# Workflows in Dynamic Development Processes

Thomas Heer, Christoph Briem, and René Wörzberger

RWTH Aachen University, Department for Computer Science 3,
Ahornstr. 55, Aachen 52074, Germany,
http://se.rwth-aachen.de,
{heer|chrbriem|woerzberger}@i3.informatik.rwth-aachen.de

**Abstract.** Process management systems are used in many domains to monitor and control processes. Development processes require specific support, which is not provided by conventional workflow systems. On the one hand, they comprise highly dynamic process parts, which cannot be defined until project runtime. On the other hand, development processes may contain subprocesses, which can be executed in the form of predefined workflows. In this paper we describe the integration of a specialized system for the management of dynamic development processes with a conventional workflow management system. This integrated solution targets the specific needs for the management of engineering design processes in the plant construction domain. It has been implemented as an extension to a commercial CAE-tool, and it will be evaluated in industrial practice in the near future.

**Key words:** dynamic workflow, development process, task net, process management

## 1 Introduction

Process management is nowadays used in many *different application domains*, which often comprise substantially different *process types*. The managed processes range from claim handling cases in insurance companies to development processes in software or mechanical engineering. The characteristics of a process type lead to *specific requirements* for the respective *process meta-model* and the *tools* for managing processes of this type.

An insurance workflow for example usually describes a *predefined procedure*, which a clerk has to follow. In contrast to that, a development process definition often merely defines the main phases or coarse-grained steps to be executed during the development of a product, as well as the main artifacts, which should be produced. In the former case, language constructs are needed for modeling *alternative courses of action*. Deviations from the predefined process definition constitute exceptional cases. In the latter case the execution of *ad-hoc processes* should be supported. These processes can be *elaborated during enactment*, whereas the process model is continually modified.

While workflow technology can be applied to support the former process type, it is not suitable for the support of development processes as a whole.

Therefore, we implemented the process management system PROCEED[1], which provides *specific support* for the management of *development processes* in the plant engineering domain.

On the one hand, conventional workflows are not appropriate to model whole development processes, because many dynamic subprocesses cannot be predefined before enactment. On the other hand, there are nonetheless many *subprocesses* in a dynamic development process, which are *repetitive and static*. These static subprocesses can be supported by common workflow management systems. But they must not be executed in isolation, unaware of their *surrounding process context*. Therefore we integrated PROCEED with a conventional workflow management system to support dynamic development processes, which comprise predefined workflows as subprocesses.

The development of this integrated solution is undertaken in the context of the *transfer project T6* [1] of the *transfer center 61* of the *collaborative research center 476 IMPROVE* [2]. In this project we transfer the research results of the AHEAD project [3] to industrial practice. The project is funded by the DFG[2] and is executed in close cooperation with the innotec corporation [4]. PROCEED is implemented as an extension to the computer aided engineering (CAE) tool *Comos*, a product of innotec.

The paper is structured as follows. In Section 2 we describe DYNAMITE meta-model for dynamic development processes and compare it to conventional workflow meta-models. In Section 3 we describe, how we integrated the two different approaches into a system, that supports all types of static and dynamic subprocesses in a development project. In Section 4 we describe and evaluate related work before we give a conclusion in Section 5.

## 2 Dynamic Task Nets and Conventional Workflows

In this section we briefly describe the DYNAMITE meta-model [5, 6, 3] for the modeling and enactment of *DYNAMIc Task nEts* and compare it to conventional workflow meta-models.

### 2.1 DYNAMITE

The development of DYNAMITE was motivated by the need for *tool support* for the *management of development processes*. Therefore, the DYNAMITE meta-model comprises language constructs for modeling the coordination of engineers in simultaneous engineering scenarios, the propagation of development artifacts and the like.

DYNAMITE is part of an *integrated meta-model* for the management of dynamic development processes, which constitutes the foundation of the process management system AHEAD [3] as well as of the newly developed PROCEED

[1] PROCess management Environment for Engineering Design processes
[2] Deutsche Forschungsgemeinschaft

system. While AHEAD was a research prototype, PROCEED is an extension to the commercial CAE-tool Comos. DYNAMITE is the partial meta-model for task management. It is complemented by ResMod (Resource Modeling) and CoMa (Configuration Management), which are meta-models for *resource management* and *product management* (documents, artifacts), respectively. The three partial meta-models are tightly integrated. In the following we briefly describe the modeling constructs of DYNAMITE by example.

