3.4 An Adaptive and Reactive Management System for Project Coordination

M. Heller, D. Jäger, C.-A. Krapp, M. Nagl, A. Schleicher, B. Westfechtel, and R. Wörzberger

Abstract. Design processes in chemical engineering are hard to support. In particular, this applies to conceptual design and basic engineering, in which the fundamental decisions concerning the plant design are performed. The design process is highly creative, many design alternatives are explored, and both unexpected and planned feedback occurs frequently. As a consequence, it is inherently difficult to manage design processes, i.e. to coordinate the effort of experts working on tasks such as creation of flowsheets, steady-state and dynamic simulations, etc. On the other hand, proper management is crucial because of the large economic impact of the performed design decisions.

We present a management system which takes the difficulties mentioned above into account by supporting the coordination of dynamic design processes. The management system equally covers products, activities, and resources, and their mutual relationships. In addition to local processes, interorganizational design processes are addressed by delegation of subprocesses to subcontractors. The management system may be adapted to an application domain by a process model which defines types of tasks, documents, etc. Furthermore, process evolution is supported with respect to both process model definitions and process model instances; changes may be propagated from definitions to instances and vice versa (round-trip process evolution).

3.4.1 Introduction and Overview

As *design processes* are highly creative, they can rarely be planned completely in advance. Rather, planning and execution may have to be interleaved seamlessly. In the course of the design process, many design alternatives are explored which are mutually dependent. Furthermore, design proceeds iteratively, starting from sketchy, coarse-level designs to detailed designs which are eventually needed for building the respective chemical plant. Iterations may cause feedback to earlier steps of the design process. It may also be necessary to revoke inadequate design decisions. Finally, design involves cooperation among team members from different disciplines and potentially multiple enterprises, causing additional difficulties concerning the coordination of the overall design process.

Technical tools such as flowsheet editors, simulators for steady-state and dynamic simulations, etc. are crucial aids for effectively and efficiently performing design tasks [354]. In addition, *managerial tools* are required which address the coordination of design processes. In fact, such tools are crucial for supporting *business decision making* [174]. In the course of the design process, many decisions have to be made concerning the steps of the chemical process, the relationships among these steps, the realization of chemical process steps by devices, etc. To perform these decisions, design alternatives have

M. Nagl and W. Marquardt (Eds.): Collaborative and Distributed Chemical Engineering, LNCS 4970, pp. 300–366, 2008.

to be identified and elaborated, and the respective design tasks have to be coordinated regarding their mutual interfaces and dependencies. To support business decision making, managerial tools must provide chief designers with accurate views of the design process at an adequate level of granularity, offer tools for planning, controlling, and coordinating design tasks, thereby taking care of the dynamics of design processes.

The management system *AHEAD* (*A*daptable and *H*uman-Centered *E*nvironment for the M*A*nagement of *D*esign Processes [120, 161, 162, 207, 209, 212, 249, 355, 392, 474–476, 478, 488]) addresses the challenge of supporting dynamic engineering design processes. It has been developed in the context of the long-term research project IMPROVE [299, 343, 352] described in this volume which is concerned with models and tools for design processes in chemical engineering. The management tool AHEAD is primarily developed to support design teams in the industrial practice. In order to develop concepts and tools which can be transferred into practice, we have chosen to use a case study in the IMPROVE project as a reference scenario and a guideline for our tool design process ([17], and Sects. 1.1, 1.2). The case study refers to the conceptual design and basic engineering of a plant for the production of *Polyamide-6* (PA6). This approach has been greatly supported by the fruitful collaboration with our engineering partners in the IMPROVE project.

AHEAD equally covers products, activities, and resources and, therefore, offers more comprehensive support than project or workflow management systems. Moreover, AHEAD supports seamless interleaving of planning and execution – a crucial requirement which workflow management systems usually do not meet. Design processes are represented by *dynamic task nets*, which may evolve continuously throughout the execution of a design process [159, 160, 163, 242, 243, 472]. Dynamic task nets include modeling elements specifically introduced for design processes, e.g., feedback relationships for iterations in the design process which cannot be represented in project plans. This way, AHEAD improves business decision making since it offers a more natural, realistic, and adequate representation of design processes.

Initially, the AHEAD system focused on the management of design processes within one organization. In particular, we assumed that all management data are stored in a central database which can be accessed by all users. This assumption breaks down in case of *interorganizational design processes*. Each of the participating organizations requires a view on the overall design process which is tailored to its needs. In particular, it is crucial to account for information hiding such that sensitive data are not propagated outside the organization.

To support the management of interorganizational design processes, we have developed an approach which is based on *delegation* [30, 208]. A subprocess may be delegated to a subcontractor, passing only those data which are relevant for the contract. Both the contractor and the subcontractor use their own instances of the management system, which maintain their data in local

databases. The management systems are coupled at runtime by exchanging state information.

We have further developed this initial cooperation approach and extended it to a *view-based cooperation model* for the AHEAD system supporting interorganizational development processes. Organizations can create dynamic process views onto their local processes and publish them to other organizations. We utilize process views to enable organizations to manage how their local processes are integrated with other processes. A broader spectrum of cooperation scenarios besides delegation is supported. Additionally, contracts between organizations can be explicitly modeled and configured according to individual cooperation needs.

The AHEAD system may be applied to processes in different domains – including not only chemical engineering, but also other engineering disciplines such as software, electrical, or mechanical engineering. In fact, the core functionality is domain-independent and relies on general notions such as task, control flow, etc. AHEAD may be *adapted* to a certain *application domain* by defining domain-specific knowledge. For example, in chemical engineering domain-specific task types for flowsheet design, steady-state simulations, dynamic simulations, etc. may be introduced.

Domain-specific knowledge is formalized by a *process model definition* (cf. Sect. 2.4) which constrains the *process model instances* to be maintained at project runtime. As a consequence, the manager may compose task nets from predefined types and relationships. The process model definition is represented in the Unified Modeling Language (*UML* [560]), a wide-spread standard notation for object-oriented modeling. A process model is defined on the type level by a class diagram which has been adapted to the underlying *process meta model* for dynamic task nets [388, 389].

The current version of AHEAD provides for *evolution* both on the definition and the *instance level*. Changes on the definition level may be propagated to instances during their execution. If required, process model instances may deviate from their definitions under the control of the project manager who may switch off consistency enforcement deliberately and selectively (i.e., in designated subprocesses of the overall process). Knowledge acquired on the instance level may be propagated to the definition level, resulting in improved versions of process model definitions. This way, AHEAD provides for *round-trip process evolution*.

AHEAD is a research prototype which cannot be applied immediately in industry in a production environment for various reasons. In addition to deficiencies with respect to stability, efficiency, and documentation – problems which are faced by many research prototypes –, an important prerequisite of industrial use constitutes the integration with other management tools which are used in industry. Therefore, we *integrated* AHEAD with *several commercial systems* for workflow, document, and project management. The ultimate goal of these research activities is *technology transfer* into industrial practice.

This section describes 10 years of research on management of design processes. It should be clearly pointed out that this research within IMPROVE was carried out in *close cooperation* with subproject A1 (see Sections 2.4 and 2.5) and I1 (see Section 5.1), but also with B1 (see Section 3.1). The latter subproject also supports processes, but on another level and with different support mechanisms.

The rest of this *section* is *organized* as follows: Subsect. 3.4.2 introduces the AHEAD core system, which supports integrated management of products, activities, and resources for dynamic design processes. In the core system, process model definitions were static, and management was constrained to local processes within one organization. The next subsections describe extensions of the core system, namely on one hand the adaptation capabilities of AHEAD as well as round-trip process evolution (Subsect. 3.4.3) and on the other hand interorganizational coordination of design processes (Subsect. 3.4.4 and 3.4.5). Subsection 3.4.6 is concerned with related work. A conclusion is given in Subsect. 3.4.7.

3.4.2 AHEAD Core System

Basic Notions

In general terms, *management* can be defined as "all the activities and tasks undertaken by one or more persons for the purpose of planning and controlling the activities of others in order to achieve an objective or complete an activity that could not be achieved by the others acting alone" [996]. This definition stresses coordination as the essential function of management.

More specifically, we focus on the management of *design* processes by *coordinating* the *technical work* of designers. We do not target senior managers who work at a strategic level and are not concerned with the details of enterprise operation. Rather, we intend to support project managers who collaborate closely with the designers performing the technical work. Such managers, who are deeply involved in the operational business, need to have not only managerial but also technical skills ("chief designers").

The distinction between *persons* and *roles* is essential: When referring to a "manager" or a "designer", we are denoting a role, i.e., a collection of authorities and responsibilities. However, there need not be a 1:1 mapping between roles and persons playing roles. In particular, each person may play multiple roles. For example, in chemical engineering it is quite common that the same person acts both as a manager coordinating the project and as a (chief) designer who is concerned with technical engineering tasks.

In order to support managers in their coordination tasks, design processes have to be dealt with at an appropriate level of detail. We may roughly distinguish between *three levels* of *granularity*:

- At a *coarse-grained level*, design processes are divided into phases (or working areas) according to some life cycle model.

- At a *medium-grained level*, design processes are decomposed further down to the level of documents or tasks, i.e., units of work distribution.
- At a *fine-grained level*, the specific details of design subprocesses are considered. For example, a simulation expert may build up a simulation model from mathematical equations.

Given our understanding of management as explained above, the coarse-grained level does not suffice. Rather, decomposition has to be extended to the medium-grained level. On the other hand, usually management is not interested in the technical details of how documents are structured or how the corresponding personal subprocess is performed. Thus, the *managerial level*, which defines how management views design processes, comprises both *coarse-* and *medium-grained* representations.

In order to support managers in their coordination tasks, they must be supplied with appropriate *views* (abstractions) of *design processes*. Such views must be comprehensive inasmuch as they include products, activities, and resources (and their mutual relationships, see Sect. 1.1):

- The term *product* denotes the results of design subprocesses (e.g., flowsheets, simulation models, simulation results, cost estimates, etc.). These may be organized into *documents*, i.e., logical units which are also used for work distribution or version control. Complete results are *subconfigurations*.
- The term *activity* denotes an action performing a certain function in a design process. At the managerial level, we are concerned with *tasks*, i.e., descriptions of activities assigned to *designers* by managers, but also complex tasks performed by subproject groups.
- Finally, the term *resource* denotes any asset needed by an activity to be performed. This comprises both human and computer resources (i.e., the designers and managers participating in the design process as well as the computers and the tools they are using). Please note that also resources might be atomic or composed.

Thus, an overall *management configuration* consists of multiple parts representing products, activities, and resources. An example is given in Fig. 3.61. Here, we refer to the Polyamide-6 design process introduced earlier. On the left, the figure displays the roles in the design team as well as the designers filling these roles. The top region on the right shows design activities connected by control and data flows. Finally, the (versioned) products of these activities are located in the bottom-right region.

Below, we give a more detailed description of Fig. 3.61:

- *Products.* The results of design processes such as process flowsheets, steady-state and dynamic simulations, etc. are represented by documents (ellipses). Documents are interdependent, e.g., a simulation model depends on the process flowsheet (PFD) to which it refers (arrows between ellipses).

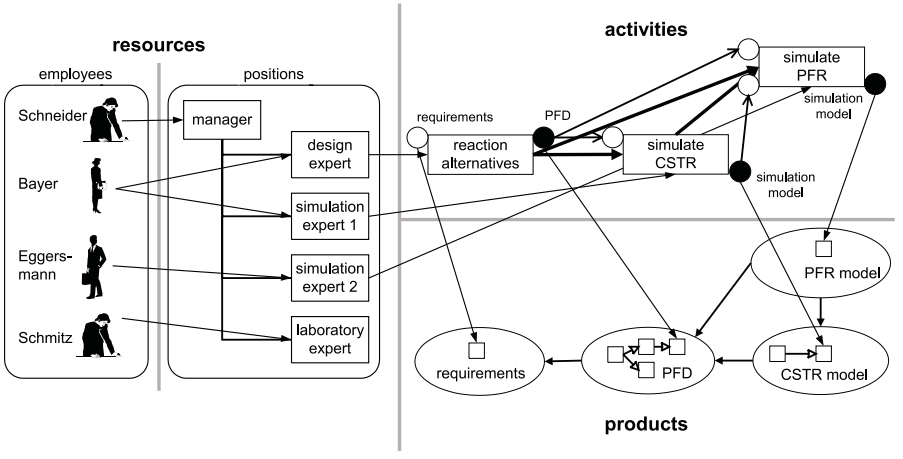

Fig. 3.61. Management configuration

The evolution of documents is captured by version control (each box within an ellipsis represents a version of some document).

- *Activities.* The overall design process is decomposed into tasks (rectangular boxes) which have inputs and outputs (white and black circles, respectively). The order of tasks is defined by control flows (thick arrows); e.g., reaction alternatives must have been inserted into the flowsheet before they can be simulated. Finally, data flows (arrows connecting circles) are used to transmit document versions from one task to the next.
- *Resources.* Employees (icons on the left) such as Schneider, Bayer, etc. are organized into project teams which are represented by organization charts. Each box represents a position, lines reflect the organizational hierarchy. Employees are assigned to positions (or roles). Within a project, an employee may play multiple roles. E.g., Mrs. Bayer acts both as a designer and as a simulation expert in the Polyamide-6 team.
- *Integration.* There are several relationships between products, activities, and resources. In particular, tasks are assigned to positions (and thus indirectly to employees). Furthermore, document versions are created as outputs and used as inputs of tasks.

It is crucial to *understand* the scope of the term "*management*" as it is used in this section. As already stated briefly above, management requires a certain amount of abstraction. This means that the details of the *technical level* are *not represented* at the managerial level. This is illustrated in Fig. 3.62, whose upper part shows a small cutout of the management configuration of Fig. 3.61. At the managerial level, the design process is decomposed into activities such as the creation of reaction alternatives and the simulation of these alternatives. Activities generate results which are stored in document versions. At the managerial level, these versions are basically considered black boxes, i.e.,

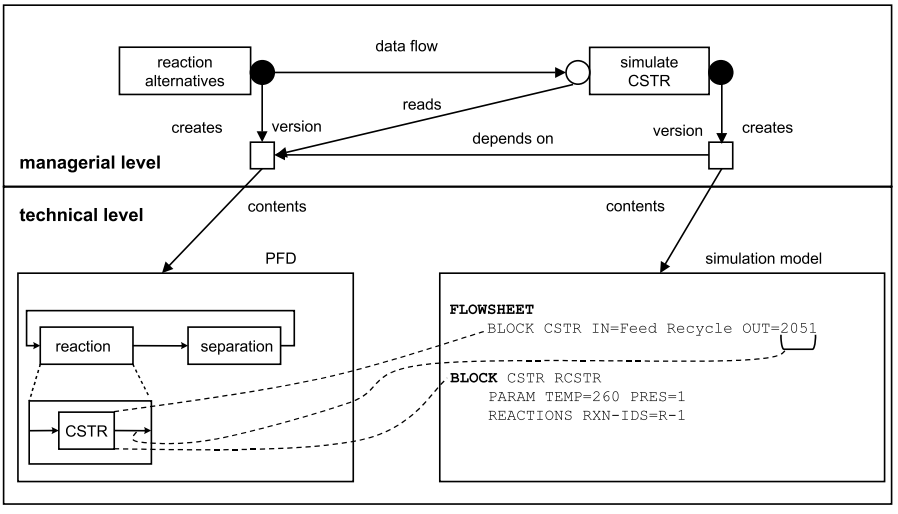

Fig. 3.62. Managerial and technical level

they are represented by a set of descriptive attributes (author, creation date, etc.) and by references to the actual contents, e.g., PFDs and simulation models. How a PFD or a simulation model is structured internally (and how their contents are related to each other), goes beyond the scope of the managerial level. Likewise, the managerial level is not concerned with the detailed personal process which is executed by some human to create a PFD, a simulation model, etc.

This does not imply that technical details are ignored. Rather, it must be ensured that the managerial level actually constitutes a correct *abstraction* of the *fine-grained information at the technical level* – and also controls technical activities. In fact, the management system described in this paper is part of an integrated environment for supporting design processes in chemical engineering. As such, it is integrated with tools providing fine-grained product and process support [21, 26]. The interplay of the tools of the overall environment is sketched only briefly in this section; see [348].

