Integrated Design and Process Technology, **IDPT-2003**
Printed in the United States of America, June, 2003
©2003 Society for Design and Process Science

A Management System for Evolving Development Processes

Markus Heller, Ansgar Schleicher, Bernhard Westfechtel
Computer Science III, Aachen University of Technology, D-52056 Aachen
(heller—schleich—bernhard)@i3.informatik.rwth-aachen.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. Furthermore, changes at the definition level may be propagated to the instance level and vice versa (round-trip process evolution). Finally, AHEAD deals with both incomplete and incorrect process knowledge by supporting untyped process instances and allowing for deviations of process instances from their definitions, respectively.

I. 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. The tasks to be performed depend on the product to be developed, which is not known in advance. Alternative designs (variants) are explored to arrive at an optimal solution. Feedback may occur frequently — including not only spontaneous feedback raised by design errors in earlier steps, but also anticipated feedback which may be used to improve the design or to select among variants of the design. Finally, development methods such as concurrent or simultaneous engineering require sophisticated coordination between inter-dependent design activities.

These considerations illustrate that development processes are highly dynamic. *Process evolution* constitutes a major challenge for effective management. Development processes cannot be fixed beforehand, rather they undergo continuous evolution. Management has to cope with process evolution in an adequate way by balancing flexibility and control. On one hand, management has to react to changes, e.g., by re-planning the development process in response to detected errors. On the other hand, management has to constrain changes in order to ensure that the development process is executed in a well-structured manner.

In order to build effective tools for managing development processes, one must face the challenge of process evolution. While this has been recognized widely, current *management systems* can cope with process evolution only to a limited extent. In particular, this applies to workflow management systems [19] which were designed for repetitive business processes, e.g., by automating routine work in banks, insurance companies, administrations, etc. In such systems a high number of workflows are executed according to a common definition, ensuring that work is performed following a pre-defined procedure. This approach cannot be transferred to development processes because it does not take process evolution into account: Developers would perceive themselves being tied in a straight-jacket so that they cannot perform their creative work as desired.

In response to these problems, a variety of mechanisms have been developed to increase the flexibility of workflow management systems. Workflow definitions are made more flexible, e.g., by relaxing ordering constraints for activities, handling exceptions, or allowing for (usually small) deviations. Moreover, mechanisms are provided for changing workflow definitions and instances. Despite these efforts, process evolution is hardly considered comprehensively, and changes are still rather difficult to implement.

In this paper, we present the comprehensive evolution support [27] offered by *AHEAD* [13], an *A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*evelopment Processes. 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 our work, we have developed an interdisciplinary approach to the management of development processes which particularly takes process evolution into account. Process evolution is supported in an integrated way by a variety of cooperating mechanisms such as wide spectrum process definitions, top-down propagation of changes from process definitions to their instances, instance-level evolution (by seamless interleaving of planning and execution), (selective) toleration of inconsistencies of process instances with respect to their definition, and bottom-up inference (learning) of process definitions from executed instances.

Below, we proceed as follows: Section 2 presents the conceptual framework underlying AHEAD's evolution support. Section 3 describes the functionality of the AHEAD system with the help of a case study in chemical engineering (space limitations do not allow to cover other engineering disciplines as well). Section 4 briefly sketches architecture and implementation of AHEAD. Section 5 discusses related work, and Section 6 concludes the paper.

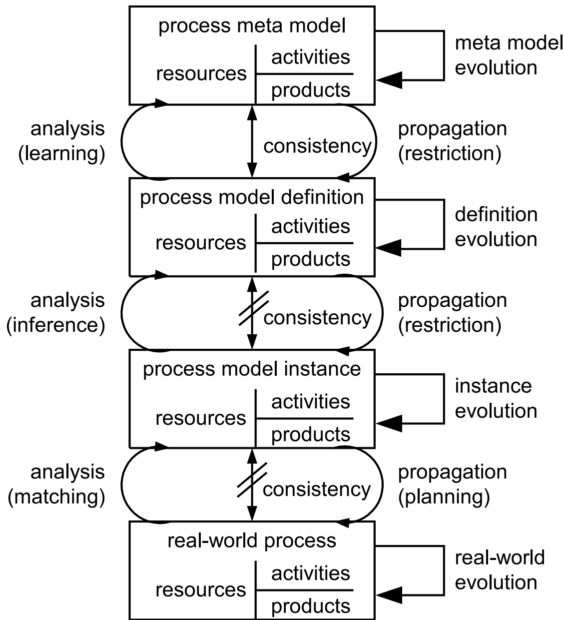

Fig. 1. Conceptual framework

II. Conceptual Framework

A. Levels of Modeling

Our work is based on a conceptual framework which distinguishes four *levels of modeling* (Figure 1). These levels are quite common in software process research [20], [6]; however, the meaning of these levels as well as their interaction differs from one approach to another. Each level deals with process entities such as products, activities, and resources. Here, we focus on activities, even though our framework equally applies to products and resources [18]. The level of abstraction on which these entities are considered decreases from the top to the bottom. Process evolution may occur on every level. Furthermore, adjacent levels are connected by propagation and analysis relationships. Propagation is performed top-down and constrains the operations that may be performed on the next lower level. Conversely, analysis works bottom-up and aims at providing feedback to the next upper level.

The *process meta model* introduces the language (or meta schema) in terms of which process models may be defined at the next lower level. In AHEAD, the activity meta model is based on *dynamic task nets* [9]. Tasks are organized hierarchically. Complex tasks are decomposed into nets of subtasks; atomic tasks form the leaves of the hierarchy. Tasks are connected horizontally by control flows, which determine the order of task execution. Feedback flows are oriented oppositely to control flows; they are used to represent feedback in the development process. Finally, tasks have inputs and outputs which are connected by data flows.

