Adding Support for Dynamics Patterns to Static Business Process Management Systems

René Wörzberger, Nicolas Ehses, and Thomas Heer

Department of Computer Science 3 - Software Engineering
RWTH Aachen, Germany
{woerzberger,ehses,heer}@i3.informatik.rwth-aachen.de

Abstract. Many companies use business process management systems (BPMS) for modeling and execution support of their business processes. Many processes are highly dynamic and require changes even during execution. Common commercial BPMS fail to support such processes appropriately since they work in a rather static manner, i.e. they demand that the structure of a process is fixed before execution.

Our research group cooperates with an industry partner who uses a static BPMS. This paper describes an approach that posteriorly extends this static BPMS inasmuch as dynamic changes of processes during execution are supported. The benefit of this approach is that our partner in industry gains support of dynamic processes but still use the existing BPMS and save investments related to it.

1 Introduction

The organization of most modern companies is aligned to their business processes. *Business process management systems (BPMS)* provide means to define, control and monitor business processes, which are often called *workflows* in this context.

Common BPMS distinguish between *build time* and *run time* with regard to processes. During build time *workflow definitions* are specified, which model a certain *process type*, e.g. "adjustment of a claim" in an insurance company. At *run time*, for each actual *process case* a *workflow instance* is created and executed according to a certain workflow definition.

Although companies strive for automation of their business processes, most processes are at least partly conducted by humans. These processes are often *dynamic*, i.e. the structure of such processes evolves during process execution. Most commercial BPMS support rapid adaptions of workflow definitions but prohibit dynamic changes in workflow instances, like adding missing activities. Thus, they cannot optimally support dynamic processes. This problem gave rise to a new research field [1,2,3].

Within the collaborative research center IMPROVE [4] our research group has realized the prototype AHEAD [5], which aims at the management of highly dynamic development processes and interdigitates build time and run time. In an ongoing research project we transfer our concepts to industry in cooperation

C. Pautasso and É. Tanter (Eds.): SC 2008, LNCS 4954, pp. 84–91, 2008.

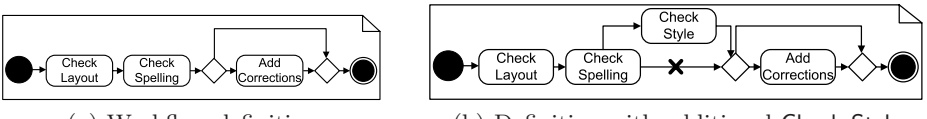
with our partner AMB Generali Informatik Services (AMB-Informatik). Our partner is the information technology service provider for the Generali Group, which is a combine of insurance companies.

AMB-Informatik uses the *WebSphere BPMS* consisting of the workflow definition tool *WebSphere Integration Developer (WID)* and the run time environment *WebSphere Process Server (WPS)*[1] as solution basis for their customers. WebSphere BPMS is rather oriented towards highly automated and predictable processes. The support for dynamic processes is not in its focus. However, these dynamic processes occur and must also be handled by insurance companies. Thus, we transfer the concepts of the prototype AHEAD by *extending* WebSphere BPMS. Thereby, we gain support for dynamic processes while investments of our partner in the WebSphere-based infrastructure can be saved.

The paper is structured as follows: Section 2 clarifies terms related to flexibility and dynamics of BPMS. Dynamics can be categorized according to patterns, which are described in Section 3. Section 4 explains our approach with regard to the patterns and the extended WebSphere BPMS. Connections to other research approaches are illustrated in Section 5. Section 6 concludes this paper with an outlook on future work.

2 Flexibility and Dynamics of Workflows

Most workflow definition languages provide construct types like decision or iteration to define at build time execution sequences of activities, which are valid at run time. These construct types allow for some *build time flexibility*, that is, one workflow definition specifies a (possibly infinite) set of valid execution sequences. The workflow definition depicted in Figure 1(a), a simple review process, allows for two execution sequences: one with Add Corrections and one without.