Particular attention has to be paid to the *dynamics* of design processes: The design process is not known in advance. Rather, it continuously evolves during execution. As a consequence, all *parts* of a *management configuration evolve* continuously:

- *Products.* The product structure is determined only during the design process. It depends on the flowsheets which is continuously extended and modified. Other documents such as simulation models and simulation results depend on the flowsheet. Moreover, different variants of the chemical process are elaborated, and selections among them are performed according to feedback gained by simulation and experiments.

- *Activities.* The activities to be performed depend on the product structure, feedback may require the re-execution of terminated activities, concurrent/simultaneous engineering calls for sophisticated coordination of related activities, etc.
- *Resources.* Resource evolution occurs likewise: New tools arrive, old tool versions are replaced with new ones, the project team may shrink due to budget constraints, or it may be extended to meet a crucial deadline, etc.

However, a management configuration should not evolve in arbitrary ways. There are *domain-specific constraints* which have to be met. In particular, activities can be classified into types such as requirements definition, design, simulation, etc. (likewise for products and resources). Furthermore, the way how activities are connected is constrained as well. For example, a flowsheet can be designed only after the requirements have been defined. Such domain-specific constraints should be taken into account such that they restrict the freedom of evolution.

Overview of the AHEAD System

Since current management systems suffer from several limitations (see introduction and section on related work), we have designed and implemented a new management system which addresses these limitations. This system is called AHEAD [212, 355]. AHEAD is *characterized* by the following *features*:

- *Medium-grained representation.* In contrast to project management systems, design processes are represented at a medium-grained level, allowing managers to effectively control the activities of designers. Management is not performed at the level of milestones, rather, it is concerned with individual tasks such as "simulate the CSTR reactor".
- *Coverage and integration at the managerial level.* AHEAD is based on an integrated management model which equally covers products, activities, and resources. In contrast, project and workflow management systems primarily focus on activities and resources, while product management systems are mainly concerned with the products of design processes.
- *Integration between managerial and technical level.* In contrast to project management systems, the AHEAD system also includes support tools for designers that supply them with the documents to work on, and the tools that they may use.
- *Support for the dynamics of design processes.* While many workflow management systems are too inflexible to allow for dynamic changes of workflows during execution, AHEAD supports evolving design processes, allowing for seamless integration of planning, execution, analysis, and monitoring.
- *Adaptability.* Both the structure of management configurations and the operations to manipulate them can be adapted by means of a domain-specific object-oriented model based on the UML [560].

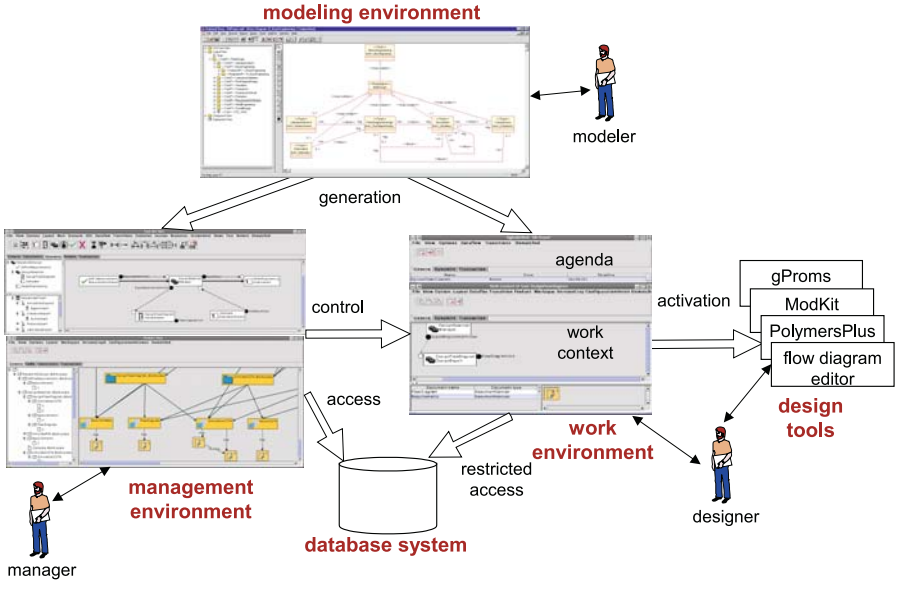

Fig. 3.63. Major components of the AHEAD system

Figure 3.63 gives an *overview* of the AHEAD *system*. AHEAD offers environments for different kinds of users, which are called modeler, manager, and designer. In the following, we will focus on the functionality that the AHEAD system provides to its users. Its technical realization will be discussed later.

The *management environment* supports project managers in planning, analyzing, monitoring, and controlling design processes. It provides graphical tools for operating on management configurations. These tools address the management of activities, products, and resources, respectively [244]:

- For *activity management*, AHEAD offers dynamic task nets which allow for seamless interleaving of planning, analyzing, monitoring, and controlling. A task net consists of tasks that are connected by control flow and data flow relationships. Furthermore, feedback in the design process is represented by feedback relationships. Tasks may be decomposed into subtasks, resulting in task hierarchies. The manager constructs task nets with the help of a graphical editor. He may modify task nets at any time while a design process is being executed.
- *Product management* is concerned with documents such as flowsheet, simulation models, cost estimations, etc. AHEAD offers version control for these documents with the help of version graphs. Relationships (e.g., dependencies) between documents are maintained as well. Versions of documents may be composed into configurations, thereby defining which versions are consistent with each other. The manager may view the version histories

and configurations with the help of a graphical tool. This way, he may keep track of the work results produced by the designers.
- *Resource management* deals with the organizational structure of the enterprise as far as it is relevant to design processes. AHEAD distinguishes between abstract resources (positions or roles) and concrete resources (employees). The manager may define a project team and then assign employees to the project positions.

Management of activities, products, and resources is *fully integrated*: Tasks are assigned to positions, inputs and outputs of tasks refer to document versions. Moreover, AHEAD manages task-specific workspaces of documents and supports invocation of design tools (see below).

AHEAD does not only support managers. In addition, it offers a *work environment* for *designers* which consists of two major components:

- The *agenda tool* displays the tasks assigned to a designer in a table containing information about state, deadline, expected duration, etc. The designer may perform operations such as starting, suspending, finishing, or aborting a task.
- The *work context tool* manages the documents and tools required for executing a certain task. The designer is supplied with a workspace of versioned documents. He may work on a document by starting a tool such as e.g. a flowsheet editor, a simulation tool, etc.

Please note that the scope of *support* provided by the *work environment* is limited. We do not intend to support design activities in detail at a technical level. Rather, the work environment is used to couple technical activities with management. There are other tools which support design activities at a fine-grained level. For example, a process-integrated flowsheet editor [21] may be activated from the work environment. "Process-integrated" means that the designer is supported by process fragments which correspond to frequently occurring command sequences, see Sect. 3.1. These process fragments encode the design knowledge which is available at the technical level. This goes beyond the scope of the AHEAD system, but it is covered by the overall environment for supporting design processes to which AHEAD belongs as a central component.

Both the management environment and the work environment access a common *management database*. However, they access it in different ways, i.e., they invoke different kinds of functions. The work environment is restricted to those functions which may be invoked by a designer. The management environment provides more comprehensive access to the database. For example, the manager may modify the structure of a task net, which is not allowed for a designer.

Before the AHEAD system may be used to carry out a certain design process, it must be adapted to the respective application domain [211]. AHEAD

consists of a generic kernel which is domain-independent. Due to the generality of the underlying concepts, AHEAD may be applied in different domains such as software, mechanical, or chemical engineering. On the other hand, each domain has its specific constraints on design processes. The *modeling environment* is used to provide AHEAD with domain-specific knowledge, e.g., by defining task types for flow diagram design, steady-state and dynamic simulation, etc. From a domain-specific process model, code is generated for adapting the management and the work environment.

A Tour through the AHEAD System

In the following, we introduce the tool *support* provided by the AHEAD system with the help of a small *demo session*. The demo refers to the overall reference process of IMPROVE, namely the design of a chemical plant for producing Polyamide-6 (see Sect. 1.2). Here, we focus on the design of the reaction which proceeds as follows: After an initial PFD has been created which contains multiple design variants, each of these variants is explored by means of simulations and (if required) laboratory experiments. In a final step, these alternatives are compared against each other, and the most appropriate one is selected. Other parts of the reference process will be addressed in subsequent sections of this paper.

Modeling Design Processes

Before the AHEAD system is used to manage some actual design project, it is provided with a domain-specific *process model definition* (cf. Sect. 2.4) which should capture the conceptual design and basic engineering of arbitrary chemical design processes to meet our own demands in the IMPROVE project.

As we have discussed earlier, all parts of a management configuration evolve throughout the course of a design project. This kind of evolution is called *instance-level evolution*. While process model instances evolve continuously, we would like to constrain this evolution in such a way that a domain-specific design process is followed.

Object-oriented modeling is well suited to meet this requirement. The core of an object-oriented model is defined by a *class diagram* of the widely known Unified Modeling Language (UML). The UML class diagram declares types of tasks as classes and relationships between task types as associations which can be constrained by multiplicity restrictions. Object diagrams contain instances of classes (objects) and associations (links) which follow these restrictions. Object diagrams represent reusable task net patterns on the instance level. Both diagram types represent the structure of a process model. For behavioral aspects, state and collaboration diagrams are used.

Although the UML is a general object-oriented language, it has to be adapted to be suitable for process modeling. We use the *extension mechanism* of the UML to introduce new meta classes and meta attributes in the UML meta model [391].

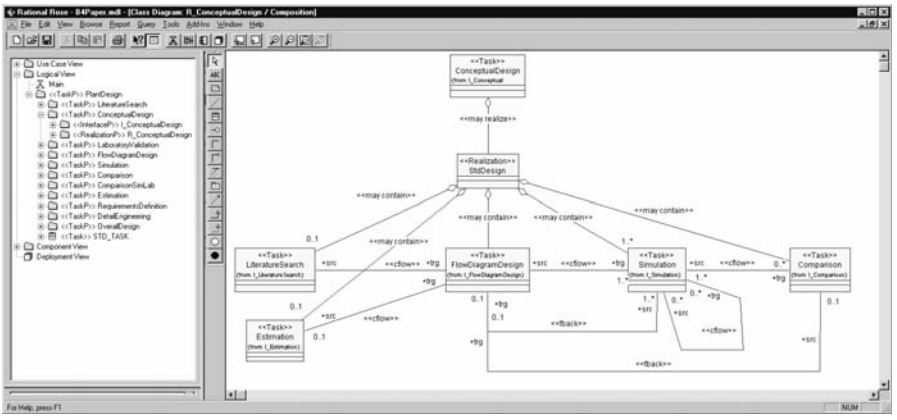

Fig. 3.64. UML model of a design process in chemical engineering (modeling environment)

In the following, we give a small *example* of *structural modeling* with the help of class diagrams; for more information, see [211, 390]. Furthermore, we will deal only with the modeling of activities, while a comprehensive process model must define products and resources, as well.

Figure 3.64 shows an *excerpt* of the *UML model* for design processes. The window on the left displays a hierarchy of packages, which are used to organize the overall model of the design process. On the right, a class diagram is presented which defines a part of the activities of the design process[32]. Here, *task classes* are modeled rather than specific instances. Instances are created dynamically at project runtime, implying that the topology of a task net is determined only at runtime. This object-oriented approach takes care of the dynamics of design processes and contrasts with the static workflows as defined in workflow management systems.

The *class diagram* introduces a complex task ConceptualDesign – dealing with the conceptual design of a certain part of the chemical process – and one of its possible realizations StdDesign. The realization contains multiple task classes, which is expressed by aggregations stereotyped with may contain. A contained task class may again be complex, resulting in a multi-level task hierarchy. For each contained task class, a cardinality is specified. E.g., 1..* means that at least one subtask is created at runtime[33]. Control flow associations (stereotype cflow) define the potential orders of task executions. E.g., a Comparison of design alternatives is performed at the very end of the design process.

[32] For the sake of simplicity, input and output parameters of tasks as well as data flows were removed from the diagram.

[33] The default cardinality is 1..1.

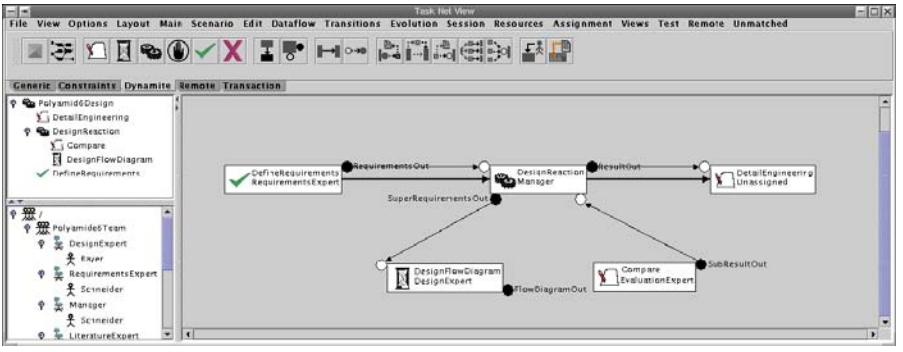

Fig. 3.65. Initial task net (management environment)

From the UML model, code is generated to *customize* the *functionality* provided by the AHEAD system. For example, the project manager may instantiate only the domain-specific classes and associations defined in the class diagrams. The core system as presented in this section enforces consistency with the process model definition. In this way, we can make sure that design proceeds according to the domain-specific model. A more flexible approach will be discussed in the next section (extending process evolution beyond consistency-preserving instance-level evolution).

Managing Design Processes

In this subsection, we illustrate the functionality of the AHEAD system provided at the instance level. This is performed with the help of a demo session which mainly focuses on the management environment, but also introduces the work environment.

Figure 3.65 presents a snapshot from the management environment taken in an early stage of the Polyamide-6 design process for reaction design. The upper region on the left displays a tree view of the task hierarchy. The lower left region offers a view onto the resources available for task assignments (see also Fig. 3.66). A part of the overall task net is shown in the graph view on the right-hand side. Each task is represented by a rectangle containing its name, the position to which the task has been assigned, and an icon representing its state (e.g., the gear-wheels represent the state Active, and the hour-glass stands for the state Waiting). Black and white circles represent outputs and inputs, respectively. These are connected by data flows (thin arrows). Furthermore, the ordering of task execution is constrained by control flows (thick arrows). Hierarchical task relations (decompositions) are represented by the graphical placement of the task boxes (from top to bottom) rather than by drawing arrows (which would clutter the diagram).

Please recall that the demo session deals only with the reaction part and particularly with its design (DesignReaction in Fig. 3.65). In this early stage, it is only known that initially some reaction alternatives have to be designed

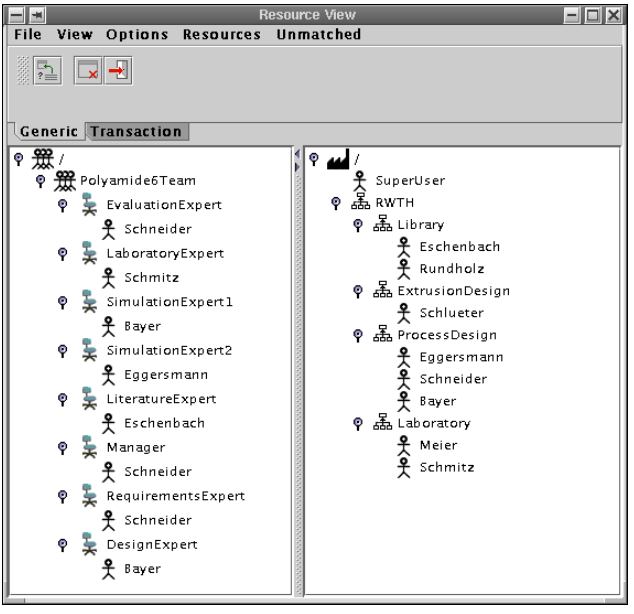

Fig. 3.66. Resource view (management environment)

(DesignFlowDiagram) and documented in a flowsheet. Furthermore, at the end these alternatives have to be compared (Compare), and a decision has to be performed. Other tasks of types defined in the class diagram of Fig. 3.64 are either left out (e.g. Estimation) or will be filled in later.