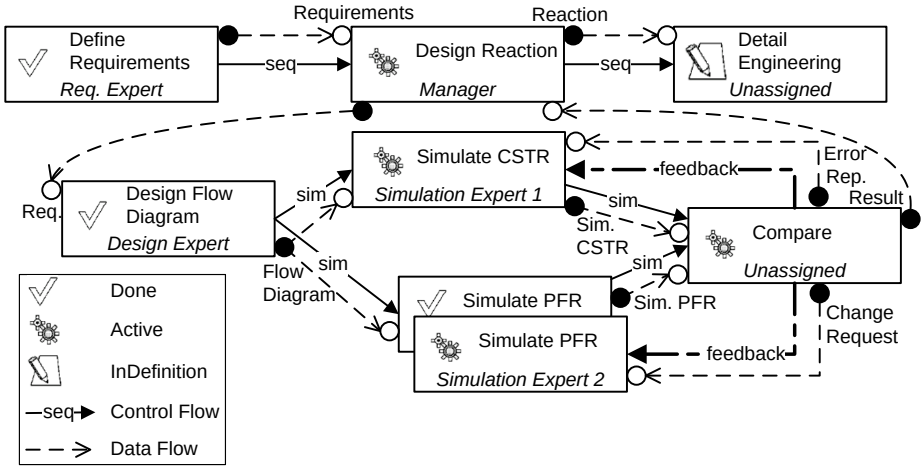


**Fig. 1.** Example of a DYNAMITE task net.

In Figure 1 a small example of a DYNAMITE task net is depicted. This example has been derived from a larger scenario from the domain of chemical engineering, which is described in [2]. Tasks are represented by rectangular boxes which are connected by *control flow*, *data flow* or *feedback flo*w edges.

The tasks of a process, which is modeled using DYNAMITE, are all part of one overall task net. This task net is *hierarchically structured*. Each task can contain arbitrarily many subtasks. In Figure 1 the hierarchy is depicted by means of the layout. The subtasks of Design Reaction are arranged below the task itself. In PROCEED it is possible to navigate through the task hierarchy.

Six different *execution states* of tasks are defined in DYNAMITE: *InDefinition*, *Waiting*, *Active*, *Suspended*, *Done*, and *Failed*. The transition rules for the execution states of a task are defined in the form of a finite state machine (cf. [5]). The possible execution state transitions of a task depend also on the execution states of all tasks, which are connected to the task by control flows and feedback flows.

In DYNAMITE a *control flow* connects exactly two tasks and it can have one out of three different semantics.

– A *standard* control flow defines, that the target task of the control flow must not be terminated before the source. This is the minimal requirement for a

control flow. However, the source and target of a control flow can be executed in parallel.
- A *simultaneous* control flow defines the additional restriction, that the source task must be started before the target. The motivation for this control flow semantic is, that the resources of the source and target tasks cooperate, and that a first intermediate result of the source task is required to start working on the target task. At the same time the standard semantics guarantees that the last version of the source's output is consumed by the target task before it is terminated.
- The most restrictive control flow is that of type *sequential*. In this case the target task may only become active after the source task has been terminated. Note, that most workflow management systems only use the sequential control flow semantics and therefore restrict the execution order of tasks too much for typical cooperation scenarios in development processes.

The DYNAMITE meta-model also comprises the concept of *data flows*. Data flows refine control flows. By means of data flows it can be explicitly modeled, which products are transferred from one task to another.

In DYNAMITE *feedback flows* address the specific dynamic situations in development processes. Although the model of a development process usually defines dependent tasks and hence an *order of execution*, the actual enactment of these tasks in a project is *carried out iteratively*. In case of a change request or a detected error in an artifact, certain *previous process steps* have to be *redone*. To trace cases, in which feedback from a succeeding task is given to a previous task, a feedback flow is introduced. If a task has already been terminated and therefore has to be restarted to handle feedback from succeeding tasks, a *new task version* is created for this and succeeding terminated tasks (cf. Fig. 1). The use of feedback flows and versioned tasks is a possibility to redo already terminated process parts during enactment and to ensure *traceability* at the same time. The former is a functionality, which most workflow management systems do not support to this day, but which is needed in dynamic development processes because of changing requirements or late detected errors.