Process (model) definitions are created as instances of process meta models. Process models are defined in the Unified Modeling Language (UML [3]). Tasks and task types are modeled as objects and classes, respectively. At the type level, processes are represented by class diagrams. In addition, collaboration diagrams define processes at the abstract instance level. In this way, recurring patterns of tasks and instances may be defined. Process definitions are organized into packages. A task package defines the interface of a task (in terms of its inputs and outputs), while a realization package (of a complex task) contains the class diagram and the collaboration diagrams of the respective subprocess. UML model elements are adapted to the process meta model with the help of extension mechanisms provided by the UML (stereotypes and tagged values [14]).

Process (model) instances are instantiated from process model definitions. A process model definition represents reusable process knowledge at an abstract level. Multiple process instances may share the same definition. In contrast, there is a 1:1 correspondence between a process model instance and the respective real-world process, i.e., each process model instance represents exactly one real-world process. A process model instance is composed of task instances which are created from the task classes provided by the process model definition.

Finally, the *real-world process* consists of the steps that are actually performed by humans or tools. The process model instance is an abstraction which represents the real-world process in the process management system. The process model is used to guide and control process participants. Conversely, process participants provide feedback which is used to update the process model instance.

B. Evolution Support

Evolution may occur on each model level. Furthermore, the consequences of evolution have to be propagated vertically between the levels. In the sequel, we describe process evolution support given by the AHEAD system at a conceptual level, proceeding from the bottom to the top.

Real-world evolution drives evolution of the upper levels. The real-world process is represented by a corresponding process model instance. When the real-world process is changed, its model has to be updated accordingly. In AHEAD, these updates are performed by and large manually. For example, a designer may indicate that he has finished a certain design task by triggering the transition Commit from state Active to state Done. Conversely, AHEAD provides mechanisms to control the real-world process. For example, for a design task a workspace is maintained which contains the design documents to read and written. These documents are owned by AHEAD, i.e., they may be accessed only via the workspace. In this way, AHEAD tries to maintain consistency between the process model instance and the real-world process.

Instance-level evolution is crucial to match the process model instance with the real-world process as closely as possible. Instance-level task nets are built up and modified

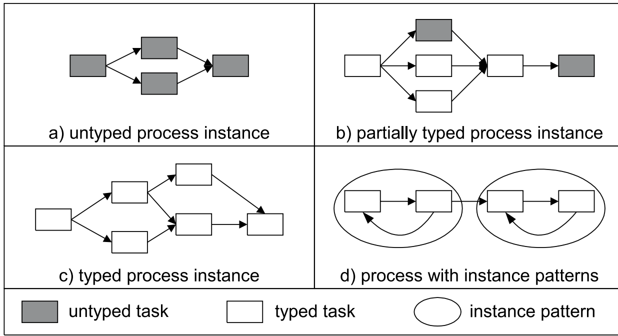

Fig. 2. Wide spectrum approach

at run time by instantiating tasks from their classes in the process model definition. Therefore, the topology of task nets is determined only at run time. Planning and execution may be interleaved seamlessly. In contrast to project plans known from project management systems, task nets may represent feedback in the development process. Handling of feedback may imply reactivation of already terminated tasks in order to propagate changes through the task net. Finally, AHEAD provides simultaneous engineering by control flows which allow for overlapping execution of predecessor and successor tasks. Preliminary versions of outputs may be released to and consumed by successor tasks, resulting in a dynamic workspace which is updated according to changes in the context of a task.

Instance evolution is controlled with the help of process model definitions. A *wide spectrum approach* allows to adjust control as desired. How stringently process models are defined, depends on the available process knowledge and the desired balance of flexibility and control. The spectrum of process model definitions is illustrated in Figure 2. If no domain-specific process knowledge is available at all, task nets may be composed from untyped tasks. More precisely, this means that all tasks are instantiated from the same predefined type Task. In this case, only built-in constraints of the process meta model are enforced. If domain-specific process knowledge is available, yet incomplete, a partially typed process instance may be created. Here, the known parts of the process are instantiated from domain-specific types explicitly defined in the model; for the rest, untyped tasks are used. In the case of complete process knowledge at the type level, a typed process instance is created in line with a class diagram defining task classes and associations. Finally, the process modeler may define instance patterns for frequently occurring subgraphs of tasks and relationships (e.g., a control flow — feedback pattern). Instance patterns are defined by collaboration diagrams which describe the operations to be executed in order to insert an instance of a pattern into a task net.

The wide spectrum approach allows to balance flexibility and control as desired. For different parts of the overall process, subprocess models may be defined at the different levels illustrated in Figure 2, according to the knowledge which is available for the respective subprocess. However, the flexibility provided by the wide spectrum approach alone is not sufficient and must be complemented by further mechanisms. No matter how carefully a process model is defined: A process definition may contain errors, i.e., it may turn out to be inadequate because of missing task or relationship types, inappropriate ordering constraints, etc. Therefore, AHEAD provides for flexible consistency control by allowing for *inconsistencies* of process model instances with respect to their definitions. The level of consistency enforcement may be controlled individually for each subprocess (refinement of a complex task). If the manager permits inconsistencies, the respective part of the overall process model instance may deviate from the definition (e.g., a task class may be instantiated which is not contained in the respective class diagram). The manager is informed about all occurring inconsistencies. Process execution may continue even in the presence of inconsistencies, which may (or may not) be removed later on.

