i_{(x+1)},..\hat{q}^i_n)\,. \quad (5)$$

*a more complex situation, when the failure is not handled within the process p_{par_x}, but propagated further to upper levels, is formulated in [2].

We now illustrate the approach with the example from Section 1. The following parameters are defined:

QoS parameters - process execution time: $q^1_{ij} = t_{p_0}$; cost: $q^2_{ij} = c_{p_0}$.

QoS goals: $g_1 = t_{p_0}$, $w_1 = 1$, $f_1(t_{p_0}) = \begin{cases} 1, \text{if } t_{p_0} \le 30 \\ 0, \text{if } t_{p_0} > 30 \end{cases}$; $g_2 = c_{p_0}$, $w_2 = 1$, $f_2(c_{p_0}) = 1 - \frac{c_{p_0}}{100}$.

Upon completion of "report site value" process (p_{21}), site value parameter is set to 8. As stated in the example, if "revision of site value" process fails (for example, not enough parameters provided from the previous process; p_{21} had to be repeated), it should be decided whether the process should be considered as *vital* or not, according to the request for the maximal *total fulfillment*:

if $p_{22} \to vital$, from (4): $q^1_{p_2} = t_{p_2}, \hat{t}_{p_2} = t_{p_{21}} + \hat{t}_{p_{21}^1} + \hat{t}_{p_{21}} + \hat{t}_{p_{22}}$, $q^2_{p_2} = c_{p_2}, \hat{c}_{p_2} = c_{p_{21}} + \hat{c}_{p_{21}^1} + \hat{c}_{p_{21}} + \hat{c}_{p_{22}}$

$$tf = f_1(\hat{t}_{p_0}) + f_2(\hat{c}_{p_0}) = f_1(t_{p_1} + \hat{t}_{p_2} + .. + \hat{t}_{p_k}) + f_2(c_{p_1} + \hat{c}_{p_2} + .. + \hat{c}_{p_k}) = f_t(29) + (1 - \frac{75}{100}) = 1.25.$$

if $p_{22} \to non\_vital$, from (5): $q^1_{p_2} = t_{p_2}, \hat{t}_{p_2} = t_{p_{21}}$, $q^2_{p_2} = c_{p_2}, \hat{c}_{p_2} = c_{p_{21}}$

$$tf = f_1(\hat{t}_{p_0}) + f_2(\hat{c}_{p_0}) = f_1(t_{p_1} + \hat{t}_{p_2} + .. + \hat{t}_{p_k}) + f_2(c_{p_1} + \hat{c}_{p_2} + .. + \hat{c}_{p_k}) = f_t(26) + (1 - \frac{82}{100}) = 1.18.$$

as the result, the system sets the process p_{22} as *vital*.

The rest of the specifications for the process execution structures (i.e. for - *safepoint*, *commit scope*, etc.) may be found in [2]. When applied to the example from Section 1, the following results are obtained:

- Upon fail of "hire manager" process, "hire labour" process is set as a *safepoint*, *tf*=1.23 (conv. *tf*=0.27).
- Based on the result from "estimate site conditions" process, "entrench water pipes" process is set as *in-process* (p_5), *tf*=1.25 (conversely, *tf*=1.20).
- Based on the forecast report that for a non-raining period, one has to wait for 3 days, the event is removed from the "construction of farmhouse" dependency list, *tf*=1.25 (conversely, *tf*=0.252).
- Due to non-affordable compensation costs and time, "construction farmhouse" process is set as *pivot*, *tf*=1.25 (conversely, *tf*=-0.l).
- Based on the offer to get site inspection experts with a 50% discount (*d*=0.5) if they are two days late, "inspection" process is set as *may be delayed*, *tf*=1.255 (conversely, *tf*=1.25).
- Though the making of corrections is not being performed as expected, the corresponding process is set as *may not be aborted* due to time limits, *tf*=1.255 (conversely, *tf*=0.255)

4 Concluding Remarks

In this short paper we have presented an approach for adaptable workflow management. We have specified its influence on arbitrary types of QoS parameters, for nested workflow structures. Our work is unique as it considers the improvement of workflow flexibility at the level of process transactional semantics, thus offering optimal QoS results for executions followed by occurring transactional exceptions and failures.

References

1. Garcia-Molina, H.: Modeling Long-Running Activities as Nested Sagas. D.E. Bulletin, Vol. 14 (1991)
2. Zdravkovic, J.: Flexible Transactional Semantics for Adaptive Workflow Management. http://www.hig.se/~jzc/research.htm (2003)
3. Kligemann, J.: Controlled Flexibility in Workflow Management. Proceedings of the 12th Int. Conf. on Adv. Information Systems Engineering (CAiSE'00), Stockholm, Sweden (2000), 126 – 141.