### 2.2 Comparison with Workflow Meta-Models

In this section we outline the major differences between DYNAMITE and conventional workflow meta-models.

The DYNAMITE meta-model has been developed based on considerably *different requirements* compared to conventional workflow meta-models. It was specifically developed to support the enactment of dynamic development processes. *Concurrent engineering* and *cooperation* of engineers is supported by specific modeling constructs like simultaneous control flows and data flows. The support goes beyond the mere parallel execution of tasks. The *interleaved modeling and enactment* of development processes is supported. *Redo* of already terminated process parts and *feedback* to earlier tasks in the process are possible. *Traceability* of all changes to the management data is provided. The task

management data is *tightly integrated with the engineering artifacts* as well as the resource data of the process.

Altogether, the DYNAMITE meta-model *combines* elements of *project plans* and *workflow definitions*. On the one hand, a work breakdown structure, scheduling data and task dependencies like end-to-end relationships can be defined. On the other hand, the tasks are executable and information flow between tasks can be modeled.

DYNAMITE's capabilities for *modeling executable process definitions* and for automatically enacting these processes are limited. This has been an intentional design decision. Modeling constructs like *alternative* or *loop* have been deliberately left out of the meta-model for the following reasons: First, development processes are *highly creative* processes, which cannot be completely predefined in advance. Second, the resources involved in a development process are nearly *exclusively human resources*, which renders automation of tasks useless. Third, DYNAMITE was developed to support dynamic development processes on a *medium-grained level*. On this level one manually decides for a course of action and fixes the process steps to achieve a certain subgoal. Loops, where in each iteration a different course of action may be chosen, are not common. These kinds of scenarios are typical for *fine-grained personal processes* of individual engineers.

Here lie the strengths of conventional WfMS. Common workflow meta-models contain constructs for alternative branching and loops. But conventional workflow management systems are not suitable for the management of whole development projects. Usually, there are only the Sequence and the Parallel control flow types. The former is too restrictive to handle the simultaneous execution of activities, and the latter is too loose to handle cooperation scenarios. Furthermore, most conventional workflow meta-models do not provide constructs for the modeling of hierarchically structured development processes. For each subprocess in the project there would have to be a workflow definition before the subprocess could be enacted by a WfMS. This is often not feasible in a development project. Redo of process parts can only be simulated by dynamically adding copies of already terminated activities to a running workflow instance. In general, it might be possible to simulate all the needed functionality by using current dynamic workflow management systems, but this would not constitute suitable support for dynamic development processes.

Workflows and processes in general can be arranged on a *spectrum* according to their *degree of flexibility*. This has been done e.g. in [7]. On one end of the spectrum there are *completely predefined static workflows*, which cannot be modified at runtime, and on the other end there are *ad-hoc workflows* without any definition. In the latter case tasks are planned and ordered as needed during the enactment of the process, which is a common case in development projects. The common approach to add flexibility to static workflow management systems is to allow *deviations of running instances* from the workflow definition in the form of structural changes. The resulting flexible workflows have to be distinguished from ad-hoc workflows. The latter can be simulated by the former by

using nearly empty workflow definitions and elaborating them during runtime. But this shows the different concepts behind conventional WfMS and the DYNAMITE approach. The *dynamic changes* are considered as *exceptional cases*, because usually the workflow definition should already allow all common courses of action. In DYNAMITE, dynamic changes to the process model are considered as the *normal case*. The process evolves during enactment[3]. In that sense a dynamic process in our sense should be understood as an ad-hoc workflow in contrast to a flexible workflow.

Between completely predefined workflows and ad-hoc processes there are all *combinations* of static, flexible and ad-hoc workflows. These cases can be modeled by hierarchically structured processes, containing either predefined or ad-hoc workflows as subprocesses.

While DYNAMITE has proven to be appropriate for the management of most medium-grained subprocesses in a development project, workflow support is needed for recurring personal processes of individual engineers and for structured procedures like e.g. a change management scenario. We try to overcome this minor deficiency of the DYNAMITE model by integrating PROCEED with our WfMS.