Toleration of inconsistencies avoids the well-known problem of getting stuck in the execution of an erroneously defined workflow. Execution may continue even in the case of an inadequate process model definition. However, it is rarely acceptable to live with inconsistencies forever. Thus, *definition evolution* has to be supported to improve process model definitions. In AHEAD, process models may be evolved in terms of packages, which serve as units of version control. Versions are immutable to ensure traceability. In order to evolve a package, a new version is created. This may induce the creation of new versions of related packages. To make use of a new version of an interface package, a version of an importing realization package has to be created likewise.

Changes at the definition level may be propagated top-down to the instance level. That is, instances may be *migrated* to new definitions. Migration is handled in a flexible way in multiple respects. The user (manager) determines the time of migration, the scope (which instances to migrate), and the target (which versions of definitions to use). Furthermore, inconsistencies may be tolerated during migration. Migration can always be performed, even if new errors are introduced or old errors persist. This provides the manager with the flexibility to clean up the task net step by step. Please note that the overall migration process can be automated only partially; human expertise is required to perform essential decisions.

After having introduced fairly comprehensive mechanisms for process evolution support, we conclude this section by mentioning a limitation which still exists: *Meta model evolution* is not supported so far. The process meta model is fixed (as usual). In general, changes to the meta model would invalidate the tools provided as well as the models maintained by the AHEAD system.

III. Example

After having introduced the conceptual framework underlying the AHEAD system, we present an example which is drawn from the chemical engineering domain. AHEAD is being developed in the context of IMPROVE [22], a long-term research project which is concerned with models and tools for design processes in *chemical 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 [24]. 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.

In the sequel, we present a sample process which is based on the Polyamide6 scenario. The example is chosen such that essential process evolution capabilities of the AHEAD system is demonstrated. The overall design process being studied is much more comprehensive; see [24]. The example below shows a *process evolution roundtrip*: During the execution of the design process, changes are performed which introduce inconsistencies with respect to the process definition. In response to this problem, an improved version of the process definition is created. Finally, the process instance is migrated to the new definition.

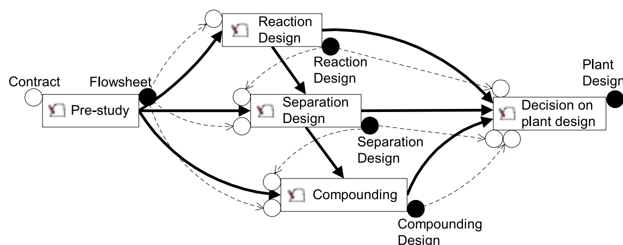

Fig. 3. Top-level task net for the design process

Figure 3 shows the *top-level task net* for the Polyamide6 design process on the instance level. Each task is represented by a box containing its name, its type, and an icon for the current state of execution. Since execution has not started yet, all tasks are still in their initial state. White and black circles stand for inputs and outputs, respectively. Solid and dashed arrows visualize control and data flows, respectively. The design process start with a pre-study in which requirements are gathered and an initial flow sheet for the chemical process is drawn (Figure 4). After that, the steps of the chemical process are studied in greater detail. The main steps are: reaction of monomers, separation of polymers from monomers, which are fed back into the reaction phase, and compounding, which is concerned with fine-tuning the properties of the produced material. After having designed reaction, separation, and compounding, an evaluation step follows to determine whether the requirements to the chemical process are met. Otherwise, the design created so far has to be revised.

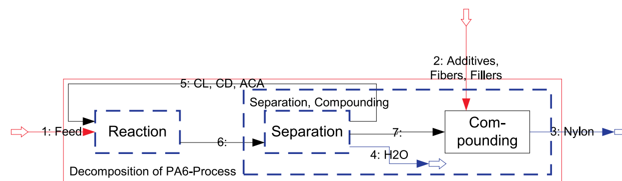

Fig. 4. Flow sheet for the chemical process

Below, we focus on a certain subprocess, namely *separation design*. Before we continue to examine the evolution of this subprocess instance, we take a look at the process definition. Figure 5 presents a definition of a subprocess design as it can be used for any part of the overall chemical process (i.e., not only for the separation, but also for the reaction and the compounding). The subprocess design is defined in two UML packages for the interface and the realization, respectively. Both are defined on the type level with the help of UML class diagrams which are adapted to the underlying process meta model by *stereotypes*. Partly, stereotypes are represented textually; e.g., a task class is decorated with the string <<Task>>. Partly, new graphical symbols are introduced to enhance the visualization (white and black circles for classes of input and output parameters, respectively). In addition to stereotypes, further meta data are represented by *tagged values* which are used to annotate model elements. In the figure, tagged values are attached as notes to model elements.

The *interface* is defined in terms of inputs and outputs. A task of class SubprocessDesign receives a flow sheet for the overall chemical process and the results of related subprocesses. The output parameter denotes the result of the subprocess design, including the flow sheet for the subprocess, simulation models, and simulation results.

The *realization* is described by a class diagram containing a class for the respective task net as well as classes for the subtasks. In general, multiple realizations may be defined. Here, we discuss only a simulation based realization (in contrast to a realization based on laboratory experiments). The tag Partial is used to distinguish between partially and completely typed processes (Figure 2b and c, respectively). The value false excludes the insertion of untyped tasks and relationships.

