Process Evolution Support in the AHEAD System*

Markus Heller[1] and Ansgar Schleicher[2] and Bernhard Westfechtel[1]

[1] Lehrstuhl für Informatik III, RWTH Aachen
Ahornstrasse 55, D-52074 Aachen
[heller|bernhard]@i3.informatik.rwth-aachen.de
[2] DSA Daten- und Systemtechnik GmbH
Pascalstr. 28, D-52076 Aachen
Ansgar.Schleicher@dsa.de

Abstract. Development processes are inherently difficult to manage. Tools for managing development processes have to cope with continuous process evolution. The management system AHEAD is based on long-term experience gathered in different disciplines (software, mechanical, or chemical engineering). AHEAD provides an integrated set of tools for evolving both process definitions and their instances. This paper describes a demonstration of the AHEAD system which shows the benefits of process evolution support from the user's point of view.

1 Introduction

Development processes in different disciplines such as software, mechanical, or chemical engineering share many features. Unfortunately, one of these common features is that they are hard to manage. Development processes are highly creative and therefore can be planned only to a limited extent. In this paper, we present the comprehensive evolution support [1] offered by *AHEAD* [2], an *A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*evelopment Processes. The tool demonstration given here complements the conceptual paper in this volume [3]. AHEAD is based on nearly 10 years of work on development processes in different engineering disciplines. So far, we have applied the concepts underlying the AHEAD system in software engineering, mechanical engineering, and chemical engineering. In contrast to the conceptual paper, our tool demonstration will refer to the chemical engineering domain.

2 Process Evolution

For managing evolving development processes, AHEAD offers *dynamic task nets* [4]. Within dynamic task nets, tasks describe units of work to be done. Tasks are organized hierarchically, i.e., a task may be decomposed into a set of subtasks. Control flow relationships define the order of subtask enactment. Data flow relationships connect input and output parameters of tasks and allow to exchange documents between them.

* The work presented in this paper was carried out in the Collaborative Research Center IMPROVE which is supported by the Deutsche Forschungsgemeinschaft.

Feedback relationships model cycles within the process which may stem from planned iterations or occurring exceptional situations. Software processes evolve continuously and it is usually impossible to completely plan a process before its enaction. Therefore, dynamic task nets are designed to be editable and enactable in an interleaved fashion.

With respect to dynamic task nets, we have to distinguish between process definitions and process instances. A *process instance* represents a specific development process being executed. In contrast, a *process definition* describes a whole class of processes. It describes the structures of processes of this class, and the constraints that have to be met. Process instances refer to definitions; e.g., the task `ImplementFoo` might be an instance of the task type `Implement`.

Process evolution is supported by the AHEAD system in the following ways:

- *Instance-level evolution.* Planning and enactment of dynamic task nets may be interleaved seamlessly.
- *Definition-level evolution.* Process knowledge at the definition level may evolve as well. Evolution is supported by version control at the granularity of packages (modular units of process definitions).
- *Bottom-up evolution.* By executing process instances, experience is acquired which gives rise to new process definitions. An inference algorithm supports the semi-automatic creation of a process definition from a set of process instances.
- *Top-down evolution.* A revised process definition may be applied even to running process instances by propagating the changes from the definition to the instance level.
- *Selective consistency control.* By default, process instances are required to be consistent with their process definition. However, the process manager may allow for deviations resulting in inconsistencies. These deviations are reported to the process manager who may decide to reinforce consistency later on (e.g., when an improved process definition is available).

3 AHEAD System

Figure 1 displays the *architecture* of AHEAD. The tools provided for different kinds of users are shown on the right-hand side. “Process modeler”, “process manager”, and “developer” denote roles rather than persons: A single person may play multiple logical roles, and a single role may be played by multiple persons. The left-hand side, which will not be discussed further here (see [3]), shows internal components of the AHEAD system which are not visible at the user interface. Furthermore, the horizontal line separates definition and instance level.

The process modeler uses a commercial CASE tool — *Rational Rose* — to create and modify process definitions in the UML [5]. Rational Rose is adapted with the help of stereotypes which link the UML diagrams to the process meta model (dynamic task nets). The process definition introduces domain-specific types and constraints for operating on the instance level. Definition-level evolution is supported through version control for model packages.