(a) Workflow definition (b) Definition with additional Check Style

Fig. 1. Example for run time dynamics

According to our industry partner, for a non-trivial process type it is neither possible due to limited anticipatability nor desired because of lower maintainability to build a workflow definition that covers all reasonable execution sequences. For instance, it might turn out during execution of Check Spelling that the particular document is written in a bad style. Then, there is a need for an additional activity Check Style before Add Corrections in the respective workflow instance. In this case, it would be appropriate to dynamically change the running

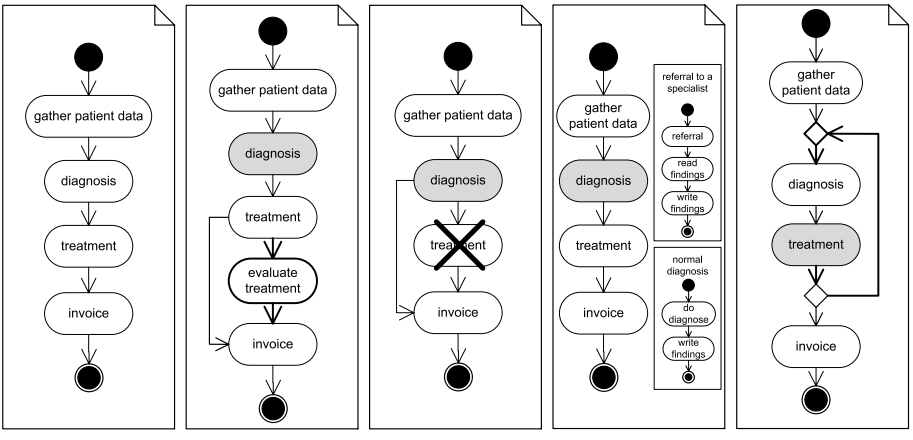
[1] All WebSphere trademarks are in possession of IBM Corp. (http://www.ibm.com).

workflow instance (*run time dynamics*) such that the instance would conform to the hypothetical workflow definition of Figure 1(b).

3 Dynamics Patterns

In most cases, the necessity to modify a workflow instance is recognized by *workflow participants* of the respective workflow instance and should therefore be conducted by them.

A *workflow modeler* normally needs the full set of constructs types at build time to cover all reasonable execution sequences he is able to envisage. In contrast, at run time a workflow participant just uses a small subset of the available construct types. That is because he does not consider workflow modeling as his primary task but rather wants to satisfy current or oncoming exigencies for a single workflow instance by a dynamic modification. Furthermore, we think that most dynamic modifications follow certain patterns, which we exemplify in the following by a simplified process type of a medical practice (cf. Fig. 2(a)).



(a) Workfl. def. (b) Adding (c) Removing (d) Realization (e) Iteration

Fig. 2. Examples for dynamics patterns

Dynamic Adding. During the enactment of diagnosis in a certain workflow instance the doctor might recall a research project concerning the effectiveness of certain treatments. Since the patient is willing to partake, the doctor has to assure that the evaluation of the treatment will be executed after the treatment itself. The best way to do this is to dynamically add an additional activity evaluate treatment between the existing activities treatment and invoice (cf. Fig. 2(b)).

Dynamic Removing. In the process of the medical practice it is possible that there are no findings resulting from diagnosis because the patient just suffers from a slight indisposition. Consequently, the doctor dynamically removes the activity treatment from the respective workflow instance (cf. Fig. 2(c)).

Dynamic Realization. In a hierarchically defined workflow, where activities are actually realized via sub-workflows, the best realization can often not be identified before run time. E.g., the doctor might dynamically realize diagnosis by normal diagnosis conducted by himself or by a referral to a specialist (cf. Fig. 2(d)). This decision is specific to each particular workflow instance.

Dynamic Iteration. In the medical process, errors might occur in each activity. For instance, a severe mistake is a wrong finding resulting from a faulty diagnosis. Such an error might be recognized during execution of treatment. Then, the affected activities of the workflow instance, namely, diagnosis and its successor activity treatment have to be dynamically re-iterated, i.e., they are carried out again such that the data they produce can be corrected (cf. Fig. 2(e)).