The class diagram of Figure 5 introduces three *classes* of subtasks. FlowsheetAlternatives, which is instantiated exactly once, is used for creating design variants for the respective subprocess. This task is followed by multiple simulation tasks, each of which deals with one design vari-

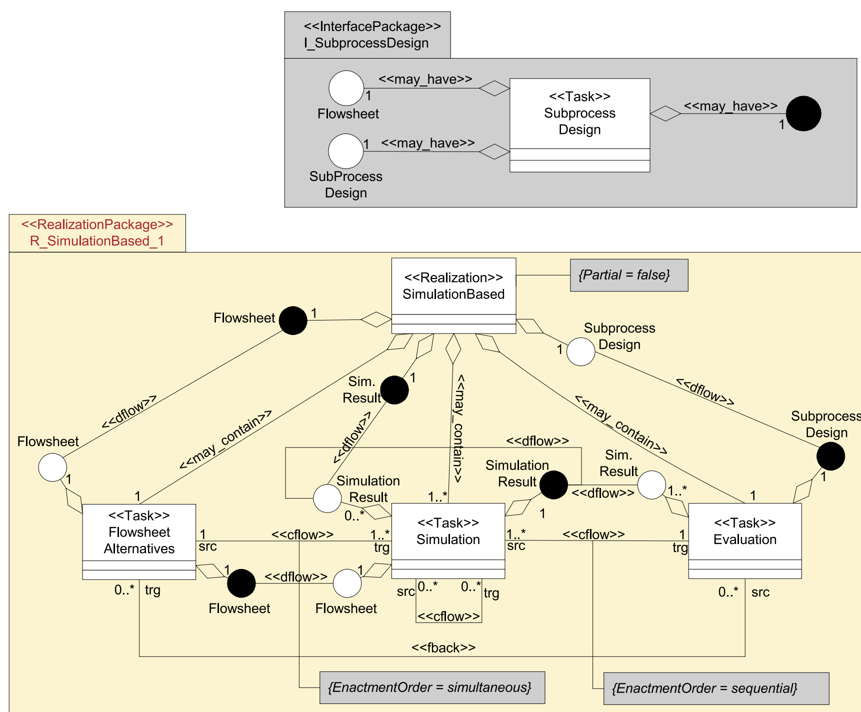

Fig. 5. Class diagram for a design subprocess

ant. Finally, the process ends with exactly one Evaluation task which examines the simulation results and composes the overall subprocess design.

Control flow associations constrain the order of task execution. Their behavior is defined by the tag EnactmentOrder. In our example, simulation tasks may overlap with the task of designing flow sheet alternatives, which is briefly called design task below (enactment order simultaneous). Thus, it is possible to start simulation of a design variant before the final flow sheet containing all variants is available. Multiple simulation tasks may be executed in parallel unless mutual control flows constrain the order of execution (not used in our example). Finally, a control flow association is defined between the class Simulation and the class Evaluation. Here, a sequential order is enforced since it only make sense to perform the final evaluation when all design variants have been investigated. If the final evaluation identifies a problem that needs to be fixed, a feedback flow may be created back to the design task, implying that the design has to be improved and investigated by new simulations.

Finally, the class diagram defines *data flow associations* between parameter classes. The design task receives the flow sheet as input as well as evaluation results (in case of feedback). It creates an extended flow sheet containing the design variants for the subprocess under study. The extended flow sheet is passed to the simulation tasks, which may also receive results from other simulations. These results are received either from the parent task (vertical communication) or from preceding simulations in the same task net. In the former case, the simulation results refer to related subprocesses (e.g., the separation design depends on simulation results from the reaction design). Finally, all simulation results are passed to the evaluation task, which in turn delivers the overall result of the subprocess design to the parent task.

Figure 6 presents a snapshot of the subprocess SeparationDesign which is currently being planned. The task is consistent with the process definition explained above, but its not yet complete. From the cardinalities in the class di-