## 3 Integrating Static and Dynamic Process Parts

In our research project we built a full-fledged WfMS as an extension of the CAE-tool Comos by means of the Microsoft Windows Workflow Foundation (WF) [8]. Our WfMS implements all interfaces of the *workflow reference model* [9] of the *Workflow Management Coalition* and even allows for dynamic changes of workflow instances at runtime. Therefore, it is superior to many commercial WfMS and comparable to most of the research prototypes in the adaptive and evolutionary workflow fields (cf. Sect. 4). In this section we describe the integration of PROCEED with the WfMS.

The overall development process is *structured hierarchically* in accordance with the DYNAMITE meta-model. It forms a tree hierarchy of subprocesses. Each subprocess on each level of the hierarchy can either be *enacted manually* or *automatically* by means of a workflow. Figure 2 shows an example of a hierarchical process in an abstract notation. Workflow-managed subprocesses are depicted with rounded edges.

The example is taken from the domain of plant engineering, in which the CAE-tool Comos is used. The design process of a plant is divided into several phases like e.g. *Basic Engineering* and *Detail Engineering*. Several flowsheets have to be created like a process flow diagram (PFD) and a piping and instrumentation diagram (P&ID). Devices, which are placed on the flowsheets, have to be specified, which can be done according to a predefined procedure defined

[3] Here, process evolution is understood with a different meaning compared to continuous improvement of a process definition.

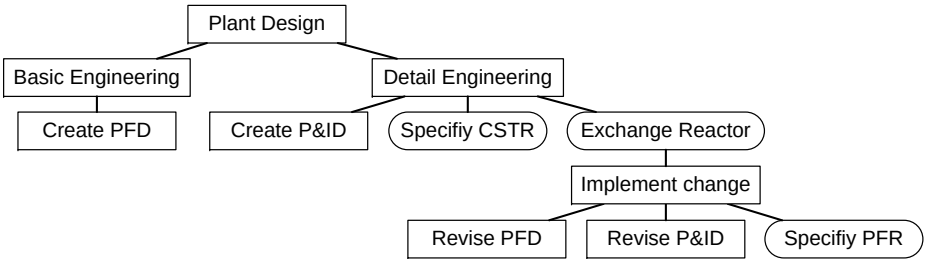


**Fig. 2.** Cutout of an example plant design process.

in the form of a workflow. There are different types of reactor devices like e.g. a continuously stirred tank reactor (CSTR) or a plug flow reactor (PFR).

While the flowsheets of the plant design are elaborated, it may occur that a task like Exchange Reactor in figure 2 is created. It defines that a certain CSTR should be replaced by a PFR in the plant design. For this purpose, a predefined *change management workflow* is started, which realizes the task Exchange Reactor. Whereas the *general procedure* for this change request is fixed, the *details are specific* for exchanging the CSTR in question, and they cannot be defined in general.

A simplified version of the change management workflow is depicted on the left hand side of Figure 3. It contains four activities[4], which define the necessary steps to handle a change request. The activity Implement change of this workflow instance contains tasks which are specific for the exchange of CSTR 47 by PFR 11. Therefore, the realization of the activity is a dynamic task net which is defined during the execution of the activity Plan change. The subtasks of Implement change together with the defined control flows form a typical engineering process. All tasks can be executed in parallel. Only the completion events of the tasks are restricted by simultaneous control flows, e.g. the revision of the P&ID must not be finished before the revision of the PFD, because the former depends on the latter. DYNAMITE provides the necessary means to model the subprocess Implement change, which would not be possible by means of the workflow management system. The strengths of the WfMS come into play, when there is a need for control structures like loops, here indicated by the backward arrow from Review change to Implement change. As long as the changes are not approved, some of the tasks of Implement change have to be re-iterated.

The process management component and the workflow management system are *tightly integrated* when it comes to the execution of workflows within a development process, which is illustrated by the following examples. When a DYNAMITE task, which should be controlled by a workflow, like Exchange Reactor becomes active, then a workflow instance is created and started via the WfMS. After this workflow instance has been successfully terminated, the task state is changed to Done. Within the workflow instance the control flow must reach Im-

[4] In the following, an element of a dynamic task net is called a *task* and the according element in a workflow definition is referred to as an *activity*.