4 Approach: Simulate Dynamics on a Static BPMS

In this section we describe our approach to provide support for the dynamics patterns described in Section 3. The distinctive feature of this approach is that our prototype is not implemented from scratch. Instead, we rather add another layer on top of an existing BPMS by which we simulate a dynamic BPMS (cf. Fig. 3). The fact that the underlying BPMS is static is hidden from both the workflow modeler at build time and from the workflow participant at run time. Although we have implemented a prototype on top of the concrete system WebSphere BPMS in order to verify our approach, we think that it can be easily adapted to other BPMS.

4.1 Approach Overview

At *build time* a WS-BPEL transformer augments workflow definitions modeled in the language WS-BPEL[2] via WID by additional WS-BPEL activities (e.g. `<invoke>`, `<switch>`, `<while>`) yielding an augmented workflow definition Xa. The transformer requires no user interaction. Hence, the additional dynamics layer is opaque to the workflow modeler.

The *run time* counterpart of the WS-BPEL transformer is the dynamics component. This component is accessed through two interfaces that serve different purposes: (1) The dynamics component stores instance specific run time information, e.g. actual variable values and routing informations for sub-workflow calls. This run time information is accessed by the WPS through the WPS interface. WPS uses this information for those WS-BPEL activities that have been added at build time by the WS-BPEL transformer. (2) Likewise the WS-BPEL transformer hides the dynamic aspect at build time, the dynamics component hides the additional WS-BPEL activities from a workflow participant at run time. Instead, it provides a participant interface which is used by a participant GUI to render a graphical view of dynamic workflow instances. Furthermore, the