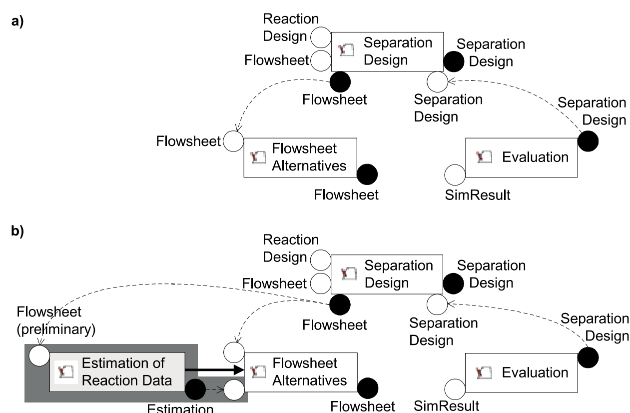

Fig. 6. Early snapshots of the separation design process

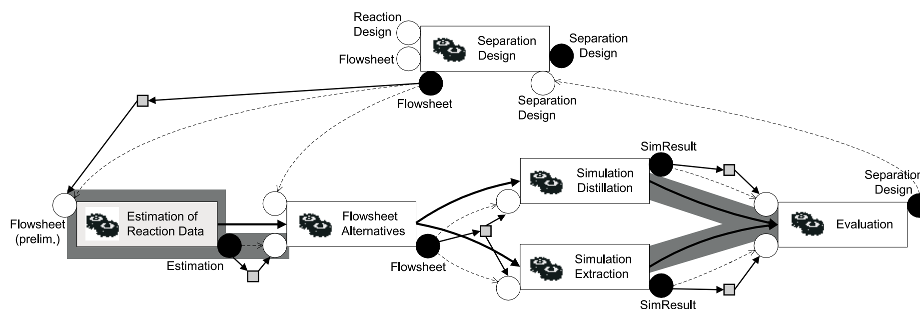

Fig. 7. Task net extend with simulation tasks

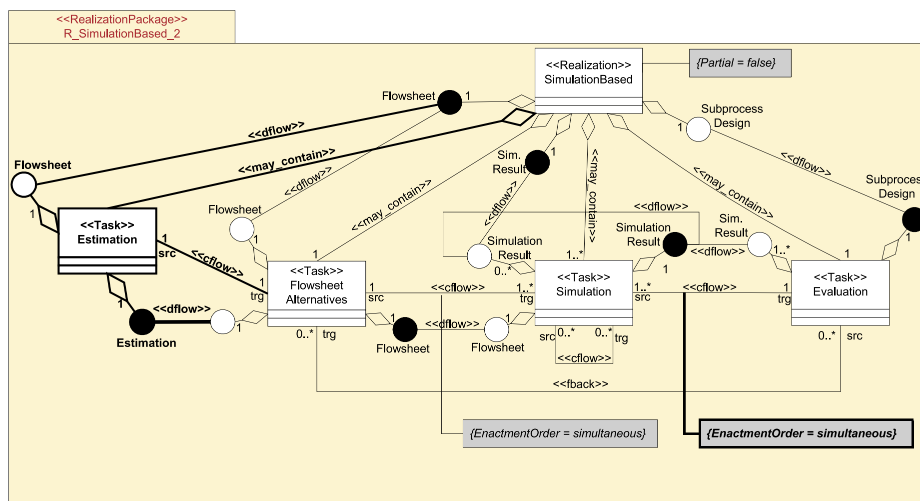

Fig. 8. Revised process definition

agram, it is known that exactly one design task and exactly one evaluation task have to be performed. These tasks have already been inserted into the task net. The simulation tasks have not been instantiated so far because the design variants have not been elaborated yet.

Now, the following problem is recognized: In order to design the separation, the flow sheet alone is not sufficient. In addition, certain data on the reaction are required, e.g., flow rates and temperature of the substances fed into the separation step. These data will eventually be delivered when the reaction design is elaborated, but waiting for them would severely slow down the design process. Therefore, the manager of the separation design calls for an initial estimation of these data. Accordingly, an estimation task is inserted. Design may proceed using initial estimations until more detailed data finally arrive from the reaction design. In this way, *simultaneous engineering* is introduced to accelerate the design process.

The resulting task net is shown in Figure 6b. Since the estimation task is not defined in the class diagram, the manager makes use of the pre-defined standard type Task to insert it into the task net as an untyped task. This modification results in an inconsistency which is signaled graphically. Likewise, the control and data flows emanating from the estimation task are marked as inconsistent, as well as the new input parameter of the design task.

Execution may continue even when the task net contains inconsistencies. Design variants are elaborated, and corresponding simulations are performed. Figure 7 shows a snapshot of the extended task net. The task net contains two simulation tasks for investigating the separation variants distillation and extraction. These simulation tasks are supplied with the estimation results via the flow sheet, which the designer has annotated with these data. So far, the simulations have been performed based on the estimation results; the simulation results from the reaction design are not yet available and will be considered later.

Although the simulation results are still preliminary, the manager of the reaction design decides to call for an initial evaluation to accelerate the design process. In this way, feedback may be provided early, potentially triggering rework of the separation design. However, this violates the conservative policy fixed in the process definition. Since inconsistencies are allowed, the evaluation task may be activated, but the sequential control flows are marked as behaviorally inconsistent.

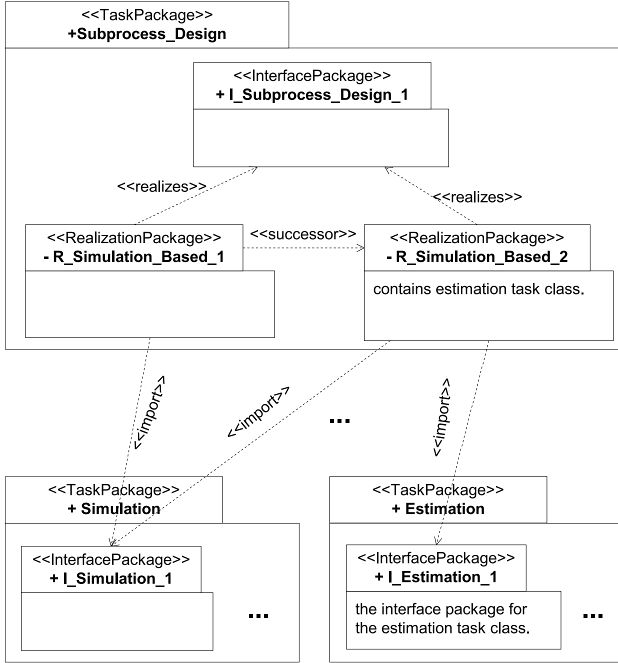

Fig. 9. Package versions

At this stage, it is decided to clean up the process definition so that it includes the process improvements which have already been practiced at the instance level. Since the old definition may not be modified for the sake of traceability, a new version is created instead. Among others, the class diagram presented in Figure 5 is revised (see Figure 8, where the changes are emphasized in bold face). A task class Estimation is inserted and embedded into its context. This implies that subsequent tasks have to be augmented with a new input parameter. To this end, new versions of these task classes (and their enclosing packages) have to be created as well. Finally, the enactment behavior of the control flow association from Simulation to Evaluation is relaxed to allow for simultaneous activation.

Figure 9 illustrates the evolution on the definition level by a *package diagram*. A task package serves as a container for interface and realization packages. The interface package for the subprocess design is not affected. For the realization, a new package version is derived from the old one. In addition, a new task package for the estimation is created.

The process evolution roundtrip is closed by propagating the changes at the definition level to the instance level. In our example, this will result in a task net as previously shown in Figure 7, which, however, uses the new definition and does not contain inconsistencies any more. In general, migration has to be performed interactively: It is not always possible to determine the target type of migration uniquely, migration may require structural changes which may not be performed automatically (e.g., insertion of new obligate tasks), the user has to control the scope of migration, etc.