On the instance level, the *management tool* assists a process manager in planning, analyzing, monitoring, and controlling a development process. The management tool

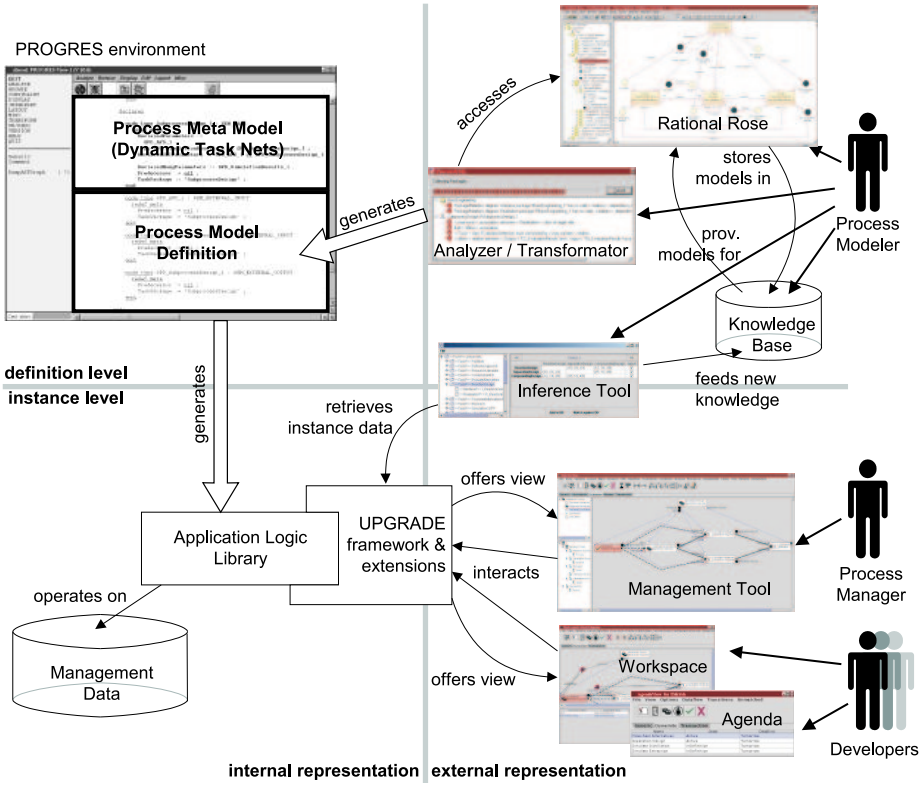

Fig. 1. Architecture of the AHEAD system

support instance-level evolution (through dynamically evolving task nets), consistency control (with respect to the process definition generated from the UML model), and top-down evolution (migration of a process instance to a new version of the process definition). The management tool is coupled with tools provided to developers (or designers) which are used to display *agendas* of assigned tasks and to operate on these tasks in *workspaces* from which domain-specific tools may be activated in order to work on design documents.

Finally, the *inference tool* closes the loop by assisting in the inference of process definitions from process instances. The inference tool analyzes process instances and proposes definitions of task and relationship types. These definitions are stored in a knowledge base which may be loaded into Rational Rose. In this way, bottom-up evolution is supported (for a more detailed description, see [1].

4 Demonstration

AHEAD is being developed in the context of IMPROVE [6], a long-term research project which is concerned with models and tools for design processes in *chemical*

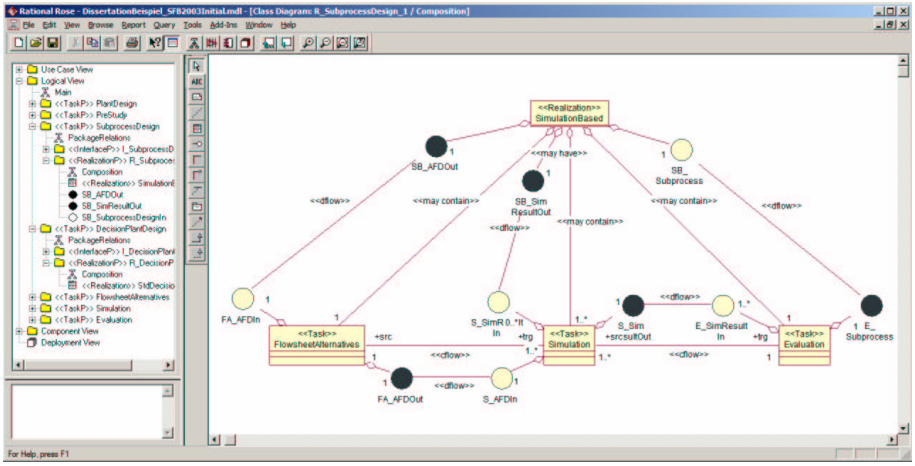

Fig. 2. Adapted UML class diagram for defining a chemical engineering design process