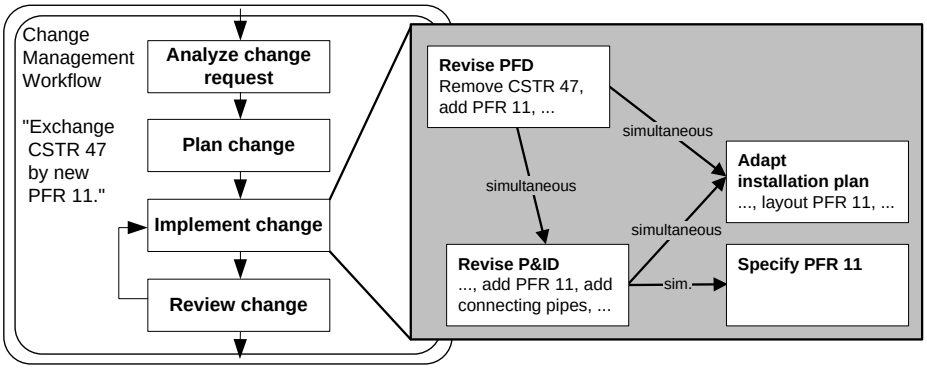


**Fig. 3.** Change management workflow for task Exchange Reactor.

plement change before the subtasks of this activity can be started. Furthermore, the workflow cannot proceed with Review change before the whole subprocess of Implement change has been successfully terminated.

If a DYNAMITE task is controlled by a workflow, then the workflow instance is not executed instead of a dynamic task net, but the *dynamic task net* which realizes the task *is controlled by the workflow instance*. We call the task a *workflow-managed task* and the task net containing its subtasks a *workflow-managed task net*. For each activity of the workflow instance there is a corresponding subtask in the task net. These subtasks are connected by sequential control flows according to the workflow definition and the current execution state of the workflow. When the workflow reaches a certain activity, the *execution state* of the corresponding task in the subnet is changed to *Waiting*, so that a human resource can start to work on this task (by changing its execution state to *Active*).

Altogether, the dynamic task net always reflects the *current state* of the running workflow instance. It comprises the *followed execution path* of the workflow instance as well as *future tasks*, which may be scheduled for execution.

*Loops* in the workflow are *unfolded* in the dynamic task net. The iterations of a loop are reflected by a corresponding sequence of tasks or – if the loop contains complex control structures – by a sequence of complex subnets. As soon as a new iteration starts, the according subtasks are inserted into the task net.

In case of an *alternative branching* construct the subnets for all branches are inserted into the task net, but they are not connected to preceding tasks. When one of the branches is entered during workflow execution, then control flows from the preceding tasks to the subnet corresponding to the selected branch are inserted. The neglected branches are removed after the termination of the workflow.

We do not go into details here, because the similar problem of creating project plans according to the execution of workflows has been investigated in related work already (cf. Sect. 4). Hence, the mapping between a workflow instance and a workflow-managed task net is not depicted graphically in this paper. For

example in fig. 3 only the change management workflow is shown but not the according workflow-managed task net.

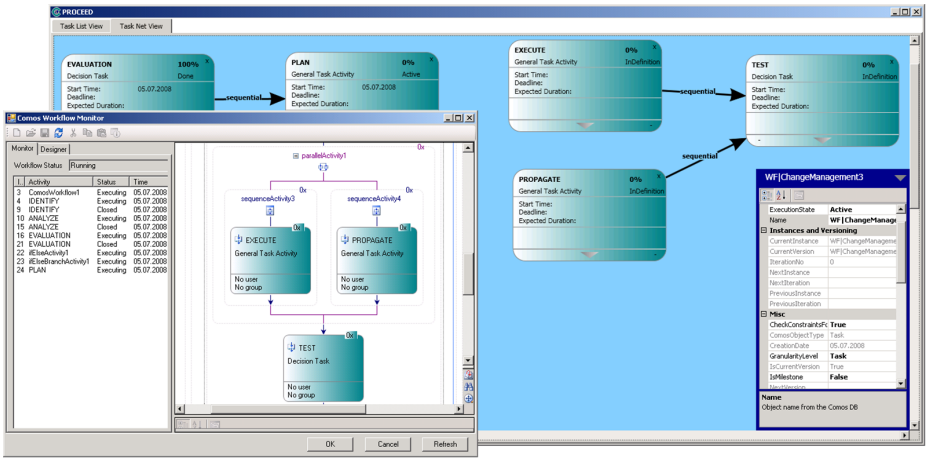


**Fig. 4.** Screenshots of PROCEED and the workflow monitor.

In contrast to a manual dynamic task net *a workflow-managed task net cannot be freely modified*. The creation and deletion of subtasks and control flows as well as certain state changes of subtasks are exclusively done by the WfMS, as it has been described above. When a task is controlled by a workflow, the WfMS serves as an *automatic manager* for this subprocess and replaces the human manager to some extent.