In our example, it appears at first glance as if all migration steps could be performed automatically after a new type version has assigned SeparationDesign. Unfortunately, this is not the case. Since we want to avoid user-programmed migrations, the AutoMigrate command offered by AHEAD reasons at a generic level. All tasks whose types were already contained in the old definition can be migrated automatically to the new type version. In our example, this rule applies to the design task and the simulation tasks. However, the estimation task cannot be migrated automatically. Since it was introduced as an untyped task, its target type cannot be determined uniquely. This argument also applies to its input and output parameters as well as parameters of related tasks. After all objects have been migrated, the relationships can be migrated automatically. This is possible even for untyped relationships (such as the control flow between the estimation task and the design task) provided that there is only one matching relationship for each pair of object types.

IV. REALIZATION

Figure 10 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 right-hand side 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. Rational Rose is adapted with the help of stereotypes which link the UML diagrams to the process meta model. A class diagram is represented as shown in Figure 5. An analyzer checks process model definitions for consistency with the process meta model. The analyzer is coupled with a transformation tool which translates the UML model into an internal representation hidden from the process modeler [26].

Internally, AHEAD is based on a formal specification as a programmed graph rewriting system. To this end, we use the specification language *PROGRES* as well as its development environment, which offers a graphical editor, an analyzer, an interpreter and a code generator [28]. Both the process meta model and process model definitions are specified in PROGRES. The former was created once by the tool builders of AHEAD; the latter ones are generated automatically by the transformation tool. That is, the process modeler shown in Figure 10 is not aware of the PROGRES specification which is employed internally.

The overall specification, consisting of both the process meta model and the process model definition, is translated by the PROGRES compiler into C code. The generated

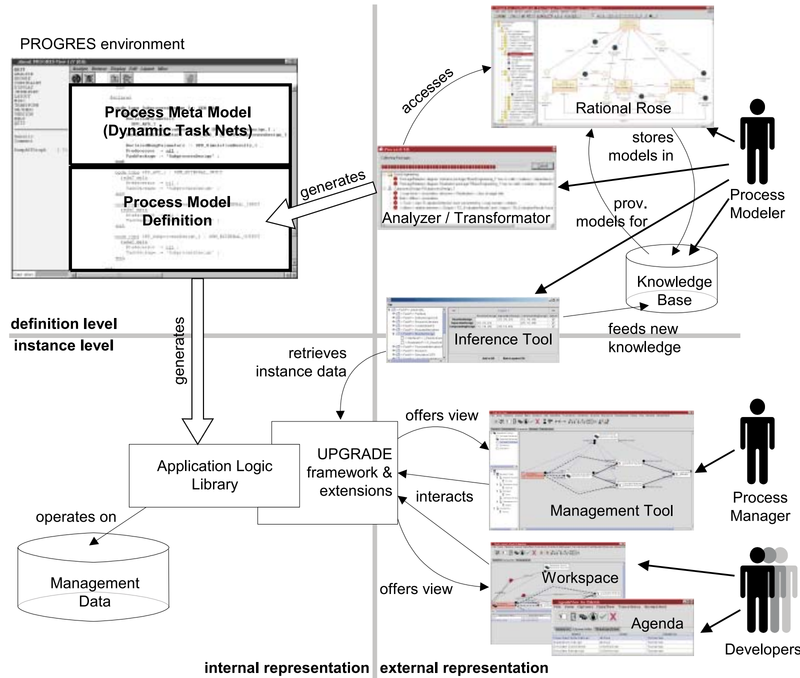

Fig. 10. Architecture of the AHEAD system

code constitutes the application logic of the instance-level tools. The application logic library operates on the management data which are stored in a graph-based DBMS. The user interface of the management tools is implemented with *UPGRADE*, a framework for building graph-based interactive tools [2].

The *management tool* assists a process manager in planning, analyzing, monitoring, and controlling a development process. A screenshot from the management tool is shown in Figure 11; it corresponds to the task net of Figure 3. The management tool is coupled with tools provided to developers 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 of the inference tool, the reader is referred to [27].

To conclude this section, let us summarize how process evolution is supported by AHEAD. The sample process presented in the previous section assumes that a type-level process definition has already been created. For a while, the design process proceeds according to the definition. Planning and execution are interleaved seamlessly, the task net is extended gradually (instance evolution). Then, the manager detects the need for a deviation. Consistency enforcement is switched off in the task net for the separation design, and the estimation task is inserted. These steps are performed with the help of the management tool. Execution continues even in the presence of inconsistencies until it is decided to improve the process definition. To this end, the process modeler creates new package versions in Rational Rose. This results in an extension of the process definition, i.e., the old parts are still present. The extended definition is transformed into the PROGRES specification, which in turn is compiled into C code. Now the process manager may migrate the task net to the improved definition.

V. Related Work

Previous papers on the AHEAD system have presented instance-level evolution [9], [24] and UML modeling of development processes [26], [14]. This paper is based on the Ph.D. thesis of the second author [27], which introduces the wide spectrum approach, round-trip process evolution, and toleration of inconsistencies. We have not described these contributions in other publications. Altogether, the AHEAD system now provides comprehensive mechanisms for managing evolving development processes. These mechanisms are combined synergetically in an integrated framework which provides an added value going beyond the sum of its individual parts.