[2] http://www.oasis-open.org/committees/wsbpel/

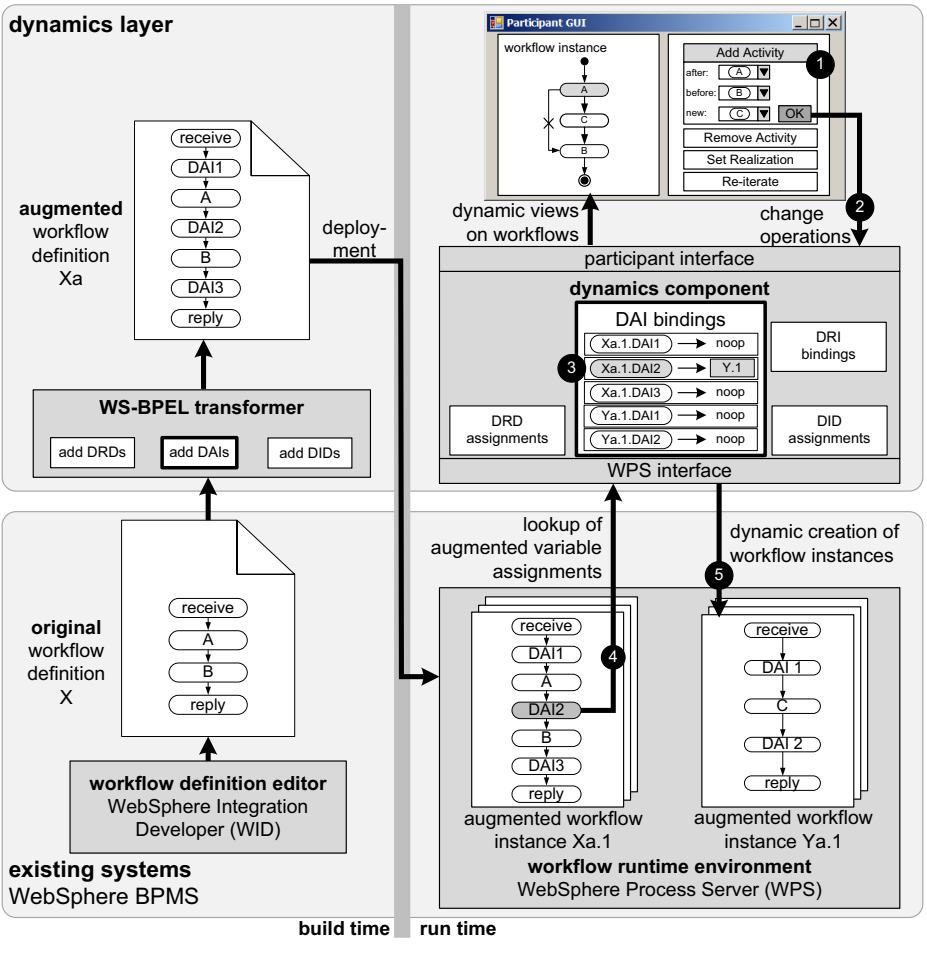


Fig. 3. System overview and example for Dynamic Adding support

participant interface of the dynamics component offers operations that can be invoked by a workflow participant via the participant GUI to perform a dynamic change in the graphical view.

4.2 Realization of Dynamic Adding

The approach is aligned to the dynamics patterns explained in Section 3. In order to keep our explanation brief, we just exemplify our approach with Dynamic Adding in a rather small workflow.

Build Time. Enabling a workflow definition to support Dynamic Adding (s. Sec. 3) requires the addition of `<invoke>` activities which we call *Dynamic*

Adding Invocations (DAI) in the following. DAIs serve as placeholders for possible additional activities. Since Dynamic Adding might take place in arbitrary positions, the WS-BPEL transformer inserts a DAI before and after each activity.

The left hand side of Figure 3 exemplifies the transformation of a simple workflow X definition, which just consists of the sequential <invoke> activities A and B, depicted as rounded rectangles. The transformation yields an augmented workflow definition Xa[3] with three more <invoke> activities DAI1 to DAI3.

Run Time. A Dynamic Adding edit operation is initiated by a workflow participant via his participant GUI (1). He specifies *which activity* is to be inserted at *which position* in the control flow. In this case, during the execution of activity A, the participant inserts a new activity C right after the existing activity A and before B. The participant GUI notifies the dynamics component about this edit operation (2). Consequently, the dynamics component replaces the default noop-binding[4] of DAI2 in workflow instance Xa.1 by a sub-workflow call to a workflow instance Ya.1 (3). When the control flow reaches DAI2, this activity calls the dynamics component (4) which creates a new workflow instance Ya.1 (5). After Ya.1 is completed, the control flow returns to Xa.1, which proceeds with activity B.

The other dynamics patterns can similarly be realized (in combination with each other), e.g. by means of so called *Dynamic Removing Decisions (DRD)* and *Dynamic Iteration Decisions (DID)*. Both require additional transformation steps in the WS-BPEL transformer and additional run time data in the dynamics component (cf. Fig. 3). *Dynamic Realization Invocations (DRI)* are similar to DAIs but refer to activities, which are already part of the original workflow definition like A or B in Figure 3.

5 Related Work

Flexibility and Dynamics. Our distinction between build time flexibility and run time dynamics is clearly aligned with the common distinction between build time and run time in workflow management systems. A similar distinction between "a-priori flexibility" versus "a-posteriori flexibility" and "offline changes" versus "online changes" is made by Joeris [6] and Bandinelli et al. [7], respectively.

Dynamic WfMS. There are several workflow and process management systems, which support run time dynamics to some extend. A comparison of some academic prototypes like ADEPT [8] or WIDE [9] and commercial systems is given by Weber et al. [10]. There is no approach known to us that posteriorly extends an existing workflow management system with a given workflow definition language by support for run time dynamics.

[3] In the figure we use a restricted subset of the notation for UML activity diagrams since there is no official graphical notation for WS-BPEL.

[4] "Noop" stands for "no operation".

Patterns. Aalst et al. [11] introduced a classification for workflow languages based on patterns which concentrates on build time flexibility. Voorhoeve [12] et al. make an implicit classification for run time dynamics with regard to preservation of certain consistency properties of petri nets. Weber et al. [10] also range run time dynamics but not with regard to a concrete technical implementation basis.