In addition to the initial task net, the manager has also used the resource management tool for building up his project team (Fig. 3.66). The region on the left displays the structure of the Polyamide-6 design team. Each position (represented by a chair icon) is assigned to a team member. Analogously, the region on the right shows the departments of the company. From these departments, the team members for a specific project are taken for a limited time span. Tasks are assigned to positions rather than to actual employees (see task boxes in Fig. 3.65). This way, assignment is decomposed into two steps. The manager may assign a task to a certain position even if this position has not been filled yet. Moreover, if a different employee is assigned to a position, the task assignments need not be changed: The tasks will be redirected to the new employee automatically.

The work environment is illustrated in Fig. 3.67. As a first step, the user logs into the system (not shown in the figure). Next, AHEAD displays an agenda of tasks assigned to the roles played by this user (top of Fig. 3.67). Since the user Bayer plays the role of the design expert, the agenda contains the task DesignFlowDiagram. After the user has selected a task from the agenda, the work context for this task is opened (bottom window). The work context graphically represents the task, its inputs and outputs, as well

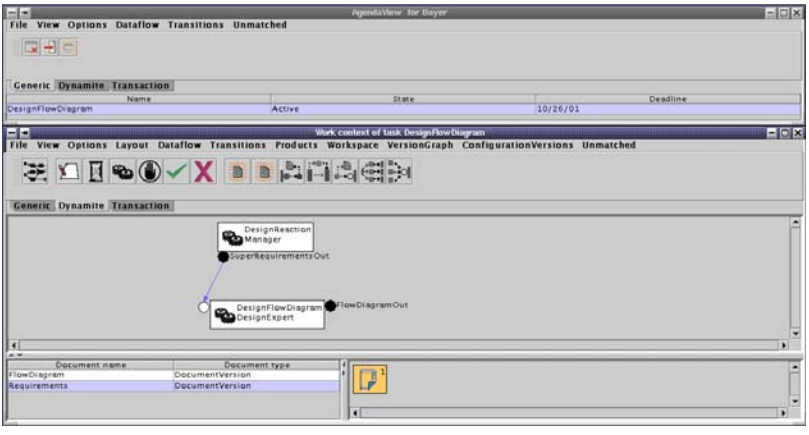

Fig. 3.67. Work environment

as its context in the task net (here, the context includes the parent task which defines the requirements to the flowsheet to be designed). Furthermore, it displays a list of all documents needed for executing this task. For some selected document, the version history is shown on the right (so far, there is only one version of the requirements definition which acts as input for the current task).

From the work context window, the user may activate design tools for operating on the documents contained in the workspace. Here, the user invokes a flowsheet editor [21] in order to insert reaction alternatives into the flowsheet for the Polyamide-6 process. The flowsheet editor, which was also developed in the IMPROVE project, is based on MS Visio, a commercial drawing tool, which was integrated with the PRIME process engine [371].

The resulting flowsheet is displayed in Fig. 3.68. The chemical process is decomposed into reaction, separation, and compounding. The reaction is refined into four variants. For our demo session, we assume that initially only two variants are investigated (namely a single CSTR and PFR on the left hand side of Fig. 3.68).

After the generation of the two variants, the manager extends the task net with tasks for investigating the alternatives that have been introduced so far (*product-dependent task net*, Fig. 3.69). Please note the control flow relation between the new tasks: The manager has decided that the CSTR should be investigated first so that experience from this alternative may be re-used when investigating the PFR. Furthermore, we would like to emphasize that the design task has not terminated yet. As to be demonstrated below, the designer waits for feedback from simulations in order to enrich the flowsheet with simulation data. Depending on these data, it may be necessary to investigate further alternatives.

Subsequently, the simulation expert creates a simulation model (using Polymers Plus) for the CSTR reactor and runs the corresponding simulations.

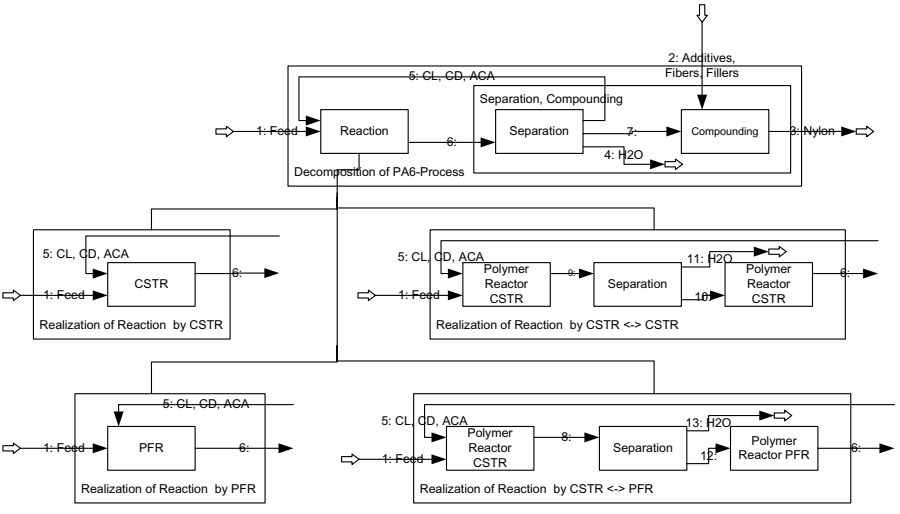

Fig. 3.68. Reaction alternatives in the process flowsheet

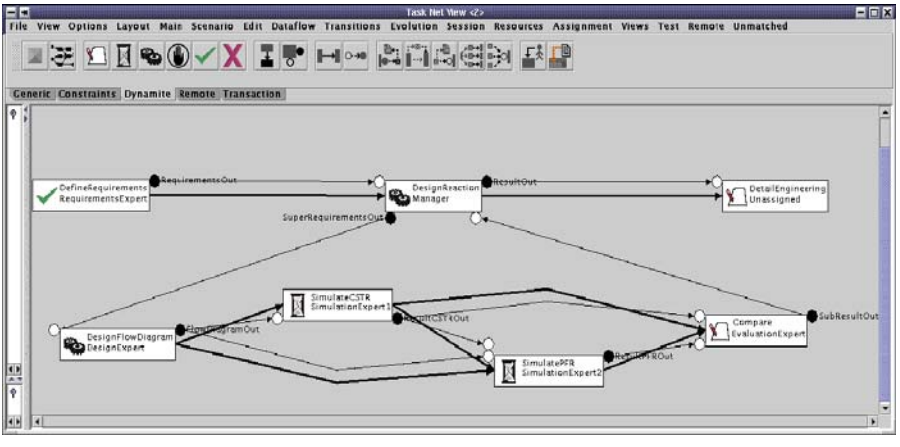

Fig. 3.69. Extended task net (management environment)

The simulation results are validated with the help of laboratory experiments. After these investigations have been completed, the flowsheet can be enriched with simulation data such as flow rates, pressures, temperatures, etc. To this end, a *feedback flow* – represented by a dashed arrow – is inserted into the task net (Fig. 3.70) [245, 246]. The feedback flow is refined by a data flow, along which the simulation data are propagated. Then, the simulation data are inserted into the flowsheet.

Please note that the semantics of the control flow from DesignFlowDiagram to SimulateCSTR is defined such that these tasks can be active simultaneously (*simultaneous engineering*) [576]. As a consequence, we cannot assume that

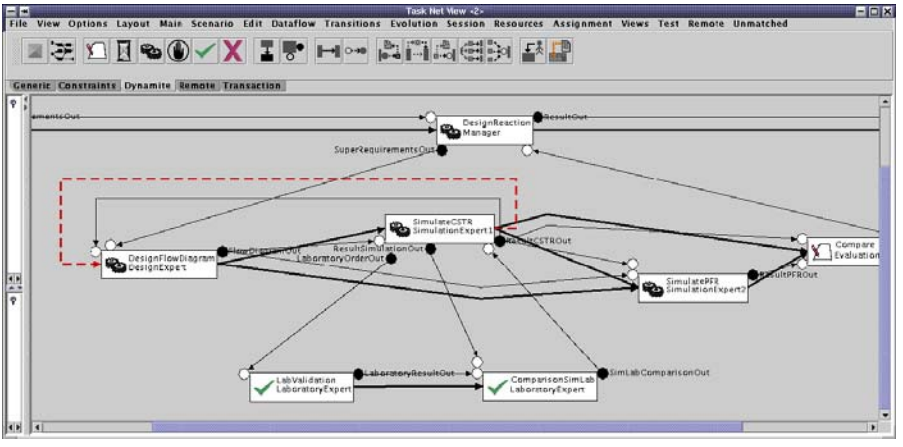

Fig. 3.70. Feedback and simultaneous engineering (management environment)

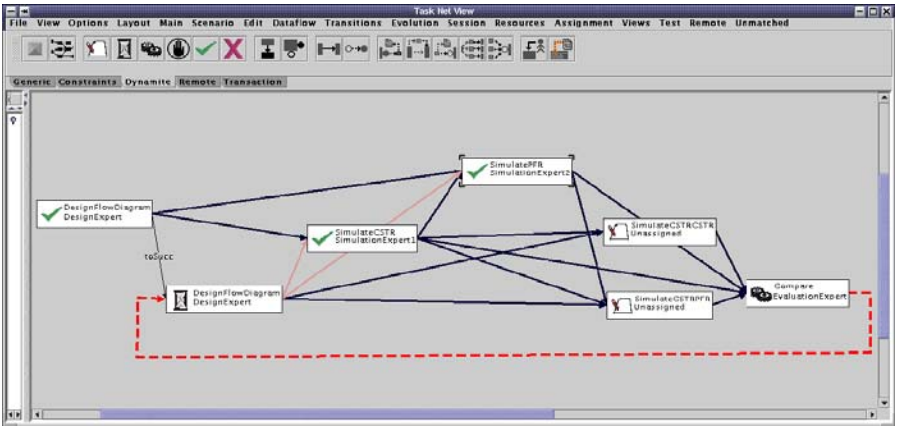

Fig. 3.71. Far-reaching feedback (management environment)

the work context of a task is stable with respect to its inputs. Rather, a predecessor task may deliver a new version that is relevant for its successors. This is taken care of by a sophisticated release policy built into the model underlying dynamic task nets [475].

After the alternatives CSTR and PFR have been elaborated, the evaluation expert compares all explored design alternatives. Since none of them performs satisfactorily, a far-reaching feedback is raised to the design task. Here, we assume that the designer has already terminated the design task. As a consequence, the design task has to be reactivated. Reactivation is handled by creating a new *task version*, which may or may not be assigned to the same designer as before. New design alternatives are created, namely a CSTR-CSTR and a CSTR-PFR cascade, respectively (see again Fig. 3.68).

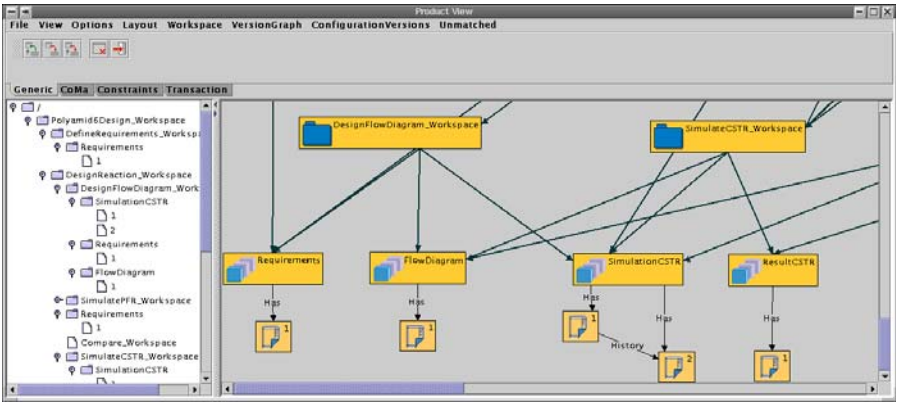

Fig. 3.72. Product view (management environment)

Furthermore, the task net is augmented with corresponding simulation tasks (Fig. 3.71[34]). After that, the new simulation tasks are assigned to simulation experts, and simulations are carried out accordingly. Eventually, the most suitable reactor alternative is selected.

So far, we have primarily considered the management of activities. Management of products, however, is covered as well. Figure 3.72 shows a tree view on the products of design processes on the left and a graph view on the right. Products are arranged into *workspaces* that are organized according to the task hierarchy. Workspaces contain sets of *versioned documents*.

Generally speaking, a *version* represents some state (or snapshot) of an evolving document. We distinguish between *revisions*, which denote temporal versions, and *variants*, which exist concurrently as alternative solutions to some design problem. Revisions are organized into sequences, variants result in branches. Versions are connected by *history relationships*. In Fig. 3.72, there is a history relationship between revisions 1 and 2 of SimulationCSTR, the simulation model for the CSTR reactor. In general, the version history of a document (flowsheet, simulation model, etc.) may evolve into an acyclic graph (not shown in the snapshot).

There is only one version of the flowsheet in Fig. 3.72. Here, we rely on the capabilities of the flowsheet editor to represent multiple variants. Still, the flowsheet could evolve into multiple versions at the managerial level (e.g., to record snapshots at different times). Moreover, in the case of a flowsheet editor with more limited capabilities (no variants), variants would be represented at the managerial level as parallel branches in the version graph.

Finally, it is worth noting that the support for process managers provided by the AHEAD system so far could easily be extended with respect to *process analysis* and *simulation* aspects. Within another sub project I2/I4 in

[34] Task parameters and data flows have been filtered out to avoid a cluttered diagram.

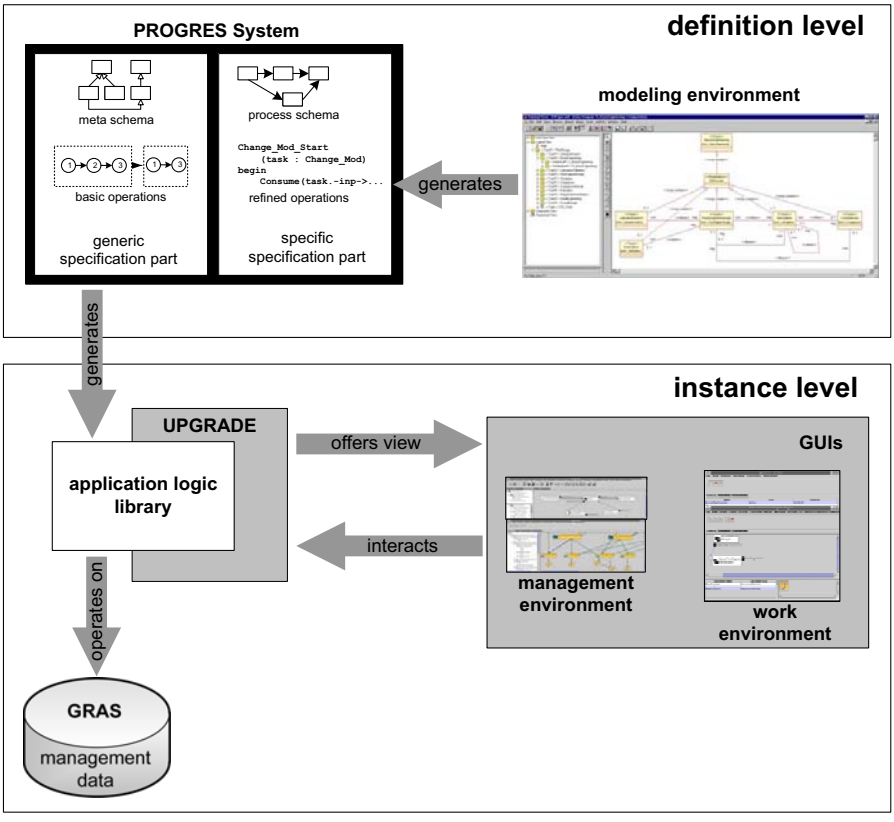

Fig. 3.73. Architecture of the AHEAD system

the CRC 476 IMPROVE, comprehensive support for detailed analysis of process activities, organizational structure, and information flow as well as the identification of weak spots within individual work processes has been developed. The chosen analysis and simulation approach and the research results are thoroughly described in Sect. 5.2 below. Although the focus in that work is on fine-grained work processes, a process manager could easily profit from similar analysis and simulation support on the medium-grained administrative management process layer above the work process layer. Further research in that direction is therefore needed.