Workflow management systems, as presented by the work of the Workflow Management Coalition [19], have

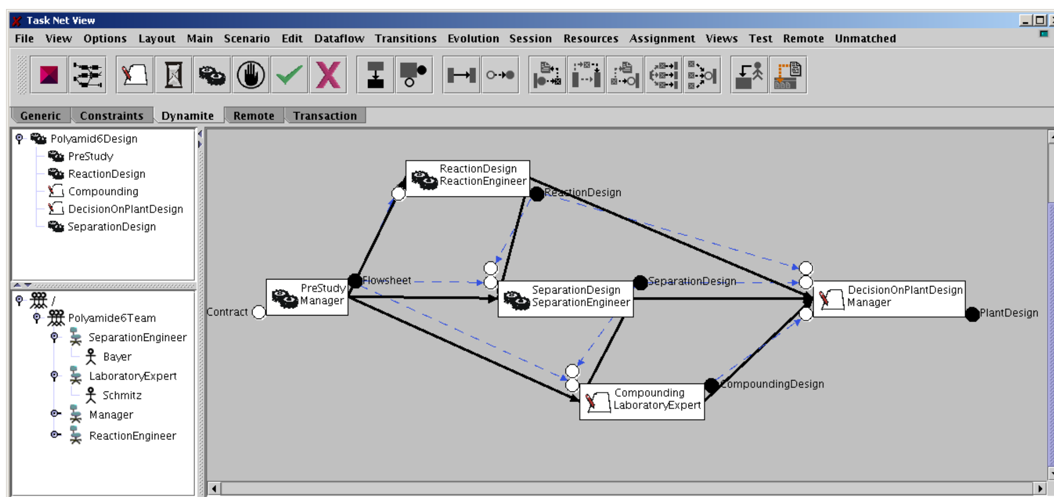

Fig. 11. Screenshot from the management tool

been developed to support repetitive processes occurring e.g. in banks, insurance companies, administrations, etc. A workflow is defined once, and this definition is used for executing a high number of processes. The WfMC does not deal with evolution, neither on the definition nor on the instance level.

The need for a wide spectrum approach to process management was recognized as a research challenge in [29]. It is explicitly addressed in GroupProcess [11], a project that has been launched recently, but does not seem to have produced technical results yet. In addition, this matter is addressed in some workflow management systems which originally focused on highly structured processes. For example, in Mobile [10] and FLOW.NET [16] the process modeler may define the control flow as restrictively as desired and may even introduce new control flow types. In addition, many commercial systems allow for deviations such as skipping, redoing or delegation of activities. Finally, exception handling [8] may be used to deal with errors and special cases. However, the main focus still lies on highly or medium-structured processes. In contrast, our approach covers the whole spectrum, including also ad hoc processes.

There are only a few other approaches to process management which are capable of dealing with inconsistencies. [7] and [21] both deal with inconsistencies between process definitions and process instances. In PROSYT [7], users may deviate from the process definition by enforcing operations violating preconditions and state invariants. However, all of these approaches do not deal with definition-level evolution, i.e., it is not addressed how inconsistencies can be resolved by migrating to an improved definition.

A key and unique feature of our approach consists in its support for round-trip process evolution. To realize this approach, we have to work both bottom-up and top-down: we learn from actual performance (bottom-up) and propagate changes to process definitions top-down. In contrast, most other approaches are confined to top-down evolution. For example, in [10], [12], [15], [17], the process definition has to be created beforehand, while we allow for executing partially known process definitions.

Modifications to process definitions may be performed in place, as in [30], [5]. However, it seems more appropriate to create a new version of the definition in order to provide for traceability. Version control is applied at different levels of granularity such as class versioning [15], [17] and schema versioning [4]. Our approach is similar to class versioning (interface and realization packages for individual task types are submitted to version control).

Different migration strategies may be applied in order to propagate changes at the definition level to the instance level. A fairly comprehensive discussion of such strategies is given in [4]. We believe that the underlying base mechanisms must be as flexible as possible. For example, in [10], [15], [17], both structural and behavioral consistency must be maintained during migration. This is not required in our approach, which even tolerates persistent inconsistencies.

Finally, there are a few approaches which are confined to instance-level evolution (e.g., [1], [25]). A specific process instance is modified, taking the current execution state into account. However, there is no way to constrain the evolution (apart from constraints which are built into the underlying process meta model). In contrast, in AHEAD instances are evolved under the control of the process definition. Inconsistencies can be permitted selectively, if required.

VI. Conclusion

We have presented a management system for evolving development processes which provides an integrated set of tools for round-trip process evolution. We have applied our approach to development processes in different engineer-

ing disciplines, namely software, mechanical, and chemical engineering. In this paper, we have focused on chemical engineering, which is studied in the IMPROVE project. Our recent work on process evolution has been driven by the study of design processes in chemical engineering to a large extent. We have applied the AHEAD system successfully to the reference scenario studied in the IMPROVE project, which was elaborated in cooperation with industrial partners. Future work will address further evaluation of the mechanisms for evolution support. While these mechanisms seem to be powerful and general enough, we still have to reduce the complexity of the user interface and have to provide for methods of use.

Acknowledgments

This work was partially supported by the Deutsche Forschungsgemeinschaft (DFG), the funder of the Collaborative Research Council IMPROVE. Support is gratefully acknowledged.

References

[1] S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, Dec. 1993.

[2] B. Böhlen, D. Jäger, A. Schleicher, and B. Westfechtel. UPGRADE: Building interactive tools for visual languages. In N. Callaos, L. Hernandez-Encinas, and F. Yetim, editors, *Proceedings of the 6th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2002)*, volume I (Information Systems Development I), pages 17–22, Orlando, Florida, 2002.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, 1998.