engineering. Within the IMPROVE project, a reference scenario is being studied referring to the early phases (conceptual design and basic engineering) of designing a plant for producing Polyamide6 [7]. To a large extent, the requirements for process evolution support were derived from this scenario, even though we also studied processes in other domains (e.g., software engineering). In fact, the reference scenario constitutes a fairly challenging benchmark against which process evolution capabilities of management systems can be evaluated. Process knowledge is incomplete, instable, and rather fuzzy, feedback occurs frequently in the design process, multiple design variants have to be considered, etc.

Based on our work on the reference scenario, we have prepared a demo session which is sketched briefly below. First, the process modeler creates a process definition for design processes in chemical engineering. The process definition describes design processes at the type level. Figure 2 shows a class diagram which defines a part of the overall design process. The described part consists of one task for designing a set of flowsheet alternatives, a set of simulation tasks (one for each alternative), and one evaluation task for selecting the best alternative.

The design process is planned and executed according to this process definition. On the top level, the design process is essentially decomposed according to the structure of the chemical process, which consists of three steps (reaction, separation, and compounding). We will focus on the design of the separation, which requires input from the reaction design. The process manager insert tasks for elaborating flowsheet alternatives and for performing a final evaluation of the alternatives. So far, the task net is still incomplete because the simulation tasks have not been instantiated yet.

Now, the process manager detects the need for performing a task which has not been anticipated in the process definition. To insert this task (a task for estimating the inputs to a certain component of the chemical plant in order to accelerate the overall design process by concurrent engineering), consistency enforcement is switched off. The task