Realization of the AHEAD Core System

Figure 3.73 displays the architecture of the AHEAD system. It also shows internal tools (left hand side) and thereby refines the overview given by Fig. 3.63.

Internally, AHEAD is based on a formal specification as a programmed graph rewriting system [206]. To this end, we use the specification language

PROGRES as well as its modeling environment, which offers a graphical editor, an analyzer, an interpreter and a code generator [414]. 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 modeling environment (cf. Subsect. 3.4.3).

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 code constitutes the application logic of the instance-level tools. The application logic library operates on the management data which are stored in the graph-based database management system *GRAS* [220]. The user interface of the management tools is implemented with *UPGRADE*, a framework for building graph-based interactive tools [49].

3.4.3 Process Evolution and Domain-Specific Parameterization

Motivation

As stated above, the AHEAD system may be applied to processes in different domains – including not only chemical engineering, but also other engineering disciplines such as software, electrical, or mechanical engineering. The core functionality is domain-independent and uses general notions such as task, control flow, etc. AHEAD may be *adapted* to a certain application domain by defining domain-specific knowledge [164]. For example, in chemical engineering domain-specific task types for flowsheet design, steady-state simulations, dynamic simulations, etc. may be introduced. Within a *process model definition*, domain-specific knowledge is defined which constrains the *process model instances* to be maintained at project runtime.

In the initial version of the AHEAD system, process model definitions were constrained to be static throughout the whole project lifecycle. Either no process model was defined at all, relying on a set of unconstrained *standard types*, or a process model had to be provided before the actual project execution could be started. Thus, evolution was constrained to the instance level (interleaving of planning and execution). However, gathering and fixing process knowledge beforehand turned out to be virtually infeasible for design processes, in particular in conceptual design and basic engineering of chemical plants. Therefore, support for *process evolution* was generalized considerably [171, 172].

While working on the Polyamide-6 reference process, it turned out that even process model definitions cannot be determined completely in advance. Therefore, we generalized evolution support to include all of the following features (cf. [171–173, 387, 390]):

- *Instance-level evolution.* Planning and enactment of dynamic task nets may be interleaved seamlessly (already part of the core system).

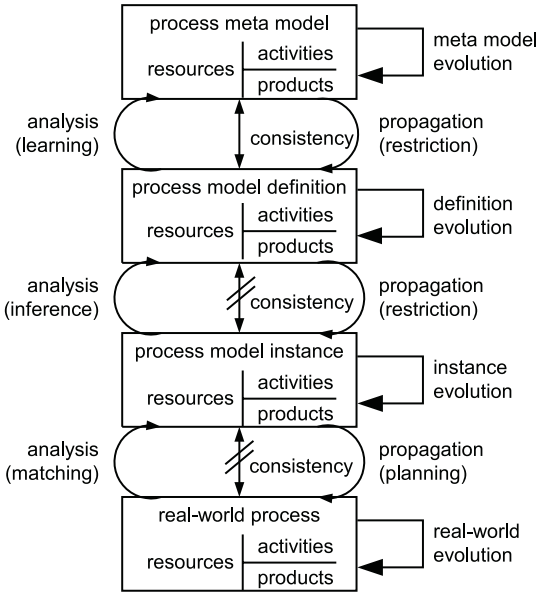

Fig. 3.74. Conceptual framework

- *Definition-level evolution.* At definition level, 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 model definition from a set of process model instances.
- *Top-down evolution.* A revised process model definition may be applied even to running process model instances by propagating the changes from the definition to the instance level.
- *Selective consistency control.* The project manager may allow for deviations of process model instances from their respective definitions resulting in inconsistencies. These deviations are reported to the project manager who may decide to reinforce consistency later on.

Conceptual Framework

Levels of Modeling

Our work is based on a conceptual framework which distinguishes four *levels of modeling* (Fig. 3.74). 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. 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) for process model definitions. The meta model is based on dynamic task nets. It provides meta elements for structural (tasks, control and data flows etc.) and for behavioral aspects (e.g. state machines for tasks) of these task nets.

Process (model) definitions are created as instances of process meta models and are defined in the UML using class diagrams at the type level and collaboration diagrams for recurring patterns at the abstract instance level. Process definitions are organized into interface packages defining the interface of a task (in terms of its inputs and outputs) and realization packages (of a complex task) containing 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).

Process (model) instances are instantiated from process model definitions. A process model definition represents reusable process knowledge at an abstract level whereas process model instances abstract from only 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 is used to guide and control process participants, who conversely provide feedback which is used to update the process model instance.

Wide Spectrum Approach

In general, a *wide spectrum* of processes has to be modeled, ranging from ad hoc to highly structured. Moreover, different subprocesses may exhibit different degrees of structuring. This requirement is met by defining for each subprocess a corresponding package which contains a model at an adequate level of structuring. As illustrated in Fig. 3.75, we may distinguish four levels of process knowledge to be discussed below.

On the *untyped level* (Fig. 3.75a), there is no constraining process knowledge (ad-hoc process). The process manager may create and connect any number of tasks with the help of *unconstrained types* (which are constrained only by the underlying meta model).

On the *partially typed level*, the process modeler is capable of defining domain-specific types of tasks and relationships, but he also permits the use of unconstrained types on the instance level. In particular, this allows to leave out exceptions like feedbacks in the definition. When feedback does occur during enactment (e.g., a design error is detected during implementation), it can be handled by instantiating an unconstrained type of feedback flow without necessarily changing the process model definition.

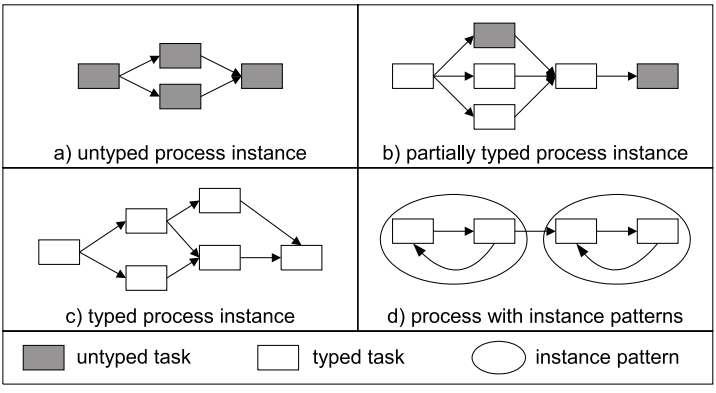

Fig. 3.75. Wide spectrum approach

The *completely typed level* requires complete typological knowledge of the respective subprocess and therefore permits only domain-specific types. Therefore, it excludes the use of unconstrained types and permits only domain-specific types. The only degree of freedom at the instance level is the cardinality of tasks which also can be constrained in the process model. For example, the cardinality [1:1] enforces exactly one instance of the respective task type.

The *instance pattern level* (Fig. 3.75d) deals with (abstract) instances rather than with types. Then, the process modeler may define an instance-level pattern which may be inserted in one step into a task net. An instance pattern for a whole subprocess is also called an *instance-level process definition.*

Consistency Control

Below, we discuss vertical *consistency* relationships between framework levels. As argued before, we assume consistency of process definitions w.r.t. the meta model.

Inconsistencies between process model instances and real-world processes are frequently caused by inadequate process models. The vast majority of process management systems demands consistency of the process model instance with the process model definition. As a consequence, the process model instance cannot be updated to represent the deviations taken by the process participants.

In our approach, we allow for inconsistencies between a process model instance and its definition. This way, the process model instance can match the real-world process as closely as possible. By default, a process model instance must be (strongly or weakly) consistent with its definition, but each subprocess can be set to allow for temporary inconsistencies (e.g., insertion of a task of some type that is not modeled in the respective process model). It is up to the process manager to decide whether these subprocesses containing *controlled*

Table 3.2. Potential consistency levels

	untyped	**partially typed**	**typed**
untyped	w	w	i
partially typed	i	w, i	i
typed	i	s, i	s, i

deviations finally have to be again consistent with their respective definitions or if they can be left inconsistent.

Like in object-oriented modeling, we distinguish between a structural model (defined by class and object diagrams) and a behavioral model (defined by state and collaboration diagrams). Accordingly, a process instance is *structurally (behaviorally) consistent* if it satisfies the constraints of the structural (behavioral) model.

We distinguish three levels of consistency ordered as follows: *inconsistent (i)* < *weakly consistent (w)* < *strongly consistent (s)*. We introduce the level of weak consistency to account for the use of unconstrained types (partial process knowledge). Table 3.2 summarizes potential consistency levels for combinations of instance- and type-level processes (rows and columns, respectively) depending on the respective degree of typing. For example, an untyped process instance is inconsistent with a typed process definition, which excludes the use of unconstrained types. Furthermore, a typed process instance is either strongly consistent or inconsistent with a typed process definition (weak consistency is only possible in the case of unconstrained types).

Process Evolution

In principle, *process evolution* may be considered at all levels of our conceptual framework though we assume a static meta model here to avoid frequent changes of the process management system itself.

Evolution of process instances is inherent to dynamic task nets. The process meta model is designed such that planning and enactment may be interleaved seamlessly. Evolution of process definitions is performed at the level of packages. To maintain traceability, packages are submitted to *version control*.

After new package versions containing improved process definitions have been created, *migration* of currently enacted process instances may be performed selectively. During migration, process instances are to be updated such that they are consistent with the improved process definition. It is crucial that temporary inconsistencies do not prevent migration because a consistent state is reached only eventually (if ever).

By permitting weakly consistent and inconsistent process instances, we may not only minimize the gap between process instance and real-world process. In addition, we support *bottom-up evolution* of process definitions by allowing for selective adaptation of incorrect or incomplete process models according to new experience expressed in the evolved process instance.

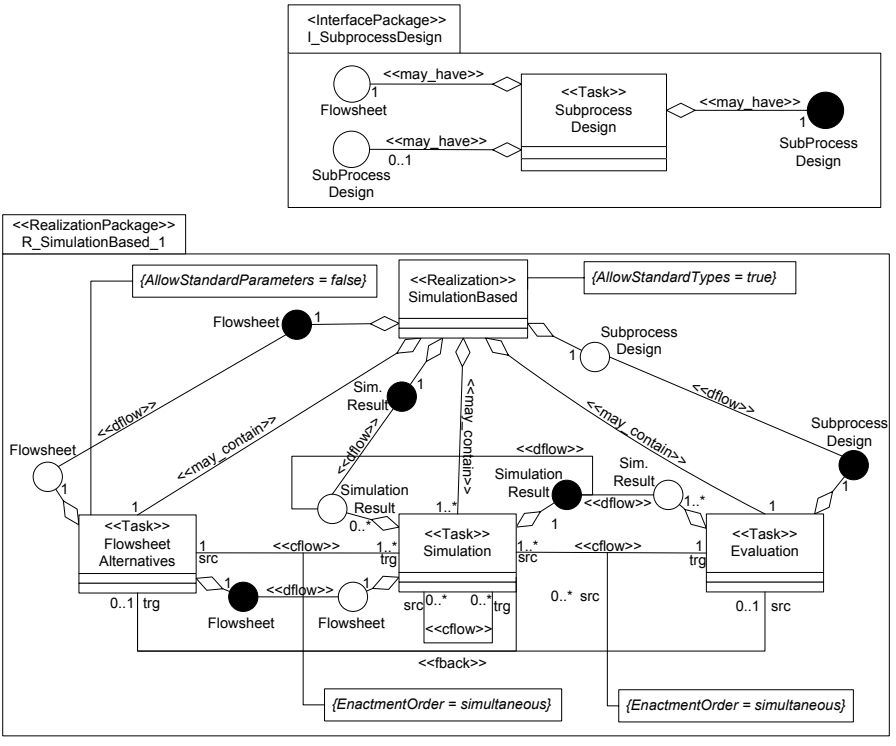

Fig. 3.76. Class diagram for a design subprocess

After having gained sufficient experience at the instance level, the process modeler may propagate these changes from the process definition to the instance level (*top-down evolution*). Changes may be propagated selectively, and inconsistencies may be tolerated either temporarily or permanently – depending on whether it pays off or it is considered necessary to reestablish consistency. Altogether, our approach supports *round-trip process evolution.*

Sample Process

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. In contrast to the previous section, we will deal with a different part of the overall reference process, namely the design of the separation (Sec. 1.2).

Figure 3.76 presents a process definition on the type level 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). This

version of the process definition will be replaced by an improved version later on.

The subprocess design is defined in two UML *packages* containing class diagrams for the interface and the realization, respectively. The class diagrams are adapted to the underlying process meta model by the use of either textual (<<Task>>) or graphical *stereotypes* (black/white circles for input/output parameters). Further meta data are represented by *tagged values* which are used to annotate model elements (shown as notes in the diagram).

The *interface* is defined in terms of inputs and outputs. A task of class SubprocessDesign receives exactly one flowsheet for the overall chemical process and (potentially) the design of a preceding subprocess. The output parameter denotes the result of the subprocess design, including the flowsheet 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. Although multiple realizations may be defined, we discuss only a simulation based realization (SimulationBased[35]). The inputs and outputs attached to SimulationBased are *internal parameters* which are used for vertical communication with elements of the refining task net.

The refining task net[36] comprises several serial and parallel tasks and is defined in a similar way as in Fig. 3.64.

Modeling elements are decorated with *tagged values* which define both structural and behavioral constraints. A few examples are given in Fig. 3.76:

- *Structural constraints.* The tag AllowStandardTypes is used to distinguish between partially and completely typed processes (Fig. 3.75b and c, respectively). Likewise, AllowStandardParameters determines whether a task may have untyped parameters.
- *Behavioral constraints.* The behavior of control flows may be controlled by the tag EnactmentOrder. The value simultaneous allows for the simultaneous activation of tasks connected by a respective control flow.

In addition to the process definition given above, the composition of subprocesses (PreStudy, ReactionDesign, SeparationDesign, Compounding and Decision; see top part of Fig. 3.77) into an overall design process has to be defined. When the overall process definition is available, a process instance is created according to the process definition. In the sequel, we will focus exclusively on separation design.

Initially, separation design is decomposed into a task for designing FlowsheetAlternatives and a final Evaluation task for comparing these alternatives. Now, the following problem is recognized: In order to design the separation additional data on the reaction are required. Waiting for these data would

[35] The <<may_realize>> association to the corresponding task class (see Fig. 3.64) was omitted from the figure.

[36] Data flows along feedback flows were omitted to keep the figure legible.

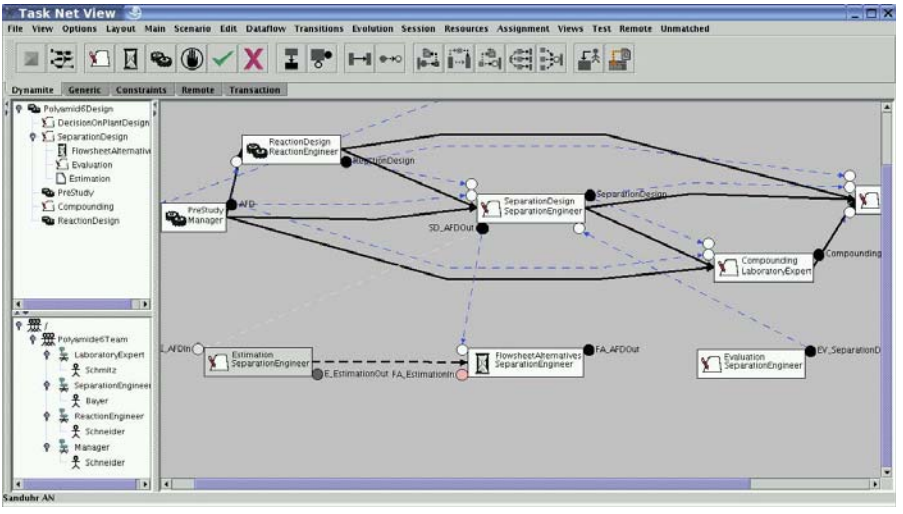

Fig. 3.77. Task net with inconsistencies

severely slow down the design process. Therefore, the manager of the separation design calls for and inserts an initial Estimation of these data so that design may proceed using initial estimations until more detailed data finally arrive.

Since the Estimation task is not defined in the class diagram, the manager makes use of the unconstrained type Task to insert it into the task net as an untyped task. This modification degrades the consistency level of the task net to weakly consistent. Unfortunately, inconsistencies are introduced, as well: The flowsheet design task has to be supplied with the estimation as input parameter. This was excluded in the process definition by the value false of the tag AllowStandardParameters. Therefore, the manager has to switch off consistency enforcement explicitly to make the modification feasible.