[4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In B. Thalheim, editor, *Proceedings of the 15th International Conference on Conceptual Modeling (ER)*, LNCS 1157, pages 438–455, Cottbus, Germany, Oct. 1996. Springer-Verlag.

[5] S.-C. Chou and J.-Y. Chen. Process evolution support in a concurrent software process language environment. *Information and Software Technology*, 41:507–524, 1999.

[6] R. Conradi, C. Fernström, A. Fugetta, and R. Snowdown. Towards a reference framework for process concepts. In J.-C. Derniame, editor, *Proceedings 2nd European Workshop on Software Process Technology (EWSPT '92)*, LNCS 772, pages 3–17, Trondheim, Norway, 1992. Springer-Verlag.

[7] G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Transactions on Software Engineering*, 24(11):982–1001, Nov. 1998.

[8] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958, Oct. 2000.

[9] P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. DYNAMITE: Dynamic task nets for software process management. In *Proceedings of the 18th International Conference on Software Engineering (ICSE-18)*, pages 331–341, Berlin, Germany, Mar. 1996. IEEE Computer Society Press.

[10] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A comprehensive approach to flexibility in workflow management systems. In D. Georgakopoulos, Wolfgang, and A. L. Wolf, editors, *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC '99)*, volume 24-2 of *ACM SIGSOFT Software Engineering Notes*, pages 79–88, San Francisco, CA, Mar. 1999. ACM Press.

[11] C. Huth, I. Erdmann, and L. Nastansky. GroupProcess: Using process knowledge from the participative design and practical operation of ad hoc processes for the design of structured workflows. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Hawaii, Jan. 2001. IEEE Computer Society Press. 10 p.

[12] M. L. Jaccheri and R. Conradi. Techniques for process model evolution in EPOS. *IEEE Transactions on Software Engineering*, 19(12):1145–1156, Dec. 1993.

[13] D. Jäger, A. Schleicher, and B. Westfechtel. AHEAD: A graph-based system for modeling and managing development processes. In Nagl et al. [23], pages 325–339.

[14] D. Jäger, A. Schleicher, and B. Westfechtel. Using UML for software process modeling. In O. Nierstrasz and M. Lemoine, editors, *Software Engineering — ESEC/FSE '99*, LNCS 1687, pages 91–108, Toulouse, France, Sept. 1999. Springer-Verlag.

[15] G. Joeris and O. Herzog. Managing evolving workflow specifications. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS'98)*, pages 310–321. IEEE Computer Society Press, 1998.

[16] G. Joeris and O. Herzog. Towards flexible and high-level modeling and enacting of processes. In M. Jarke and A. Oberweis, editors, *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE'99)*, LNCS 1626, pages 88–102, Heidelberg, Germany, June 1999. Springer-Verlag.

[17] M. Kradolfer and A. Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proceedings of the International Conference on Cooperative Information Systems (CoopIS'99)*, pages 104–114, Edinburgh, Sept. 1999. IEEE Computer Society Press.

[18] C.-A. Krapp, S. Krüppel, A. Schleicher, and B. Westfechtel. Graph-based models for managing development processes, resources, and products. In G. Engels and G. Rozenberg, editors, *TAGT '98 — 6th International Workshop on Theory and Application of Graph Transformation*, LNCS 1764, pages 455–474, Paderborn, Germany, Nov. 1998. Springer-Verlag.

[19] P. Lawrence, editor. *Workflow Handbook*. John Wiley, Chichester, UK, 1997.

[20] J. Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proceedings of the 2nd International Conference on the Software Process*, pages 41–53, Berlin, Germany, Feb. 1993. IEEE Computer Society Press.

[21] T. Murata and A. Borgida. Handling of irregularities in human centered systems: A unified framework for data and processes. *IEEE Transactions on Software Engineering*, 26(10):959–977, Oct. 2000.

[22] M. Nagl and W. Marquardt. SFB-476 IMPROVE: Informatische Unterstützung übergreifender Entwicklungsprozesse in der Verfahrenstechnik. In M. Jarke, K. Pasedach, and K. Pohl, editors, *Proceedings Informatik '97*, Informatik aktuell, pages 143–154, Aachen, Sept. 1997. Springer-Verlag.

[23] M. Nagl, A. Schürr, and M. Münch, editors. *Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE).* LNCS 1779. Springer-Verlag, Castle Rolduc, The Netherlands, Sept. 1999.

[24] M. Nagl, B. Westfechtel, and R. Schneider. Tool support for the management of design processes in chemical engineering. *Computers & Chemical Engineering*, 27(2):175–197, Feb. 2003.

[25] M. Reichert and P. Dadam. ADEPT$_{flex}$ — supporting dynamic changes without loosing control. *Journal of Intelligent Information Systems*, 10(2):93–129, Mar. 1998.

[26] A. Schleicher. Formalizing UML-based process models using graph transformations. In Nagl et al. [23], pages 341–358.

[27] A. Schleicher. *Management of Development Processes — An Evolutionary Approach*. Deutscher Universitaets-Verlag, Wiesbaden, Germany, 2002.

[28] A. Schürr, A. Winter, and A. Zündorf. Graph grammar engineering with PROGRES. In W. Schäfer and P. Botella, editors, *Proceedings of the European Software Engineering Conference (ESEC '95)*, LNCS 989, pages 219–234, Barcelona, Spain, Sept. 1995. Springer-Verlag.

[29] A. Sheth et al. NSF workshop on workflow and process automation. *ACM Software Engineering Notes*, 22(1):28–38, Jan. 1997.

[30] M. Weske. Formal foundations and conceptual design of dynamic adaptations in a workflow management system. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, Hawaii, Jan. 2001. IEEE Computer Society Press.