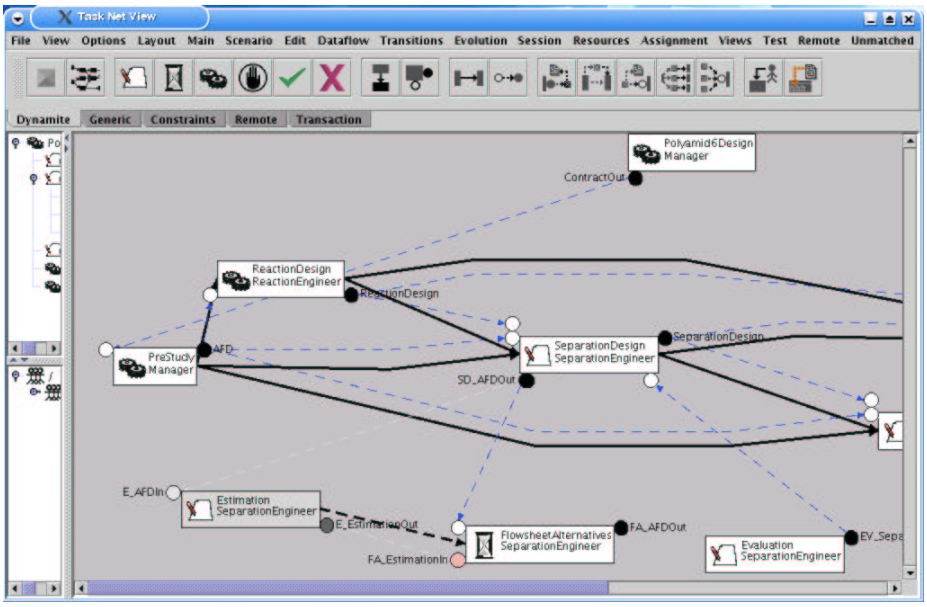

Fig. 3. Highlighting of inconsistencies in the management tool

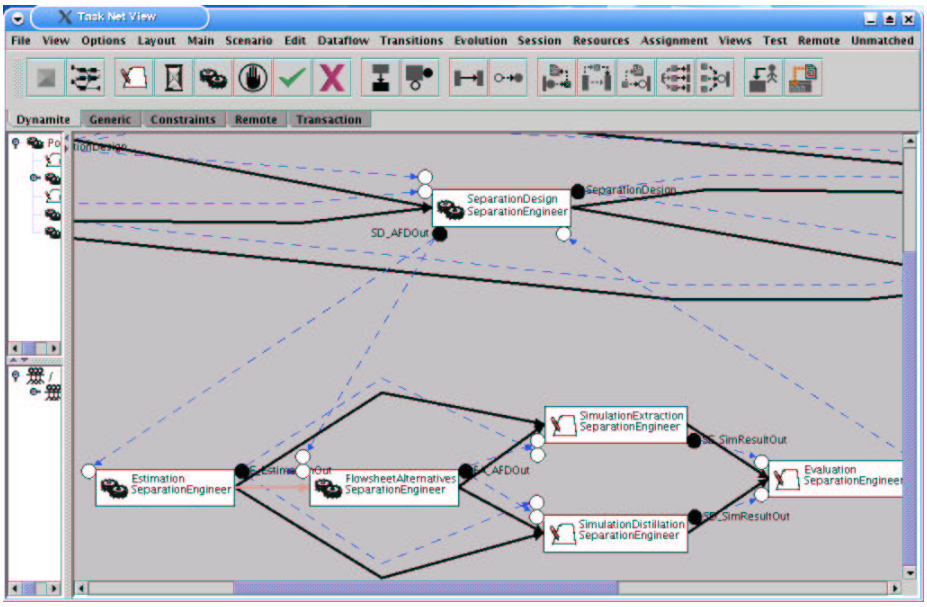

Fig. 4. Task net after migration

net is extended, resulting in inconsistencies which are displayed by colored markings (Figure 3).

The process definition is extended accordingly by introducing a new package version (not shown in a figure). In this way, traceability on the definition level is maintained. The corresponding class diagram differs from the previous version in the definition of a task class for estimation tasks.

The modified process definition is propagated to the process instance. In this way, running process instances may be migrated to improved definitions. In general, migration has to be performed in a semi-automatic way (i.e., some steps may be automated, but some have to be performed interactively by the process manager). Figure 4 shows a state of the task net which is nearly consistent with the improved definition. The control flow between the estimation task and the design task is still inconsistent: Although it has been defined as a sequential control flow, source and target are active simultaneously. This inconsistency will be removed when the estimation task is terminated. After that, consistency enforcement may be switched on again. This final step closes the process evolution roundtrip.

5 Conclusion

We have presented a management system supporting comprehensive evolution support for dynamic development processes. AHEAD provides *round-trip process evolution*, combining instance-level evolution, definition-level evolution, bottom-up and top-down evolution as well as toleration of inconsistencies into a coherent process evolution framework. We have applied our system to an industrially relevant reference scenario. In the future, we hope to transfer the implementation into industrial practice, resulting in feedback from practical use.

References

1. Schleicher, A.: Management of Development Processes — An Evolutionary Approach. Deutscher Universitäts-Verlag, Wiesbaden, Germany (2002)
2. Jäger, D., Schleicher, A., Westfechtel, B.: AHEAD: A graph-based system for modeling and managing development processes. [8] 325–339
3. Heller, M., Schleicher, A., Westfechtel, B.: Graph-based specification of a management system for evolving development processes. In Nagl, M., Pfaltz, J., eds.: Proc. AGTIVE 2003. LNCS, Springer (2004) In this volume.
4. Heimann, P., Krapp, C.A., Westfechtel, B., Joeris, G.: Graph-based software process management. Int. Journal of Software Engineering and Knowledge Engineering **7** (1997) 431–455
5. Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison Wesley, Reading, Massachusetts (1998)
6. Nagl, M., Marquardt, W.: SFB-476 IMPROVE: Informatische Unterstützung übergreifender Entwicklungsprozesse in der Verfahrenstechnik. In Jarke, M., Pasedach, K., Pohl, K., eds.: Proc. Informatik '97. Informatik aktuell, Aachen, Springer-Verlag (1997) 143–154
7. Nagl, M., Westfechtel, B., Schneider, R.: Tool support for the management of design processes in chemical engineering. Computers & Chemical Engineering **27** (2003) 175–197
8. Nagl, M., Schürr, A., Münch, M., eds.: Proc. AGTIVE '99. LNCS 1779. Springer (1999)