

Dynamic Failure Recovery of Generated Workflows*

Michał Gajewski Mariusz Momotko
Rodan Systems S.A.
ul. Pulawska 465, 02-844 Warsaw, Poland
{Michał.Gajewski,Mariusz.Momotko}@rodan.pl

Harald Meyer Hilmar Schuschel Mathias Weske
Hasso-Plattner-Institute for IT-Systems-Engineering at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
{harald.meyer,hilmar.schuschel,mathias.weske}@hpi.uni-potsdam.de

Abstract

An important research area in the workflow management domain is the adaptation of workflows to unexpected events or failures at runtime. In this paper we present a concept for dynamic and automated workflow re-planning that allows to recover from such failures. To handle the complex dynamic situation of a partially executed workflow, we propose a multi-step procedure that includes the termination of failed activities, the sound suspension of the workflow, the generation of a new process definition, and the adequate process resumption. An important aspect of the presented re-planning procedure is that all steps including the generation of a new process definition are fully automated.

1. Introduction

Flexible business processes that can be easily adapted to handle unexpected changes or failures are one of the core challenges for today's companies [8]. One promising step in this direction is the application of Workflow Management Systems (WfM systems) [7, 11]. Business processes are explicitly modelled as workflows and enacted by a Workflow Management System. To handle unexpected changes or failures during enactment, the workflow has to be adapted [6, 12]. A problem with this approach is that workflow modelling is a manual task and thus costly, slow, fault-prone and organisationally difficult. As a result, adjustments and optimisations according to non-functional

properties for each individual business case are normally not economically feasible. Another problem with manually modelled workflows arises when the activities inside the workflow are calls to services. Service-oriented systems are open: services from different providers are registered and de-registered. Manually modelled workflows can hardly reflect these changes. An approach to solve these problems is automated workflow composition [5, 14, 10, 4, 3]. Workflows are not modelled manually but automatically composed. This approach allows optimising the workflow according to the goal and the non-functional properties of each case. Using automated workflow composition in a service-oriented system also allows reflecting changes in the available services easily. An additional advantage of automated workflow composition is the ability to automatically react on failures or unforeseen events. If the enactment of a workflow fails, it can be suspended and a new workflow can be composed from the current state. This approach is called re-planning.

In this paper we present a concept for dynamic workflow re-planning. To handle the complex dynamic situation of a partially executed workflow, we propose a multi-step procedure that includes the termination of failed activities, the sound suspension of the workflow, the generation of a new process definition, and the adequate process resumption. Considering related work there are some approaches for automated re-planning, but none of them explicitly take workflow suspension and resumption into account. In [10] requests expressed in the XSRL Web service request language are matched against standard business processes to generate an execution plan. Every time a knowledge gathering action is executed the additional information is used to generate a new plan. Ontologies of domain services and domain integration knowledge are used as a model for workflow integration rules in [3]. DYflow [14] avoids predefined

* This paper presents results of the Adaptive Services Grid (ASG) project (contract number 004617, call identifier FP6-2003-IST-2) funded by the Sixth Framework Programme of the European Commission

process definitions and allows the dynamic composition of Web services to business processes by applying backward-chain, forward-chain and data flow inference.

The example scenario used in this paper is modified version of the buddy scenario from the Adaptive Services Grid project [2]. The buddy scenario is an instant-messaging service for mobile devices. You are able to manage one or more lists of your buddies and you can talk to them if they are online. Besides this basic instant-messaging functionality, additional location-based services are offered. Figure 1 shows the process definition for a possible workflow to reach our goal. The speciality of this concrete service request is, that we know that our friend works at the Hasso-Plattner-Institute (HPI) and that the HPI offers a service to get the location of HPI. It also offers two additional services: *GetBuildingNumber* and *CalculateLocation*. While the previous can determine the building where a person is located, does the latter calculate a more exact location given the institute's coordinates and the concrete building number.

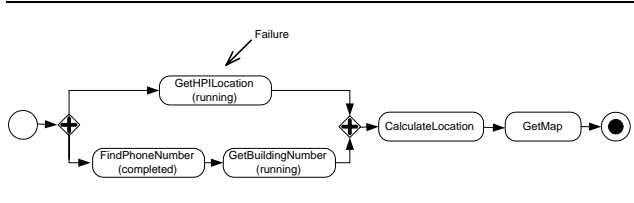


Figure 1. Scenario Enactment

In the next section we present an example scenario we will use to illustrate re-planning. In Section 3 the behavioural models of process and activity instances for the usage of re-planning are introduced. Our concept for dynamic re-planning is presented in Section 4. The paper closes with a conclusion and an outlook on future work.