Manual changes to a workflow-managed task net can only be achieved by *changing the definition* of the *running workflow instance*. Our WfMS allows for this kind of dynamic changes. When the workflow definition is changed during the execution of a workflow, then the *corresponding task net* is *automatically adapted*. For example, when an activity has been removed from the workflow definition, its corresponding task is removed from the dynamic task net as well.

Figure 4 shows a screenshot of PROCEED and the workflow monitor of the WfMS. The task net view of PROCEED shows the dynamic task net, which is controlled by the monitored workflow instance. PROCEED currently comprises a work list view and a task net view. Views for resource management and project status analysis are currently under development. PROCEED has been integrated with MS Project. We use e.g. the Gantt-chart view of MS Project to present the coarse-grained tasks of the development project to the user. Changes made to the management data in either PROCEED or MS Project are directly reflected in the other system, respectively.

## 4 Related Work

There are two major research areas that are related to the contents of this paper. First, there are plenty of research works dealing with *flexibility* in processes that are supported by some workflow management system. Second, also the *integration of project and workflow management systems* is an issue that has been dealt with by diverse research groups.

*Flexibility.* In many domains there are processes which cannot be modeled completely before execution. Thus, lots of works have been devoted to the issue of process flexibility. Concerning this matter, Weber et. al. provide an overview in [10]. We consider the issues of schema evolution, version control and instance migration as relevant for standardized workflows in engineering design processes. However, the most important issues in this context are support for the enactment of ad-hoc processes and the traceability of changes. These issues were addressed by the development of dynamic task nets.

In [7] a classification of workflows regarding their degree of flexibility. The different workflow types range from ad-hoc workflow without any definition to completely predefined static workflows. The paper concentrates on business processes and office workflows supported by CSCW technology. We partly agree with the authors that ad-hoc workflows have been viewed as not worthwhile to be automated in the past. This is probably true for rather short-lived workflows. But the support of ad-hoc subprocesses as part of complex development processes has been studied for years [3].

Casati et al. advocate the automated handling of unforeseen process *exceptions* using an "exception-specification language" [11]. This language provides additional declarative set-oriented conditions which complement common imperative workflow definitions. Apparently, their approach targets at processes for which automated exception handling is feasible. Since development processes do not call for complete automation and are very creative, we think that changes are better done manually for these kinds of processes.

Within the ADEPT project Reichert et al. investigate dynamic changes to running workflow instances, which are represented by graphical "control flow graphs" [12]. At this, they use a graph based calculus with some optimizations to ensure the correctness of control flow graphs. Though support for dynamic changes is a crucial requirement for development processes, the ADEPT approach does not entirely match our needs. For example, ADEPT control flows are not suitable for modeling dependencies between task that are supposed to be executed in a "simultaneous engineering" mode. Furthermore, the ADEPT meta model rather focuses on the activity aspect of processes but does not yet deal with complex product and resource structures, which are also relevant in development processes.

*Integration of project and workflow management.* The approach described in this paper is closely related to the problem of integrating a WfMS with a project management system (PMS). In [13] the IPPM system is described which integrates features for both project and process management. An approach for

the unfolding of workflow control structures to tasks in a project plan is presented. We followed a similar approach, but we implemented a slightly different branching method, where neglected branches are not removed from the task net before workflow termination, and we do not need to insert estimation tasks for expected loop iterations because our workload and progress estimation for a workflow-managed task works independent of its subtasks in the task net.