Figure 3.77 illustrates how weak (grey) and strong inconsistencies (red) are signaled to the manager[37].

Execution may continue even in the presence of inconsistencies. Further modification, like the insertion of parallel tasks SimulationDistillation and SimulationExtraction may cause additional inconsistencies.

At this stage, it is decided to clean up the process definition so that it includes the recent process improvements. 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 Fig. 3.76 is revised (see Fig. 3.78, where the changes are emphasized in bold face).

[37] Unfortunately, there is hardly any difference between grey and red in grey-scale reproduction.

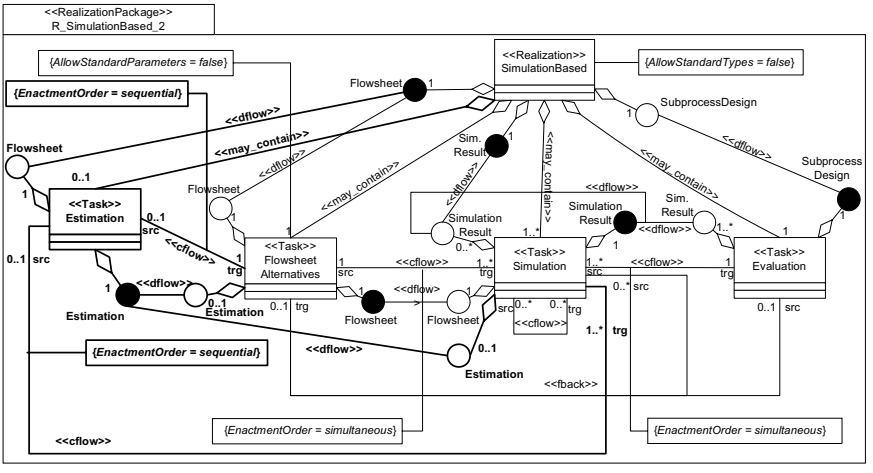

Fig. 3.78. Revised process definition

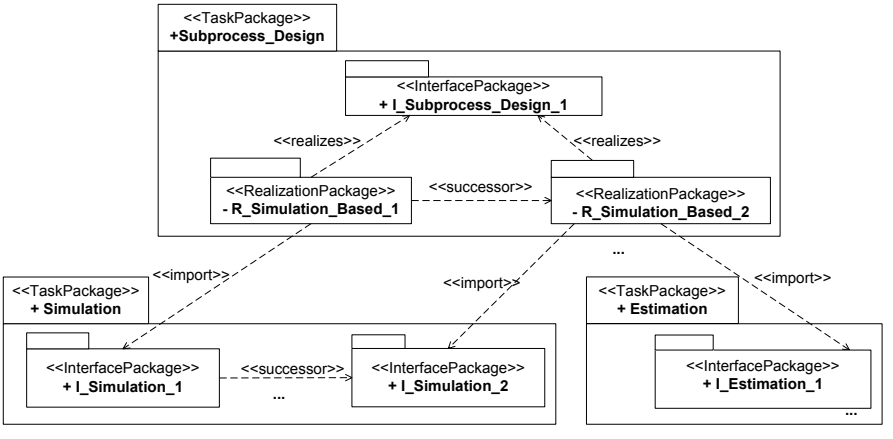

Fig. 3.79. Package versions

Figure 3.79 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. Finally, the interface package for the simulation task has to be revised such that the simulation task may be supplied with an estimate of a preceding subprocess.

The process evolution roundtrip is closed by propagating the changes at the definition level to the instance level. In general, migration has to be performed

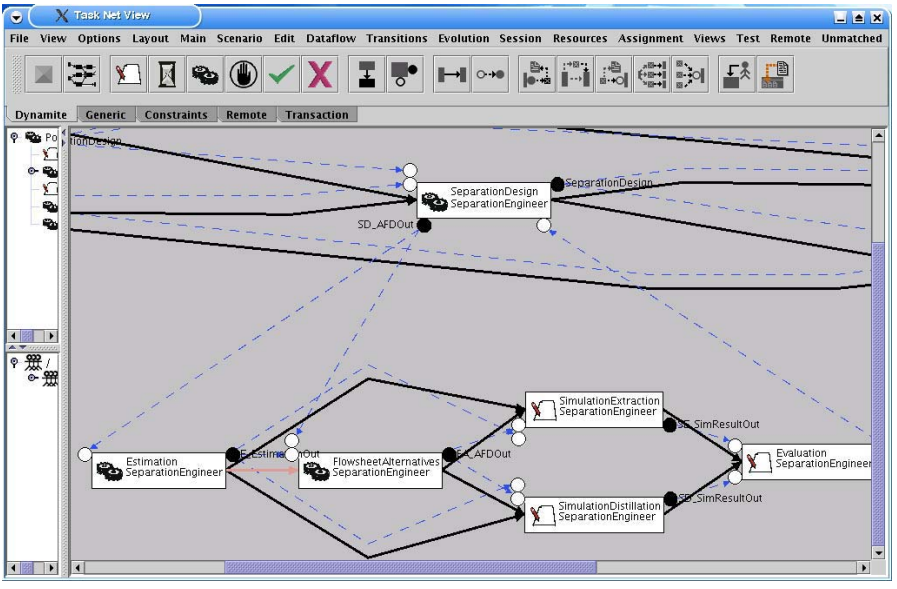

Fig. 3.80. Task net after migration

interactively since it is not always possible to uniquely determine e.g. the target type of migration.

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. In contrast, the estimation task's target type cannot be determined uniquely since it was introduced as an untyped task. After all objects have been migrated, the relationships can be migrated automatically. This is possible even for untyped relationships provided that there is only one matching relationship for each pair of object types.

The task net after migration is shown in Fig. 3.80. The control flow from the estimation task to the flowsheet design task is marked as behaviorally inconsistent (emphasized by red color). Both tasks are currently active, while the revised process definition prescribes a sequential control flow. This illustrates that migration does not necessarily result in a task net which does not contain inconsistencies. Migration can always be performed – even if inconsistencies persist.

Realization

Figure 3.81 displays the architecture of AHEAD extended by details of the modeling environment concerning process evolution (see upper right corner).

The process modeler uses a commercial CASE tool – Rational Rose – to create and modify process definitions in the UML. Rational Rose is adapted

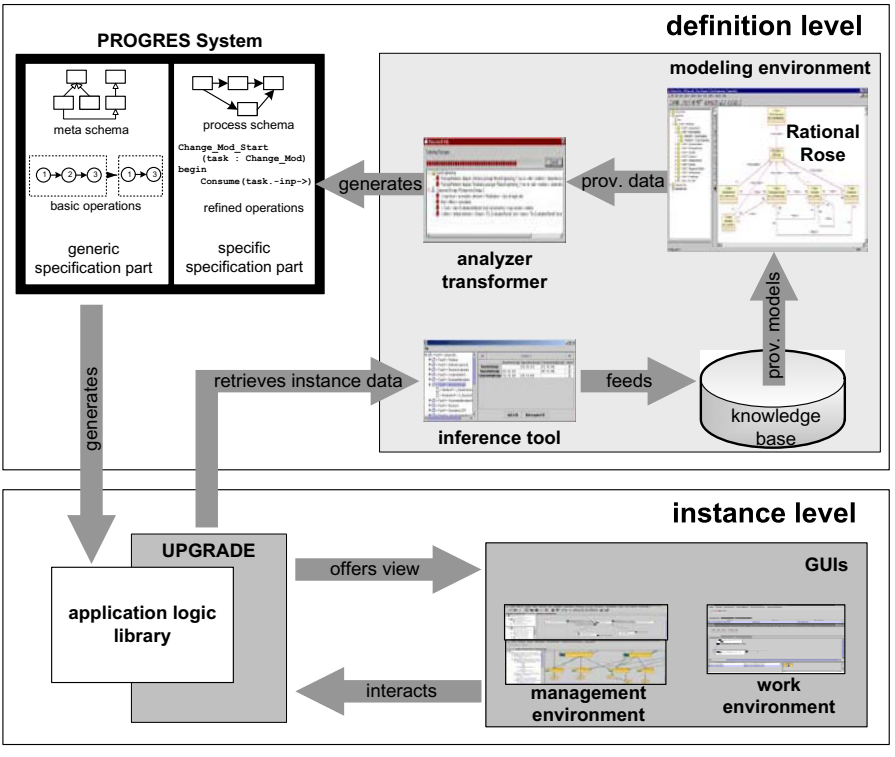

Fig. 3.81. Architecture of the AHEAD system with process evolution support

with the help of stereotypes which link the UML diagrams to the process meta model. A class diagram is represented as shown in Fig. 3.76. An analyzer checks process model definitions for consistency with the process meta model. The analyzer is coupled with a transformer which translates the UML model into an internal representation hidden from the process modeler [211].

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 [390].

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.

3.4.4 Delegation-Based Interorganizational Cooperation

So far, we have assumed tacitly that the overall design process is performed within one company. However, there are many examples of processes which are distributed over multiple organizations.

We have developed a delegation-based model for *cooperation between companies* and a generalization thereof. We first concentrate on the delegation-based cooperation and introduce a scenario for this kind of interorganizational cooperation.

Delegation of Subprocesses

Figure 3.82 illustrates the key components of the *distributed AHEAD system* [30, 166, 167, 169, 208]. The local systems are structured as before; for the sake of simplicity, the modeling environments are not shown. The extension of AHEAD to a distributed system is illustrated by the arrows connecting different instances of the AHEAD system.

AHEAD may be used to *delegate* a subprocess to a subcontractor. In general, a delegated subprocess consists of a connected set of subtasks; delegation is not confined to a single task. When the subcontractor accepts the delegation, a database is created which contains a copy of the delegated subprocess. Subsequently, execution of the subprocess is *monitored* such that the *contractor* may control the progress of work performed by the *subcontractor*.

The *delegation model* underlying the AHEAD system meets the following *requirements*:

- *Delegation of subprocesses.* A delegated subprocess consists of a connected set of subtasks. This way, the contractor may define *milestones* for controlling the work of the subcontractor.
- *Delegation as a contract.* The delegated subprocess serves as a contract between contractor and subcontractor. The contractor is obliged to provide the required inputs, based on which the subcontractor has to deliver the outputs fixed in the contract.
- *Autonomy of contractor and subcontractor.* The autonomy of both parties is retained as far as possible; it is restricted only to the extent required by the contract.

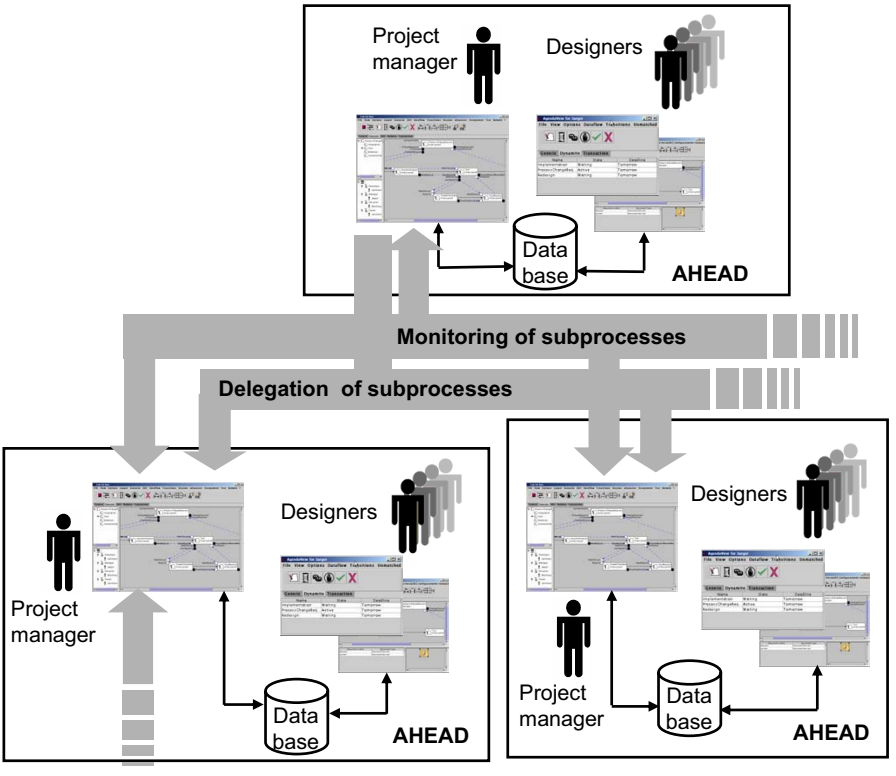

Fig. 3.82. Distributed AHEAD system

- *Need-to-know principle.* The parties engaged in a contract share only those data which are needed for the contract. This includes the respective subprocess as well as its context, i.e., its embedding into the overall process. Other parts of the process are hidden.
- *Refinement of delegated subprocesses.* The subcontractor may refine delegated subprocesses if this is required for managing the local work assignments. Since these refinements are not part of the contract, they are not visible to the contractor.
- *Monitoring of process execution.* The contractor is informed continuously about the state of execution of the subprocess delegated to the subcontractor. In this way, the contractor may monitor execution and control whether set deadlines are met.
- *Support of dynamic design processes.* Support for process dynamics is extended to interorganizational design processes. In particular, contracts can be changed dynamically. However, this requires conformance to a *change protocol* because cooperation among different enterprises requires precisely defined formal rules. The change protocol ensures that the contract may be changed only when both involved parties agree.

Delegation is *performed* in the following *steps*:

1. *Export.* The contractor exports the delegated subprocess into a file. A copy of the delegated subprocess is retained in the database of the contractor.
2. *Import.* The subcontractor imports the delegated subprocess, i.e., the file is read, and the local database is initialized with a copy of the delegated subprocess.
3. *Runtime coupling.* The AHEAD systems of contractor and subcontractor are coupled by exchanging events. Coupling is performed in both directions. This way, the contractor is informed about the progress achieved by the subcontractor. Vice versa, the subcontractor is informed about operations relevant for the delegation (e.g., creation of new versions of input documents).
4. *Changing the contract.* The contract established between contractor and subcontractor may be changed according to a pre-defined change protocol. The change is initiated by the contractor, who issues a change request. In a first step, the proposed change is propagated to the subcontractor. In a second step, the subcontractor either accepts the change – which makes the changes valid – or rejects it, implying that the propagated change is undone.

Please note that steps 1–3 are ordered sequentially. Step 4 may be executed at any time after the runtime coupling has been established.

Sample Process for Delegation-Based Cooperation

Scenario

When designing a chemical plant, expertise from multiple domains is required. For example, in the case of our Polyamide-6 reference process experts from *chemical engineering* and *plastics engineering* have to cooperate. Plastics engineering is needed to take care of the last step of the chemical process, namely compounding, which is performed with the help of an extruder.

The scenario to be discussed below involves two companies. The overall design of the chemical plant is performed in a chemical company. Compounding is addressed by an plastics engineering company. Designers of both companies have to cooperate closely with respect to the separation step of the chemical process since separation can be performed partly still in the extruder. In Fig. 3.83, a detailed extruder configuration including the polymer feeding, a mixing section, a degassing section followed by the fiber adding and the degassing of air is shown. The substances fed into the extruder still contain a small fraction of monomers which are fed back into the reaction step. Thus, a major design decision concerns the question to what extent separation can still be performed in the extruder.

Further on we will concentrate on the *delegation* of the activities from chemical engineering to plastics engineering. The compounding expert receives

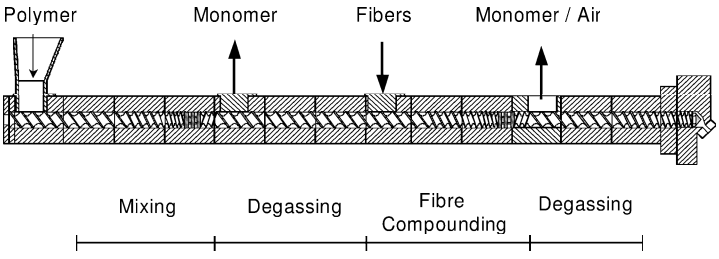

Fig. 3.83. Functional sections in a compounding extruder

the compounding steps and information about process boundary conditions such as mass flow, estimated viscosity, and thermophysical polymer properties (e.g., heat capacity, thermal conductivity). Afterwards he estimates compounding specific process parameters like the machine size, the extruders' rotational speed, the mass flow in every extruder, and the number of needed extruders.