2. Behavioural Models

One of the main preconditions for process re-planning is the assumption that the state of the process instance and all its activity instances does not change during re-planning. This assumption may be satisfied if re-planning is proceeded by suspending a given process instance and followed by resuming it.

These two above functions operate on process instance and activity instance entities and need to be represented in their behavioural models. The usual way of doing it is to define a new *Suspended* state together with two events: *suspend* which goes to and *resume* which goes from the defined state [1, 9]. A silent assumption of this approach is that suspension is done only for entities remaining in state

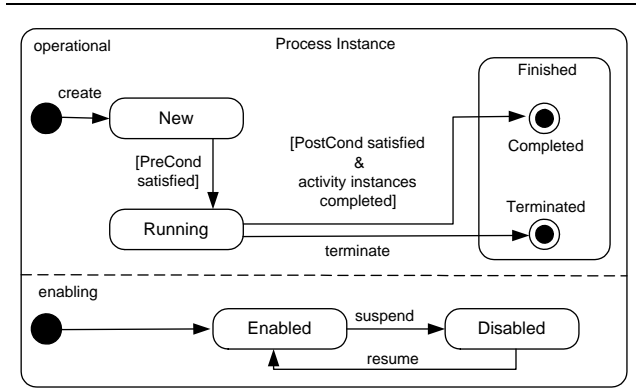


Figure 2. Process Instance

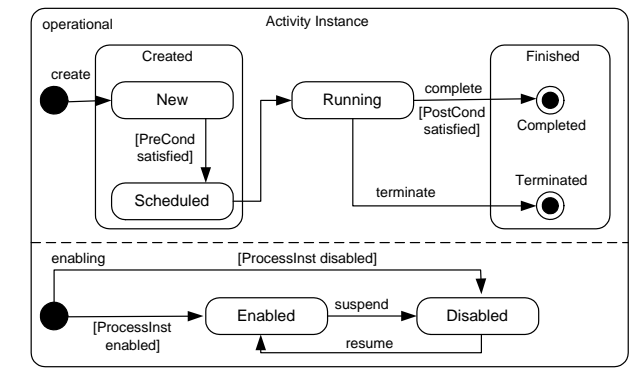


Figure 3. Activity Instance

Running. This assumption, however, seems to be too restrictive in case of re-planning since re-planning may occur to an activity instance remaining in any of its state (e.g. *Scheduled*).

To cope with the above problem we propose to introduce two new states, which enable or disable the Workflow Management system to process the mentioned entities. These states are called *enabling states* and are orthogonal to the rest of standard states which are called *operational states*. Regardless of the operational state in which a given entity remains, at any time it is possible to suspend or to resume a process or activity instance.

If the *suspend* event occurs, the entity moves to state *Disabled*. In this state the functions operating on the entity may behave in a different way. Basically, the functions responsible for assignment or execution of the entities omit disabled entities. For example, activity instances (or tasks) remaining in states *Scheduled+Disabled* will not be displayed in the participants' worklists. On the contrary, the functions responsible for completing or terminating entities will behave

in a similar way as those for enabled activities do. Suspension of a process instance causes suspension of all its activity instances. In this case, the suspend event is also sent to all activity instances of a given process instance.

If the *resume* event occurs a suspended entity may move back to state *Enabled*. This also triggers appropriate operation related to the entity operational state. For example, for activity instance remaining in state *Running*, this entity is re-run. Resuming of a process instance causes resuming of all its activity instances. In this case, the resume event is also sent to all activity instances of a given process instance. In addition, in the proposed approach state *Suspended* is not needed anymore. Now, it is expressed as a combination of *Running+Disabled* states.

2.1. Process Instance Behavioural Model

The model presented in Figure 2 consists of two group of states: operational and enabling ones. The state *New* is the operational state in which a process instance is created. If all its preconditions are satisfied, it starts its first activity and moves to state *Running*. Then it remains in this state until all its activity instances are finished and all its post-conditions (if present) are satisfied. Afterwards it moves to the state *Completed*. At any time it is possible to terminate the process instance. In this case, all its running activities are also being terminated. In addition, at any time it is also possible to suspend or resume the process instance.

2.2. Activity Instance Behavioural Model

The model presented in Figure 2 is an extension of the model proposed in [1]. Firstly, we add enabling states introduced earlier and, in addition, support it with starting conditions. When an activity instance is created its enabling state is the same as for the corresponding process instance. This feature is needed if we use the suspension technique based on completion of running activities. Secondly, an activity instance moves to the state *Scheduled* only if all its preconditions are satisfied. Similar situation is with post-conditions and the state *Completed* which is triggered by the event *complete*.