In [14] Bussler discusses issues regarding the integration of WfMS and PMS in general. He distinguishes two parts: schema integration and behavior integration. Regarding schema integration, the main conflicts and different possible mappings are discussed. We encountered similar problems, since the elements of the DYNAMITE meta-model are closely related to the common concepts in project management, e.g. there is no conditional branching but an end-to-end task relationship. We decided to restrict the usage of modeling constructs in workflow-managed task nets. For the mapping of control structures we used the continuous mapping approach for loops and the static mapping approach for alternative branching. Regarding behavior integration, Bussler describes an ideal integration independent of any specific system. In our case the major problems arose from the peculiarities of the specific systems at hand.

Bauer addresses in [15] the issue of integrating existing workflow and project management systems. He distinguishes two approaches: loose coupling, where several workflow instances can be mapped to a single project task, and close coupling, where there is a one-to-one mapping of workflow activities and tasks in the project plan. The close coupling approach is not applicable, when the project plan and the workflows are on different abstraction levels. Hence Bauer presents a generic integration architecture for loose coupling based on an integration layer between the WfMS and the PMS. The propagation and aggregation of runtime data via the integration layer is specified by means of event-condition-action(ECA)-rules. In our research project we realized a loose coupling between PROCEED and MS Project and a close coupling between PROCEED and our WfMS. PROCEED serves as an integration layer between the WfMS and the PMS, but instead of using ECA-rules the connection between the project plan an the workflow instances is defined by a dynamic task net, which represents the whole development process.

## 5 Conclusion

In this paper we proposed DYNAMITE as an appropriate process meta-model for dynamic development processes. We compared it to conventional workflow meta-models and identified the respective strengths and weaknesses. To support the whole spectrum of dynamics in development processes, we integrated the PROCEED system with our conventional WfMS. A hierarchically structured process with interleaved static predefined and dynamic ad-hoc subprocesses can be enacted. The integrated system has been implemented as an extension of the commercial CAE-tool Comos of the innotec corporation. Altogether, we advanced research results of the CRC 476 IMPROVE [2] and transferred them to

industrial practice. The new process management functionality of Comos will be evaluated by selected customers of innotec.

## References


1. Heller, M., Nagl, M., Wörzberger, R., Heer, T.: Dynamic Process Management Based Upon Existing Systems. [2] 733–748
2. Nagl, M., Marquardt, W., eds.: Collaborative and Distributed Chemical Engineering: From Understanding to Substantial Design Process Support. Volume 4970 of LNCS. Springer, Berlin (2008)
3. Heller, M., Jäger, D., Krapp, C.A., Nagl, M., Schleicher, A., Westfechtel, B., Wörzberger, R.: An Adaptive and Reactive Management System for Project Coordination. [2] 307–373
4. innotec corporation: Website (2008) `http://www.innotec.de/en`.
5. Krapp, C.A.: An Adaptable Environment for the Management of Development Processes. PhD thesis, RWTH Aachen University, Aachen (1998)
6. Nagl, M., Westfechtel, B., Schneider, R.: Tool Support for the Management of Design Processes in Chemical Engineering. Computers and Chemical Engineering **27**(2) (2003) 175–197
7. Huth, C., Erdmann, I., Nastansky, L.: GroupProcess: Using Process Knowledge from the Participative Design and Practical Operation of Ad Hoc Processes for the Design of Structured Workflows. In: Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), IEEE Conference Publishing Services (2001)
8. Microsoft: Windows Workflow Foundation (2008) `http://msdn.microsoft.com/en-us/netframework/aa663328.aspx`.
9. Workflow Management Coalition: The Workflow Reference Model, `http://www.wfmc.org/`. (January 1995) Document Number WFMC-TC00-1003, Issue 1.1.
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.L., Sindre, G., eds.: Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007), Trondheim, Norway. Volume 4495 of Lecture Notes in Computer Science (LNCS)., London, Springer (June 2007) 574–588
11. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Database Systems **24**(3) (1999) 405–451
12. Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129
13. Chan, K., Chung, L.: Integrating Process and Project Management for Multi-Site Software Development. Annals of Software Engineering **14** (2002) 115–143
14. Bussler, C.: Workflow Instance Scheduling with Project Management Tools. In: DEXA '98: Proceedings of the 9th International Workshop on Database and Expert Systems Applications, Washington, DC, USA, IEEE Computer Society (1998) 753
15. Bauer, T.: Kooperation von Projekt- und Workflow-Management-Systemen. Informatik - Forschung und Entwicklung **19** (2004) 74–86