Because the degassing process in the extruder can be quantified only with high experimental effort or by a simulation program [147], at first the degassing section is investigated by the compounding simulation expert. In a meeting, all necessary tasks are discussed and afterwards the compounding simulation expert starts a calculation to quantify the amount of degassed monomer while the compounding expert estimates the process behavior for the fibre adding section by his experience based knowledge. In the following meeting, first design results are discussed with the separation expert representing the chemical company. This collaboration for the design of the separation process is necessary, because the separation of volatile components like monomers and solvents from the polymer is possible both in e.g. the wiped film evaporator and the compounding extruder as mentioned above.

As a result of the interdisciplinary meeting, the members decide to make a detailed analysis of the homogenization processes in the mixing section by use of 3D-CFD tools (Computational Fluid Dynamics). Afterwards the results are discussed among the plastics engineers in a second meeting to prepare a report for the chemical engineering contractor.

The parallel activities in chemical and plastics engineering require powerful and smart management tools which can handle the highly dynamic concurrent processes. If any of the analyzed process steps turns out to be not feasible or not economically reasonable, various activities can be affected and a large part of the complete project has to be reorganized or in the worst case canceled.

Initial Situation

The example session described here deals only with the part of the overall design process which is related to the design of the extruder. The chemical company acts as a contractor and delegates the task of designing the extruder component to its subcontractor, the plastics engineering company.

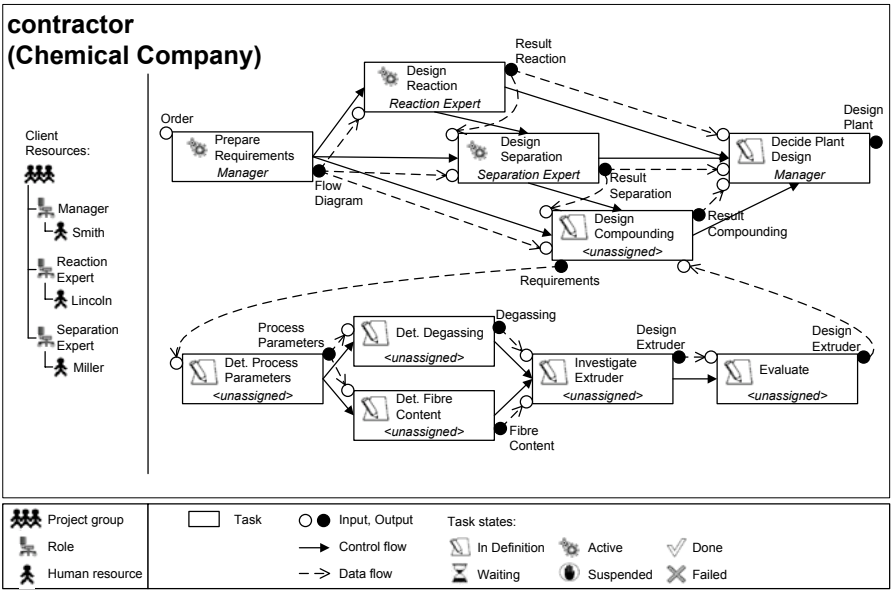

Fig. 3.84. Task net after refinement of task Design Compounding

The task net in Fig. 3.84 results after the manager of the chemical company, acting as the contractor, has refined the extruder design task by a subnet. The task Design Compounding and all subtasks will be delegated to the subcontractor. The task definition of Design Compounding can be seen as a *contract* between both companies, where the subcontractor has to produce a certain output (extruder design alternative) based on the inputs (requirements) which are provided by the contractor.

As stated before, in this subprocess an extruder is developed according to a set of desired product properties. The subtask Determine Process Parameters receives a product quality specification and the extruder's properties as input and produces rough estimates for the extruder's parameters. The content of fibres as well as the degassing of volatile components of the plastics are investigated in separate tasks. The subsequent investigation of the extruder's functional sections in task Investigate Extruder is based on the output of the three previous tasks. The results are evaluated and if the desired properties are met, the extruder design is propagated as a preliminary result to the parent task Design Compounding.

Establishing the Delegation

The delegated subprocess Design Compounding and its refining task net is are exported to a file. For further monitoring, all delegated tasks remain in the local data base and in the task net view on the contractor side but are assigned to a newly created resource Remote: Plastics Engineering Company. The

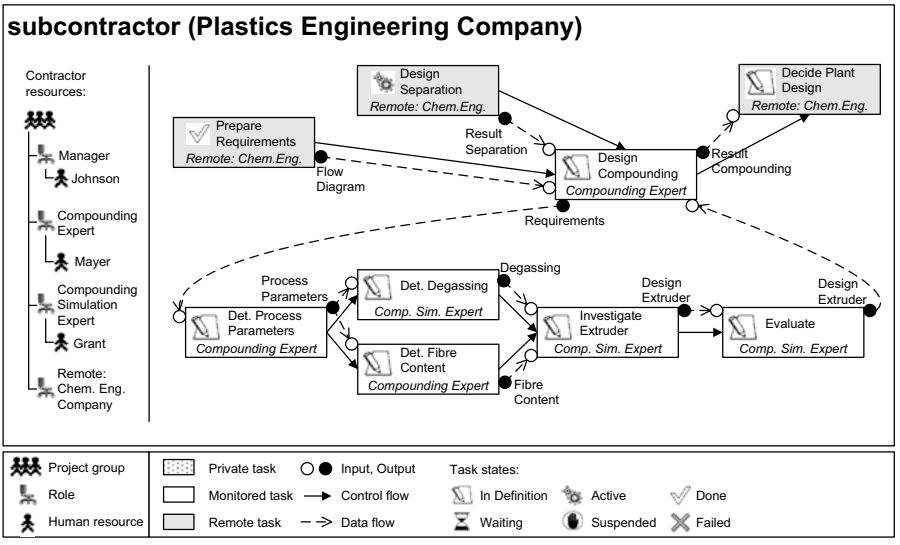

Fig. 3.85. Task net on subcontractor side after delegation and resource assignments

export file also contains contextual information about the delegated process, namely those tasks that are not delegated but connected via control flows to a delegated task (here: Prepare Requirements, Design Separation and Decide Plant Design). They can be *monitored* on the subcontractor side in contrast to *private tasks* that are not in the context (e.g. Design Reaction).

The plastics engineering company (subcontractor) imports the process description file into its AHEAD system. Figure 3.85 shows the corresponding task net and its context, which are instantiated in the local database on the subcontractor side. All delegated tasks are still in state In Definition, so that the manager on the subcontractor side can ask the contractor for a revised version if he does not agree with the contract consisting of delegated tasks, their parameters, control flows and data flows. If the subcontractor agrees to execute the delegated process, he may begin with the execution of the corresponding task net in his management environment, e.g. by assigning all delegated tasks to either the role Compounding Expert or Compounding Simulation Expert as shown in Fig. 3.85.

The management systems of contractor and subcontractor are loosely coupled together by exchanging events. The contractor is informed about changes of the delegated tasks' execution state which are considered *milestone* activities. Vice versa, the subcontractor is informed about changes of the context tasks which are executed on the contractor side. For instance, if Prepare Requirements is changed from Active to Done on the contractor side, a change event triggers the same change on the subcontractor side (cf. Fig. 3.85).

The delegated task Design Compounding is activated by a Compounding Expert. During execution of the delegated tasks, roles are assigned (e.g. Compounding Expert for Determine Process Parameters), task states are changed, and results are passed according to the defined data flows. All these updates of the delegated tasks by the subcontractor can be monitored by the contractor as well as the produced result, the first version of the extruder design.

Changing the Delegated Task Net Dynamically

In our example, the contractor and the subcontractor agree that the preliminary design alternative for the extruder could be optimized if the mixing quality of the materials in the extruder is investigated further. This is done by performing a three-dimensional simulation of the polymer flow in the extruder. The subcontractor agrees to carry out the additional simulation and the contract between contractor and subcontractor can be extended. Changing a delegated process after having started its enactment is not unusual in the design process of a chemical plant.

AHEAD supports dynamical changes of the contract between contractor and subcontractor. Changes are allowed only when both parties agree on them. Therefore, the delegated task net is changed according to a formal *change protocol*. The delegated task net is at every time in exactly one of the three *delegation states* Accepted, Change, and Evaluate. As described below, the transitions between these states define the commands which can be executed either on contractor and subcontractor side during the change process.

Initially, the subcontractor has issued the command Allow changes (from Accepted to Changed) to signal that he agrees to the change proposal of the contractor. After that, the contractor is able to modify the delegated process. The contractor adds a new task Determine Mixing Quality in the subnet of the Design Compounding and adds the appropriate control and data flow relationships from Determine Process Parameters and to Investigate Extruder. While the contractor changes the task net, all changes to the task net are propagated to the subcontractor. Eventually, the contractor may either discard his changes by using the command Reset Changes (Changed→Accepted) or he may signal that the structural changes are finished by using the command Changes Finished (Changed→Evaluate).

In our example, the subcontractor evaluates and accepts the proposed changes of the delegated process. Triggering the command Accept Changes (Evaluate→Accepted) yields an update of both processes on the contractor side and the subcontractor side according to these changes. As an alternative, the subcontractor may reject the change of the contract by use of the command Reject Changes (Evaluate→Changed). In this case, the changes are discarded and the contractor would be informed about the rejection. Both partners then would have to talk about the problem again before eventually the subcontractor would accept a proposal made by the contractor.

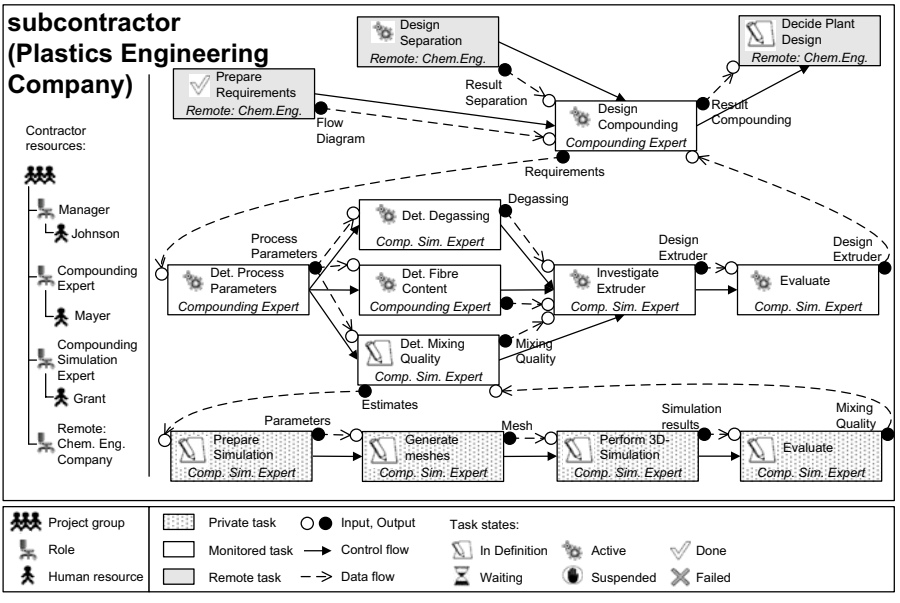

Fig. 3.86. Private refinements of the delegated process on the subcontractor side

Information Hiding Regarding Delegated Processes

The complex new task Determine Mixing Quality in the delegated process is refined by the manager on the subcontractor side by a private subnet to break it down into smaller working units and assign separate resources to each of the tasks. This refining task net comprises the tasks Prepare Simulation, Generate Mesh, Perform 3D-Simulation and Evaluate as shown in Fig. 3.86. The subnet is not part of the contract between contractor and subcontractor and can therefore be hidden from the contractor by means of private tasks.

Finishing the Delegation

After the process has been resumed, on the subcontractor side a second version of the extruder design has been finally produced and released to the task Design Compounding. This result should be taken as the final outcome of the delegated task. The subcontractor can signal this to the contractor with the command Complete Delegation stating that he wishes to complete the contracted delegation relationship. The contractor can confirm this with the command Confirm or reject it with the command Reject. If the result is accepted, the coupling of the two AHEAD systems of contractor and subcontractor is finished. In the other case, the rejection is signaled to the subcontractor and the coupling is maintained.

3.4.5 View-Based Interorganizational Cooperation

Motivation

In the previous subsection, a *delegation-based* relationship between cooperating organizations has been explained, where a contractor delegates a part of his process to a subcontractor organization. This model is now criticized. We want to *generalize* the *model* in order to support a broader spectrum of cooperation scenarios.

The previously discussed delegation-based relationship is restricted with respect to its flexibility and *adaptability* to different cooperation scenarios:

- The *visibility* of elements in the contractor process for the subcontractor can only be defined for *tasks* in the *direct neighborhood* of the delegated process. Thus, it is not possible to expose process parts without a delegation-relationship.
- AHEAD currently supports only a delegation-relationship for the cooperation between processes where both parties have *different* roles during the collaboration (namely contractor and subcontractor) implying different rights to define all cooperation aspects. However, other possible cooperation scenarios should also be possible. For example, the *same* rights and duties can be given to the partners of a *peer-to-peer* cooperation.
- Only *connected* parts of a process can be *delegated*. If multiple parts of a process are delegated, they are all regarded as independent new processes on the subcontractor's side. They cannot be composed into an overall process with a shared process context. Following this approach, the integration of pre-existing processes with each other is not possible.
- Cooperation can require *less formal* or *more formal* configurations regulating the procedures and mechanisms used by the organizations for defining, executing, and evaluating interorganizational processes. Therefore, flexible and configurable cooperative processes for interorganizational processes need to be supported by the AHEAD system. For example, not every delegation requires very strict and formal contracts about the agreements and procedures between the partners. Currently, the cooperation protocols for delegation within AHEAD are built-in and cannot be tailored to specific cooperation needs according to a higher or lower level of trust between the cooperating organizations.

Hence, we have identified two important *requirements* for flexible cooperation support in dynamic development processes: (1) An organization should be able to use powerful and flexible mechanisms for defining the *visibility* of process information shared with other organizations. (2) Interoganizational cooperation has to be supported insofar as the different processes of the cooperation partners can be *integrated* with each other not exclusively according to delegation-based relationships between them. A broad set of *customizable cooperation relationships* has to be supported instead.

Dynamic Process Views

Our approach to interorganizational cooperation in development processes builds on the definition of *dynamic process views* onto development processes as its foundation [175–177]. Dynamic process views support better visibility management for process elements carried out within an organization.

A dynamic process view is defined for a *process instance* (i.e. a dynamic task net) with its products and resources and it resembles a *subconfiguration* of the *process instance*. A process view constitutes a certain cut-out of its underlying process with products and resources which should be made visible to external parties.

A process view basically *contains* the following *elements*:

- A *view name* and a unique *view identifier*.
- A *subgraph* of a dynamic task net (partial abstraction): This subgraph represents a fragment of a dynamic task net which is structurally and behaviorally consistent with respect to the process meta-model of DYNAMITE [243]. Zero or more tasks can be part of the process view and not all of a task's parameters need to be in the process view. Only a subset of the existing flow relationships between tasks needs to be represented in the process view.
- A *view product workspace* maintaining all view-related products and product versions which are contained in the underlying process and should be visible within the process view.
- A*view resource space* which contains all view-related resources, i.e., all abstract or concrete resources of the underlying process which should be made visible within the process view.
- *View definition rules*: A set of rules defines which elements of the underlying private process are also part of the process view. For instance, some specific model elements (i.e. tasks, products, resources, or flow relationships) can be assigned to the view, or all model elements of a specific type can be chosen instead.

The process view concept is *illustrated* by an *example* in Fig. 3.87. The top part of the figure shows a part of the Polyamide-6 process (used throughout the entire subsection) as it is seen from the perspective of the Chemical Company. In the middle part of Fig. 3.87, a process view definition named ReactionSimulationTasks for this process is shown which contains two simulation tasks Simulate CSTR and Simulate PFR with their input and output parameters from the overall process, while the control flow between both tasks is not included in the process view.