3. Re-planning

Re-planning of a workflow process is a quite complex operation and consists of several steps. Their fundamental concepts are described in the consecutive sections. Such description also refers to the re-planning scenario presented in the previous section.

3.1. Termination of the Failed Activity Instances

In the first step all the failed activity instances are terminated (i.e. they move to operational state *Terminated*). Usually there is only one such activity instance which caused re-planning. However, it is also possible that during re-planning some other activities that are still running (a suspending technique - see next section) will also fail. In that case re-planning must be terminated and then started once more with information about all failed activity instances (i.e. that which had failed before the first re-planning as well as those which have failed after the first re-planning started).

In our example, the activity instance represented by the *GetHPILocation* service had violated a constraint on its cost and the workflow monitor signalled its failure. Since it was not possible to re-assign the service implementation (this part is not explained in this article), the workflow enactment engine sends a request for re-planning this process instance. As the consequence, the *GetHPILocation* activity has been terminated.

3.2. Process Instance Suspending

In the next step the process instance and all its activity instances are suspended. This results in changing the enabling state of these entities to state *Disabled*. This operation is to assure that the mentioned entities will not change their state until re-planning is completed. Since an atomic activity may be executed as an application, for every instantiation of such activity it is also required to send a request for suspension to the corresponding application. If the application implements the operation *suspend* then such request will be handled correctly. Otherwise, it is not possible to suspend properly the application and thus the activity instance. There are two possible techniques to cope with this situation; we may either leave the running application (activity instance) to complete its execution or terminate the running application (activity instance). The former technique allows the running applications to complete. If they complete before finishing the re-planning, they will move to state *Completed*. Then the WfM system will evaluate the outgoing transitions and create new activity instances being the successors of the completed activity instances. Since the process instance remains in state *Disabled*, also new activity instance will remain in this state and thus they will not be processed until the state changes to *Enabled* (when the process instance is resumed). If the running application completes after finishing re-planning, the process will behave in the normal way. This is because resuming operation will change the enabling state of all activity instances to *Enabled*.

The former technique allows us not to waste the results of activity instances which have not failed but can not be suspended for re-planning. This is especially useful in case when the mentioned activities are still present in the new process definition and thus will have to be executed once again. Also using this technique we do not increase the constraints on the non-functional requirements such as response time or cost. On the contrary, if, during re-planning, such running application fails, it triggers a request for a new re-planning. In that case, we will have to prolong the re-planning operation (which may increase the time constraints on the process instance execution) and to ignore all the re-planning effort achieved so far (the input data for re-planning has changed). The latter technique terminates immediately all running activity instances. This technique may avoid re-planning reiteration as it was pointed out for the former technique. However, using this technique it is possible to waste the results that may be retrieved and to re-execute the same application. In addition, some application may not support termination. In our example, the former technique is used and the *GetBuildingNumber* activity instance is left to complete. When it completes the WfM System evaluates the outgoing transition and creates the routing activity. Since this routing activity remains in state *Disabled*, it will not be processed (i.e. waits for another token).

3.3. Generation of a New Process Definition

After suspending the process instance a new process definition is generated based on the current state of the case. This automated process generation works analogous to the initial generation of a process definition as described in section 1. The main difference is that instead of the initial state the current state is used as input. The current state is derived by starting with the initial state and retracing all effects of all currently executed or terminated activities. In this way, the current state reflects all previously unexpected effects of failing activities. Thus, these failures are automatically taken into account when using the current state to generate the new process definition. As a result, the new process definition describes a way of how to compensate the failed activity.

In our running example depicted in Figure 1 the activity *FindPhoneNumber* completed and *GetHPILocation* failed. This means that in the current state the phone number is determined, but the HPI location is not. Generating a new process definition based on this state results in the process definition shown in Figure 4. It describes a new way to reach the goal despite of the failure. Instead of using *GetHPILocation* the phone provider's localization services *TelekomGetLocation* or *VodafoneGetLocation* are used, depending on which provider the phone number belongs to.

An important aspect is that there may be some activities,

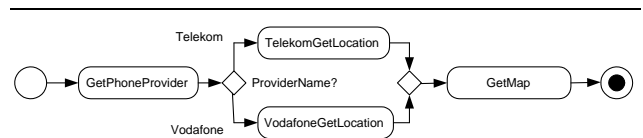


Figure 4. Scenario New Process Definition

which can not be suspended for re-planning. These activities can have effects that change the state of the case. Thus, the state of the case at the beginning of re-planning can be different from the state of the case at the time process enactment is to be resumed. To handle this problem, running activities can be represented by proxy activities during re-planning as described in [13].