6 Conclusion

Summary. In this paper we described how support for run time dynamics, e.g. dynamic modifications of workflows, can be realized by an additional dynamics layer based on a static workflow management system. Our approach is motivated by requirements of our industrial partners at AMB-Informatik who want to support dynamic processes but also keep the existing static WebSphere BPMS. We classified dynamic changes according to dynamics patterns. In our twofold architecture, consisting of a WS-BPEL transformer for build time and a dynamics component for run time, there is dedicated support for each dynamics pattern.

State of Implementation. Presently, we are implementing the dynamics layer and the graphical participant GUI only using common and publicly available technologies. The WS-BPEL transformer is realized with XSL Transformations whereas the dynamics component is written in plain Java. The graphical participant GUI is implemented using the Graphical Editing Framework (GEF) of the Eclipse Foundation. Though the implementation is still in an early phase, first results already substantiated the suitability of our approach.

Current Limitations and Future Work. In the original workflow definition X in Figure 3 we only used a small subset of all WS-BPEL construct types. Actually, occurrences of *complex construct types* like decisions and iterations but not yet compensation and fault handlers. We will also deal with problems arising from the *concurrency* and *distribution* of workflows. Here, we can continue work that has been done in a preceding project of our group [13]. *Generalization* of the approach is another important goal beginning with the adaption of other WS-BPEL-based workflow management systems and proceeding with workflow management system of other kinds. *Optimizations* will be applied particularly to the WS-BPEL transformer in order to reduce the size of the augmented workflow definitions. Besides the work presented in this paper there are *other related parts* in our cooperation with our partner AMB-Informatik again carrying on preceding work. Dynamic changes made by workflow participants have to be checked against certain constraints in order to guarantee technical consistency, e.g. data dependencies between activities. Furthermore, professional constraints have to be enforced, e.g. non-deletability of strictly mandatory activities. Both will be supported by a *consistency checker*. We will also build a tool that provides a *condensed view* of completed workflow instances with dynamic changes to the workflow modeler. By using this tool, the modeler can identify similar dynamic changes among the workflow instances and copy them to the workflow definition where appropriate.

References

1. Ellis, C.A., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COOCS, pp. 10–21. ACM Press, New York (1995)
2. van der Aalst, W.M.P., Jablonski, S.: Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science and Engineering 15(5), 267–276 (2000)
3. Bernstein, A., Dellarocas, C., Klein, M.: Towards adaptive workflow systems: CSCW-98 workshop report. SIGGROUP Bull. 20(2), pp. 54–56 (1999)
4. Nagl, M., Marquardt, W. (eds.): Collaborative and Distributed Chemical Engineering Design Processes / From Understanding to Substantial Support. Springer, Heidelberg (2008)
5. Westfechtel, B.: Ein graphbasiertes Managementsystem für dynamische Entwicklungsprozesse. Informatik Forschung und Entwicklung 16(3), 125–144 (2001)
6. Joeris, G.: Flexibles und adaptives Workflowmanagement für verteilte und dynamische Prozesse. PhD thesis, University of Bremen (2000)
7. Bandinelli, S., Di Nitto, E., Fuggetta, A.: Policies and Mechanisms to Support Process Evolution in PSEEs. In: Proceedings of the 3rd International Conference on the Software Process, pp. 9–20. IEEE Computer Society Press, Los Alamitos (1994)
8. Reichert, M., Dadam, P.: ADEPTflex-Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems 10(2), 93–129 (1998)
9. Casati, F.: Models, Semantics, and Formal Methods for the Design of Workflows and their Exceptions. PhD thesis, Politecnico di Milano (1998)
10. Weber, B., Rinderle, S.B., Reichert, M.U.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
11. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distrib. Parallel Databases 14(1), 5–51 (2003)
12. Voorhoeve, M., Van der Aalst, W.: Ad-hoc Workflow: Problems and Solutions. In: Tjoa, A.M. (ed.) DEXA 1997. LNCS, vol. 1308, p. 36. Springer, Heidelberg (1997)
13. Heller, M., Wörzberger, R.: A Management System Supporting Interorganizational Cooperative Development Processes in Chemical Engineering. Journal of Integrated Design and Process Science: Transactions of the SDPS 10(2), 57–78 (2007)