Process views can be used to provide *different perspectives* of the underlying private process. For instance, managers can use process views to gain overview with minimum technical process information. Technical experts can use process views containing all necessary process information with respect to

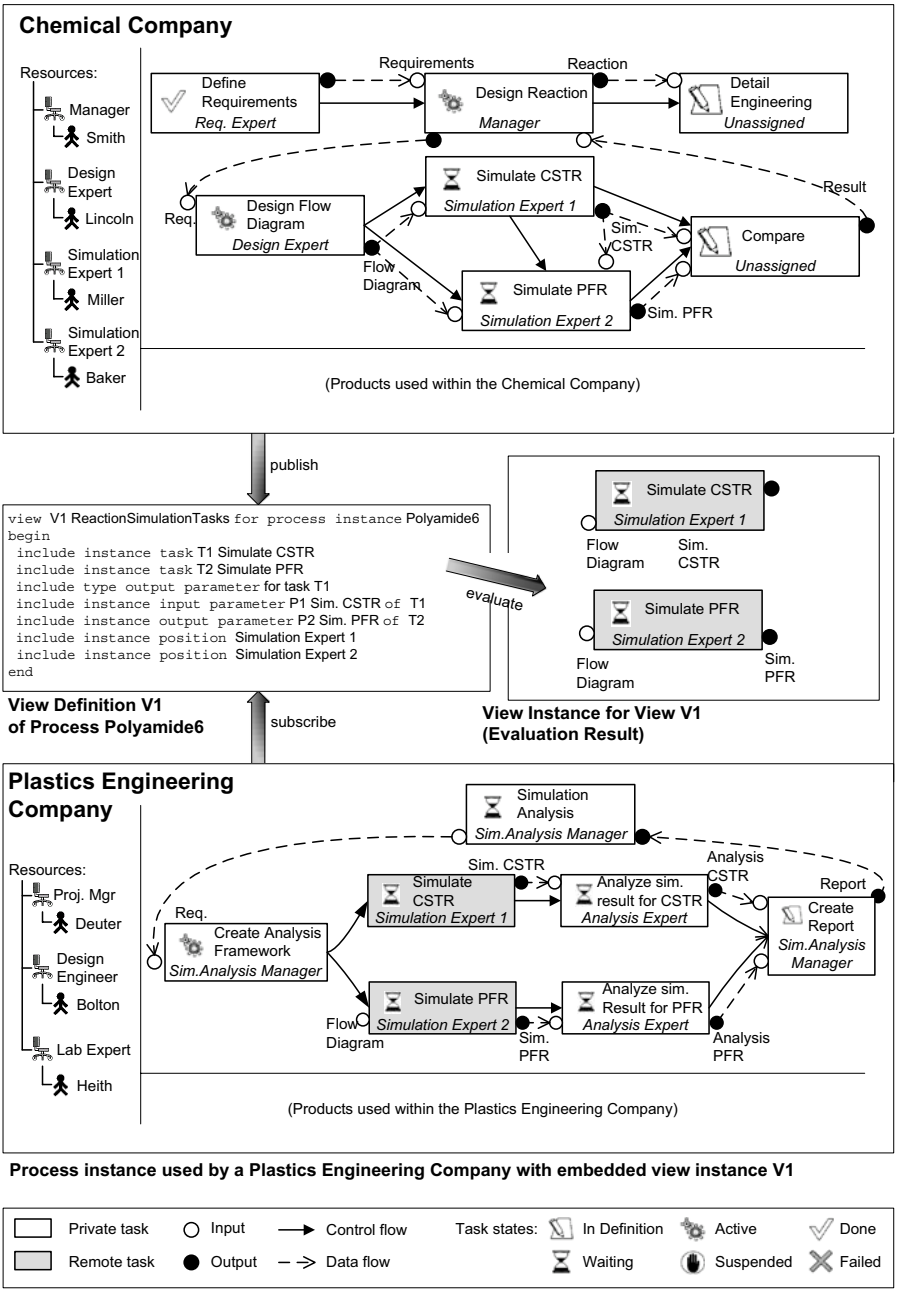

Fig. 3.87. Dynamic process view for the Polyamide-6 process

a specific information need. In our opinion, a view-based approach is a natural approach for managing the visibility of process elements to external parties.

Process view definitions are *published* by one organization (publisher) and they can subsequently be *subscribed* by other organizations (subscribers). The application of a view definition to a process results in a *view instance* containing all process elements which are visible according to the (automatic) evaluation of the view definition rules of the subscribed process view. The subgraph, view workspace, and view resource set of a process view exactly contain all process elements which are determined by the view definitions rule set.

Subsequently, the process elements of a view instance can be embedded into private process instances of the subscriber. Thereby, any new restrictions on the embedded elements, e.g. new incoming control flows, have to be negotiated between the subscriber and publisher before they can take effect.

Process view instances change, either when the underlying private process or when their corresponding view definition are modified. Therefore, view instances are *re-evaluated* whenever the underlying processes or the view definitions are changed, in order to update the contents of the process view. For example, the lower part of Fig. 3.87 shows the private process of the Plastics Engineering Company with the embedded view instance of View 1.

Private processes contain *local* as well as *remote tasks* embedded from other organizations within process views. The embedded view elements (here tasks Simulate CSTR and Simulate PFR) can be *interconnected* with tasks of the private process where the view is embedded by control flows, feedback flows or data flows to establish inter-process cooperation. This provides the basis for interorganizational cooperation as explained in the following paragraph.

View-Based Interorganizational Cooperation Model

We are now in the situation to introduce our *interorganizational cooperation model* which is based on dynamic process views [175–177]. The model is described according to *three* layers which are located on top of each other starting at the bottom of the layer stack (Fig. 3.88):

- The private processes are modeled on the *private process layer*, where the process manager of each organization defines, controls and monitors a task net instance reflecting the development process within the respective organization.
- Dynamic process views are located at the *process view layer* above. Parts of the overall process within each organization are made externally visible by the definition and publication of one or more process view definitions. These process view definitions are subscribed by other organizations, where the respective private processes are extended with the contents of the corresponding view instances. In our approach, the remote process view elements are directly embedded into the private task nets to allow

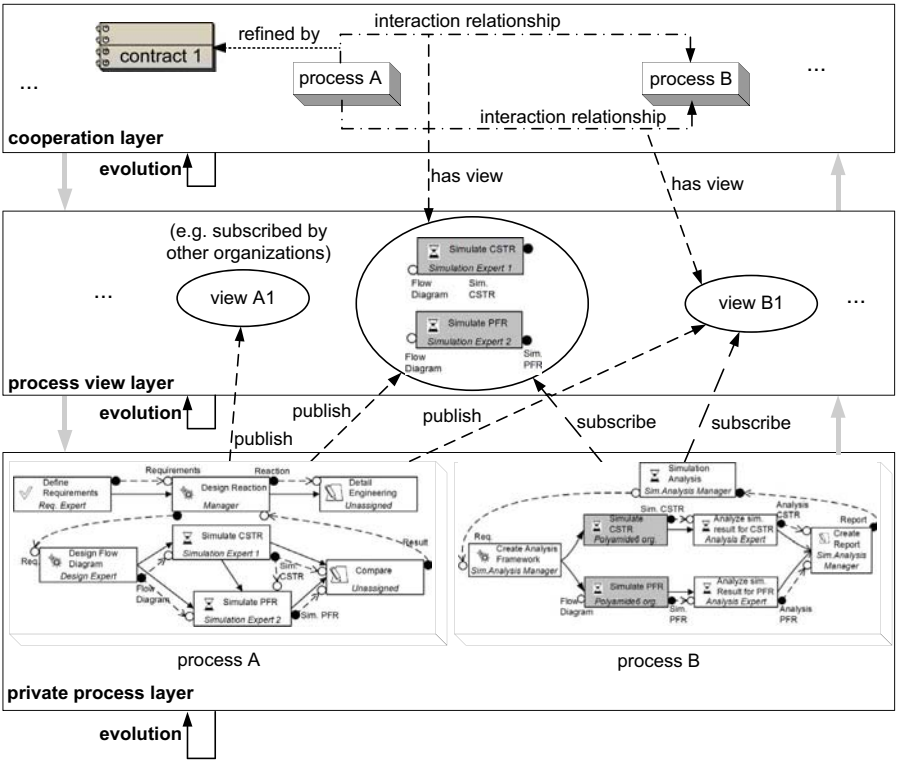

Fig. 3.88. Layers of the view-based interorganizational cooperation model

for a complete overview of all process elements together. Process views are used to enable inter-process coupling as described below.

- Details about the intended cooperation relationships between organizations are contained in the *cooperation layer* on top of the process view layer. Different kinds of *cooperation relationships*, e.g. outsourcing relationships, are introduced here (described later). Cooperation relationships model the interactions between project teams residing in different organizations and they prescribe how control and data can be transferred between the organizations.

Our approach to interorganizational view-based cooperation management comprises the following *cooperation phases*:

1. *Private task net planning.* Within each organization, the process manager plans its own private process instance.
2. *Process view publication.* The process manager creates process views to make certain cut-outs of his managed process instance externally visible. View definitions with view rules on the instance-level and the type-level

are created and subsequently published to other organizations. Selected tasks of a process view definition can be marked as *outsourced* in order to execute them within other organizations.

3. *Process view subscription.* The process manager of another organization subscribes the published process view definition to embed a corresponding view instance into his private process instance (dynamic task net).
4. *Process inter-connection.* By connecting private and remote process elements with each other, process instances are coupled across organizational borders.
5. *Cooperation policy definition.* The process manager can define the cooperation relationship on the cooperation layer. Additionally, he can assign selected process view definitions to the cooperation policy. Each relationship can be further refined with a contract, if needed.
6. *Process coupling.* The AHEAD systems of the cooperating organizations are coupled to exchange process update messages with each other. The different processes are executed locally in the organizations. All process views are updated upon changes of the underlying process instance. Whenever a process instance is modified locally, the respective AHEAD system computes all affected process views. Subsequently, it notifies all those remote AHEAD systems about the change, wherein the computed process views are embedded.
7. *Completion of process inter-connection.* The process manager of each organization decides autonomously when to terminate or cancel the process interconnection. Therefore, he marks a selected process view definition as completed. After that, the corresponding view instance is not updated any more and the process instances evolve independently from that moment on (although all embedded view elements remain in the private processes of all subscribers).

Layers and Components of the Cooperation Model in Detail

We now *describe* the different *layers* and components of our cooperation model in the following four subparagraphs in more detail.

Private Process Layer

Within the private process layer we allocate the process instances of each organization. Of course, all process aspects are visible within the organization. But due to a lack of trust, in most cases it is not suitable to expose private process details completely to other parties but only a certain fraction of the overall process. For this purpose, the process view concept is introduced.

Process View Layer

All process views are located on this layer above the private process layer. Process views are used to enable inter-process connection. The private processes can contain tasks which are executed locally as well as tasks which are

embedded locally using subscribed process views from other organizations. A private process can contain local process elements as well as remote process elements. Therefore, process managers can oversee their local process together with all connections to process parts executed in other organizations within a single task net representation.

In order to achieve inter-process coupling, the process instances of the cooperating organizations can be connected by control flows, feedback flows, or data flows. We do not need an additional modeling language for modeling the coupling of processes. Instead, we re-use the known control flow, feedback flow, and data flow concepts of dynamic task nets. While other approaches favor to model intra-organizational and inter-organizational control and data flows differently, we aim at modeling both in the same way. From a manager's point of view, control flow is transferred between two tasks regardless if they both are locally executed or not. Of course, intra- and inter-organizational dependencies between tasks have to be handled differently, but they can be modeled the same way for ease of use. Intra-organizational and inter-organizational flows can be identified, because the source and target tasks of these flows are either both local tasks or not. We believe, that modeling intra-organizational as well as interorganizational cooperation in a uniform way is feasible and should be supported by a modeling approach that is simple to understand and to use by process managers.

Cooperation Layer

On the cooperation layer, we model *basic cooperation relationships* between processes. This model represents which connections to other organization's processes exist and how they relate to each other. We distinguish between monitoring relationships, interaction relationships, and outsourcing relationships.

A process view instance can be embedded into a private process of an organization in order to observe the progress of the process cut-out visible by that process view. Such *monitoring relationships* are the simplest form of cooperation because no direct inter-connection of process elements from different organizations is needed here. Using monitoring relationships helps process managers to oversee their own processes as well as interesting remote process parts in one uniform representation.

If local and remote process elements are connected by control flows, feedback flows, or data flows, we model *interaction relationships* between the coupled organizations on the cooperation layer. Interaction relationships resemble situations where control flow or data are transferred between processes. For example, local tasks can be restricted to start only after some remote tasks have terminated by inter-process control flows. This allows interweaving different processes in the sense that the processes are executed in parallel while they are loosely coupled at the same time.

We define *outsourcing relationships* to model cooperation in a customer-producer relation. In our approach, outsourcing means that an organization

(termed as customer) can plan single process tasks or a task net fragment to be executed by another organization (termed as producer) within the process view definition. The outsourced tasks are then transferred to the other organization and regarded there as a local task in the future. The outsourcing organization will no longer be responsible for the outsourced tasks, because they are executed within the other organization. In an extreme scenario, cooperation can even happen without outsourcing (or delegation) at all. This represents a useful scenario when different organizations cooperate with each other with the goal to allow access to selected parts of the private processes while prohibiting any further process coupling. Interaction relationships and outsourcing relationships can complement each other and can exist between two organizations at the same time.

Contracts

The extent of trust is a key factor in cooperations and must be modelled appropriately. For instance, if an organization wants to delegate a process to another organization, *different cooperation relationships* are possible. If the contractor has not worked with the planned subcontractor before, a very strict and formal cooperation setting may be suitable. If the partners know each other well, or if they are engaged within a long-term relationship, it may be more appropriate to work together in a less formal relationship, without fixing all details fixed within a contract beforehand.

In our approach, *contracts* are used to tailor cooperation relationships to individual cooperation needs. For example, the object of discourse, the different partner roles, or other data are defined within the contract. Additionally, selected process views can be assigned to the contract if the task net structure of some process fragments shall be a part of the contract between the cooperation partners.

A broad spectrum of cooperation scenarios can be realized with different contract configurations. On the cooperation layer, all three basic cooperation relationships (monitoring, interaction, and outsourcing) between processes are orthogonal to contracts. They can optionally be refined by contracts. The basic idea is to implement a very light-weight default contract protocol for the interaction between partners and to provide contracts as a means to further *define* the *fine-grained structure of a cooperation* between partners if this is needed. Formerly, only one fixed contract between a contractor and a subcontractor was supported in the delegation-based management approach of AHEAD.

Sample Process for View-Based Cooperation

We now demonstrate the view-based approach to interorganizational cooperation. In the example described below, we focus on the part of the overall design process which is related to the design of the reaction and separation as well as the design of the extruder. The Chemical Company acts as a *customer*

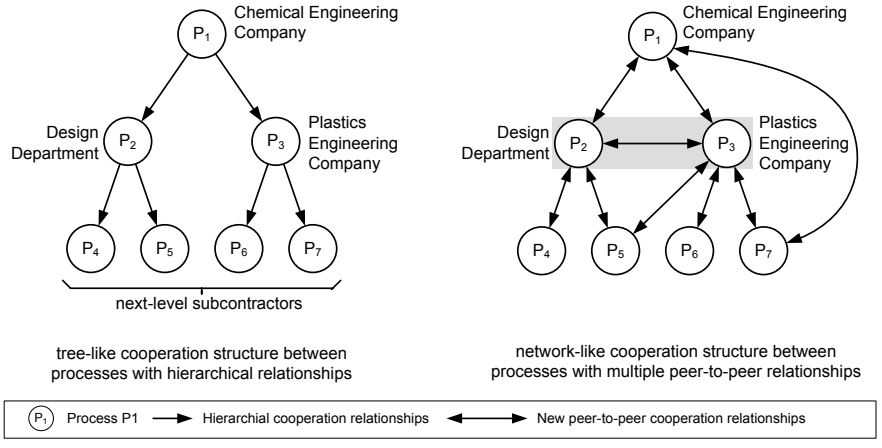

Fig. 3.89. Cooperation relationships in the example scenario

organization and *outsources* the task of *designing* the reaction and separation to another organization, the Design Department, having an own project manager who manages his own process, products, and resources independently. The task to design the *compounding* is outsourced to an Plastics Engineering Company.