3.4. Process Instance Re-planning

Afterwards the old and newly generated process definitions are merged and, according to that merged definition, the process instance is prepared for resuming and further execution (continuation). In order to merge the old and new process definitions the activities present in both definitions must have the same identifiers (not changed during generation of the new definition). At the activity level re-planning is carried out in the following way, for activities present:

- in both old and new process definitions no action is taken,
- only in the old process definition in case the operational state was *Completed* or *Terminated* no action is taken, otherwise they need to be terminated,
- only in the new process definition we need to create new activity instances. In addition, for intermediate activities (one or more ingoing and outgoing transitions) we also need to instantiate all ingoing transitions required for the activities.

In our example the activity *FindPhoneNumber* was previously completed so nothing happens with it. During failure *GetBuildingNumber* was left to complete and it does not occur in the new process definition. Since the activity *GetHPILocation* was terminated, the succeeding routing activity will not be processed, therefore it has to be terminated. Then the WfM system creates the new activity *GetPhoneNumber* and instantiates its ingoing transition (from *FindPhoneNumber*).

3.5. Process Instance Resuming

The final step for re-planning is to resume the re-planned process instance and continue it with the new, merged process definition. To resume the process instance at least one

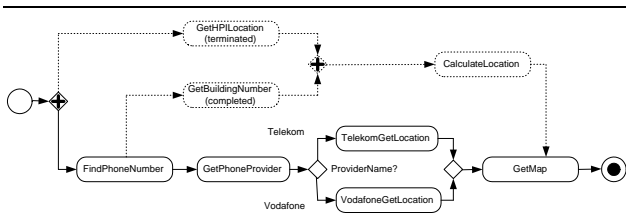


Figure 5. Scenario Resuming

of its activity instances has to remain in a not finished state (i.e. *New*, *Scheduled* or *Running*). This precondition assures that after resuming the process instance will be executed. The resuming event causes that the enabling state of the process instance and all its activity instances is changed to *Enabled*. In addition, this event also triggers re-running of the activity instances remaining in state *Running*. In our example presented in Figure 5 the resuming operation triggers execution of *GetPhoneProvider* activity and then the rest of the process.

4. Conclusion

Concerning areas of application, the presented concept offers little additional value over traditional workflow management systems, if execution always runs as expected. In contrast, applications in which activities are likely to fail or re-planning has to be done without human interaction, can benefit from the ability of an automated re-planning procedure. Thus, the presented approach has the potential to become an enabling technology for new fields of application for workflow management technology.

Future work concentrates on validating our approach in real world, industrial scenarios. An important step into this direction is the Adaptive Services Grid Project. In this project the concepts presented in this paper are augmented by additional concepts to increase adaptivity like negotiation with providers and the usage of grid services. These concepts will be used in domains like telematics, dynamic supply chain management and dynamic marketplaces.

References

- [1] W. Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2), 2003.
- [2] Adaptive Services Grid-Projekt. <http://asg-platform.org>.
- [3] V. Atluri and S. A. Chun. Handling dynamic changes in decentralized workflow execution environments. In *Proceedings of the 14th International Conference on Database and Expert Systems Applications*, LNCS. Springer, 2003.
- [4] C. Beckstein and J. Klausner. A planning framework for workflow management. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.
- [5] J. Blythe, E. Deelman, and Y. Gil. Planning for workflow construction and maintenance on the grid. In *ICAPS'03 Workshop on Planning for Web Services*, 2003.
- [6] C. Ellis, K. Keddera, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, pages 10–21. ACM Press, 1995.
- [7] D. Georgakopoulos, M. F. Hornick, and A. P. Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
- [8] M. Hammer and J. Champy. *Reengineering the corporation*. Harper Collins Publishing, New York, 1993.
- [9] D. Hollingsworth. The workflow reference model version 1.1. Technical Report WFMC-TC-1003, Workflow Management Coalition, January 19th 1995.
- [10] A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. In *1st international conference on service-oriented computing (IC-SOC'03)*, Trento, 2003.
- [11] F. Leymann and D. Roller. *Production workflow: concepts and techniques*. Prentice Hall, 2000.
- [12] M. Reichert and P. Dadam. ADEPTflex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
- [13] H. Schuschel and M. Weske. Triggering replanning in an integrated workflow planning and enactment system. In *8th East-European Conference on Advances in Databases and Information Systems*, volume 3255 of LNCS. Springer, 2004.
- [14] L. Zeng, B. Benatallah, H. Lei, A. H. H. Ngu, D. Flaxer, and H. Chang. Flexible composition of enterprise web services. *Electronic Markets - Web Services*, 13(2), 2003.