The Chemical Company works together with its subcontractors, the Design Department and the Plastics Engineering Company, in outsourcing relationships. For the moment, we will deal with the situation after these outsourcing relationships have been established in order to show how a *direct cooperation* relationship between *both subcontractors* can be achieved with the process view concept. After that, we will explain how outsourcing relationships can be configured with process views.

This kind of direct cooperation between organizations resembles a *graph-like network cooperation* structure in a peer-to-peer mode which is not supported in the former delegation-based concept of AHEAD: Both subcontractors cannot cooperate with each other directly but only through their common contractor, the chemical company (shown in the left part of the Figure). In this way, only tree-like cooperation structures are possible. Although this delegation-based process decomposition approach is sufficient in many situations, often direct cooperation between all partners of a cooperative network of companies is needed as well.

Initial Situation

Fig. 3.90 shows the process part which has been delegated to the Design Department from the manager of the Chemical Company: Some tasks for the investigation of multiple reaction or separation alternatives will be carried out in the Design Department. The tasks Define Reaction Alternatives and De-

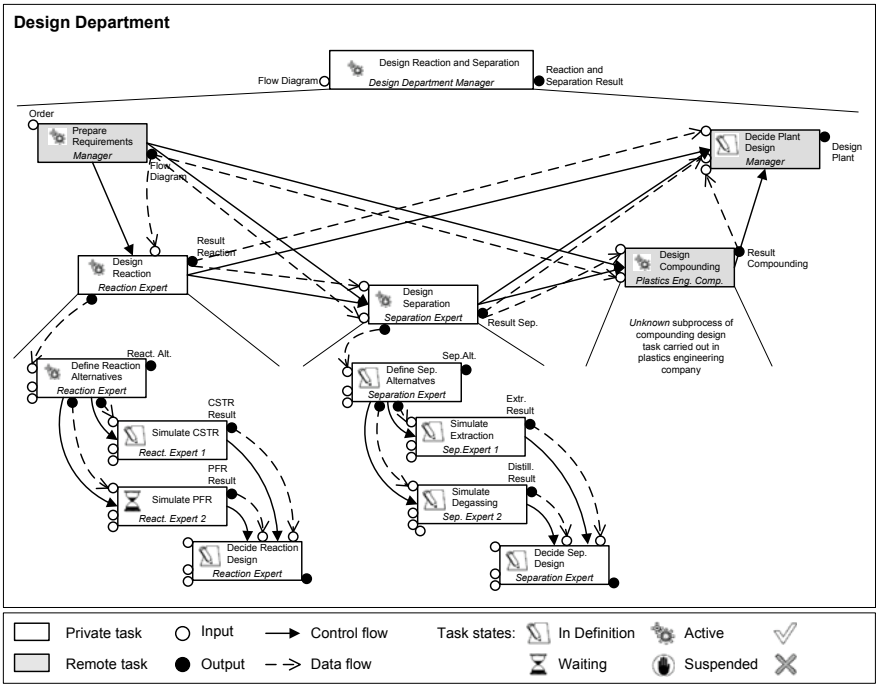

Fig. 3.90. Initial situation

fine Separation Alternatives have been refined, while four new tasks Simulate CSTR, Simulate PFR, Simulate Extraction, Simulate Degassing still need further refinement.

The subnet of the task Design Compounding contains several tasks (not shown in the figure): First, the expected output of the extruder will be roughly estimated in order to provide a starting point for the extruder design (task Determine Process Parameters). Second, separate tasks deal with the investigation of the fibers content and the degassing of volatile components of the plastics and all results will be used in the central task Investigate Extruder. Third, the extruder design is forwarded as a preliminary result to the parent task Design Compounding.

View Definition, Publication, and Subscription

The process manager of the Design Department can provide different views onto his private process (Fig. 3.91):

- A process view V1 provides information about the reaction part of the overall process. The tasks Define Reaction Alternatives and Simulate PFR for the simulation of the plug flow reactor is published for that purpose.

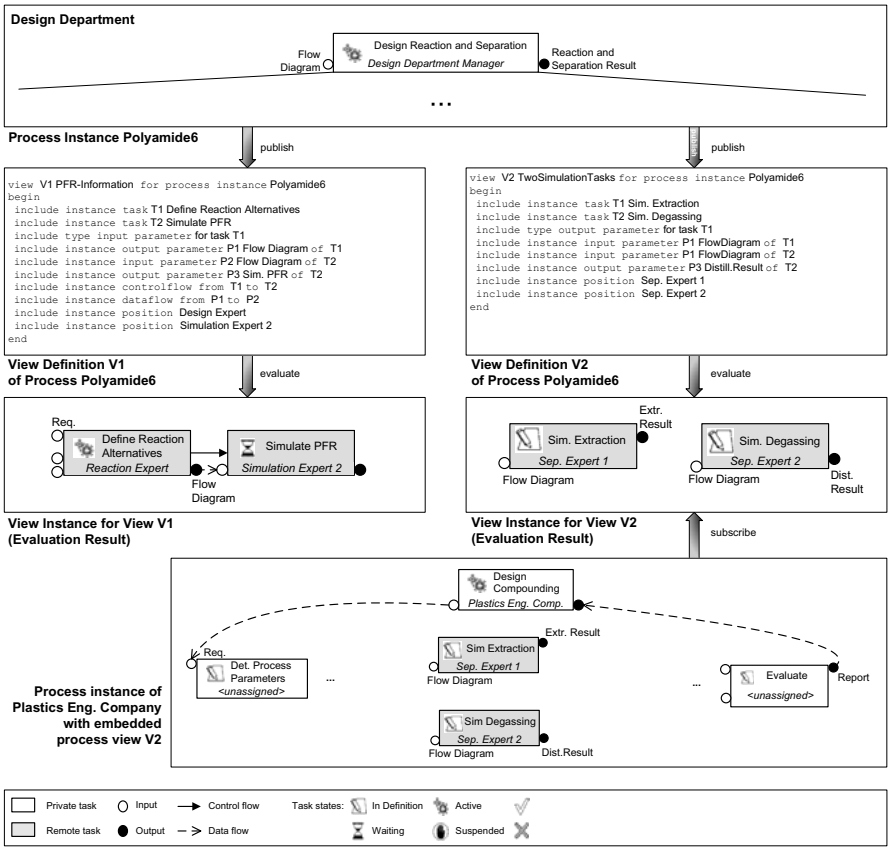

Fig. 3.91. Definition, publication and subscription of process views

- A process view V2 gives access to the simulation parts of the reaction design and contains the tasks Simulate Extraction and Simulate Degassing with some of their input or output parameters.

After both process views have been published, the manager of the Plastics Engineering Company can subscribe both process views and embed the corresponding process fragments into his own private process. This would result in the situation, that both processes are connected at two different locations (views V1 and V2) which can be planned and evolved independently. This demonstrates the advantage of our process view approach, where multiple cooperation contexts between both processes can be maintained simultaneously within logically separate process views. In the sequel, only the process view V2 is subscribed while the view V1 is neglected (lower part of Fig. 3.91).

Changes in the published process parts are transmitted particularly from the Design Department's AHEAD system to the Plastics Engineering Com-

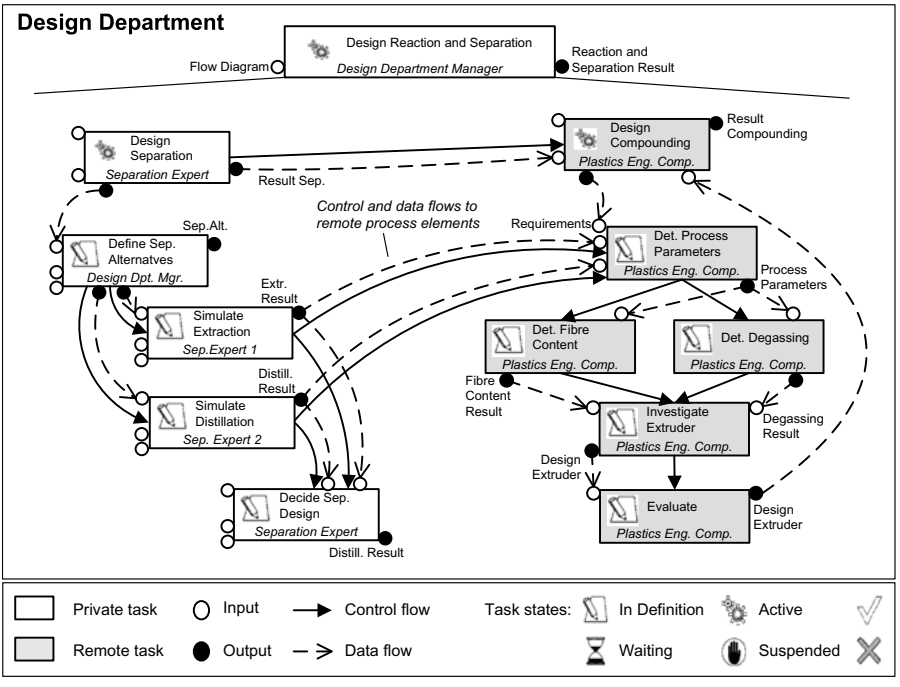

Fig. 3.92. Interconnections between Design Department process and Plastics Engineering process

pany's AHEAD system and displayed immediately. For example, the activation of task Define Reaction Alternatives on the side of the Design Department would be propagated to the AHEAD system of the Plastics Engineering Company through the process view V1.

Bottom-Up Process Composition

Both parallel processes in the Design Department and the Plastics Engineering Company can be inter-connected (Fig. 3.92). According to the plan of the process manager in the Design Department, the two simulation tasks Simulate Extraction and Simulate Distillation shall be synchronized with the task Determine Process Parameters of the other organization with respect to their execution states and documents shall be transferred between these tasks. This is an example of inter-organizational control and data flow.

The process manager of the Design Department creates new control flow and data flow dependencies between these tasks in his private process instance. It is important whether a flow dependency goes from a local task to a remote task or vice versa. *Locally relevant* inter-process dependencies between tasks (going out of a remote task into a local task) do not cause problems since they do not impose new behavioral restrictions on the remote tasks. But *remotely*

relevant inter-process dependencies (going from a local task into a remote task) are problematic. In our example, both new control flows going into the remote task Determine Process Parameters are remotely relevant and the intended changes are only allowed if the manager of the Plastics Engineering Company agrees to them.

The process manager of the Design Department can either re-use an already existing process view definition or create a new process view definition. Here, he decides to re-use the process view definition V2 and inserts all related tasks (Simulate Extraction, Simulate Distillation, and Determine Process Parameters) as well as all new control and data flows there. After that, he publishes the view (view definition evolution).

The manager of the Plastics Engineering Company subscribes the published view definition V2 (if it is not already subscribed there). Then he inspects the changes in the view definition. If he accepts them, the changes become persistent in both systems. The Design Department manager could also make modifications to the changed task net fragment under discussion. He can even choose to discard the modifications if no consensus can be reached.

After the changes have been carried out in both management systems, the managed process instances remain coupled with each other. Both process instances evolve autonomously and they are only loosely coupled with each other through the two newly inserted control and data flows between elements of both processes.

Top-Down Process Decomposition with Outsourcing

We now demonstrate the outsourcing of a task from a customer organization for execution within a producer organization. The manager of the Design Department requests the Plastics Engineering Company to investigate the different alternatives for separation as soon as possible. In this way, possible design flaws within the separation alternatives or their interplay with other design details can be detected very early. This helps to reduce the risk of far-reaching process feedbacks in later project phases due to closer communication between the partners in the beginning. Then, the manager of the Design Department creates a new task Investigate Distillation and outsources it to the Plastics Engineering Company (Fig. 3.93).

In this situation, he refrains from re-using an existing process view and creates a new process view V3a instead with the new task and its control and data flows to the tasks Define Separation Alternatives and Decide Separation Design. He marks the task Investigate Distillation as outsourced in the view definition. After that, he calls a command to add a minimal *context* of the outsourced task in order to maintain consistency with the surrounding task net. In the example, the context comprises the predecessor task Defines Separation Alternatives and the parameter Flow Diagram as well as the control and data flows to task Investigate Distillation. The process view V3a is published by the manager of the Design Department.

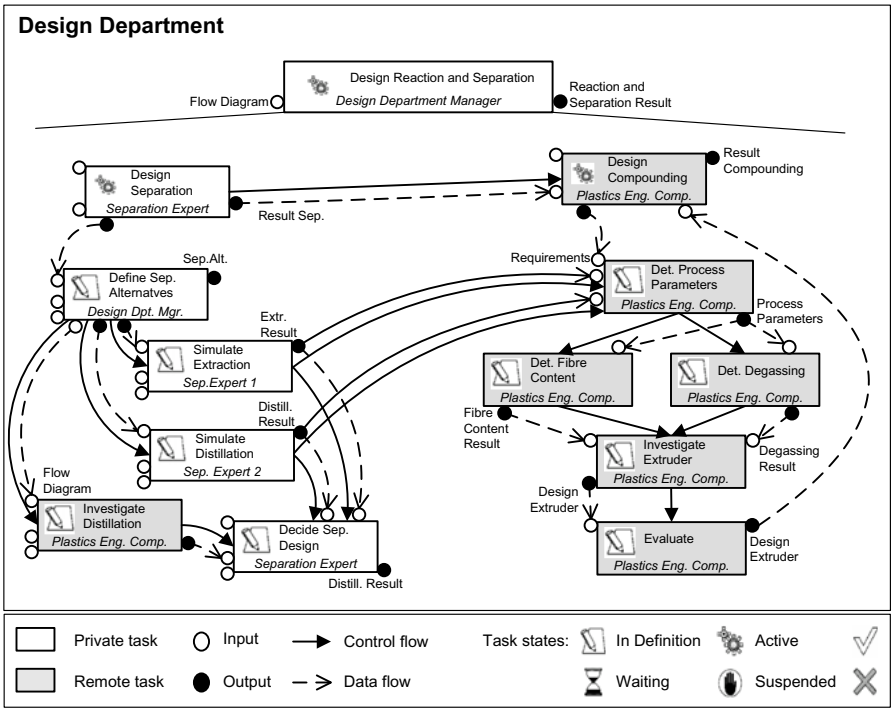

Fig. 3.93. Outsourcing of a task net fragment to external organization

Upon publication, the marking outsourced is detected and the system uses a special *outsourcing procedure* on both sides in the sequel. When the manager of the Plastics Engineering Company subscribes the process view V3a, he is asked to accept the process view as usual. When he accepts the view, he also accepts the announced task outsourcing therein. Then a new task instance Investigate Distillation is created in the private process of Plastics Engineering Company. This task is private upon creation and has to be published within a process view definition. Therefore, a new process view V3b (*back view*) is created and filled with the outsourced task and its context, where V3a and V3b are structurally the same, but the role of local tasks and remote tasks is reversed in the view. This new process view V3b is published to the Design Department. The manager there is also asked to accept this announced view. If he accepts also, then both managers have accepted the new cooperation situation and the outsourcing relationship between both processes is fully established. The outsourced task Investigate Distillation is executed by the Plastics Engineering Company. The other context tasks are not transferred between organizations so that they are executed by the Design Department as before. In this way, the delegation model presented in the previous section is simulated with the view model.

Integration of Workflow Processes in the Design Process

The view-based cooperation model presented so far addresses the inter-organizational integration of design processes carried out in several organizations. In AHEAD, design processes are represented by *dynamic task nets*, which may evolve continuously throughout the execution of a design process. We now extend the cooperation model with an approach for the intra-organizational integration of processes executed within heterogeneous process management systems, for example workflow management systems.

Although the overall design process cannot be planned fully in advance and thus cannot be executed completely within workflow management systems, this may be possible for some fragments of the overall design process (e.g. the design of an apparatus may be predictable). If the structure of such *static fragments* of the design processes is well-defined and most of the needed planning information is available, then these fragments can be specified in advance on a fine-grained level as a workflow process. Although workflow management systems have originally been designed to support repetitive business processes (e.g., in banks or insurance companies), the use of workflow management systems for design process support is investigated in other research projects, as well (e.g., in [832]). In contrast to workflow management systems, AHEAD supports the seamless interleaving of planning and execution – a crucial requirement which workflow management systems usually do not meet [475].

We have developed an approach to integrate workflow processes into the overall design process and have realized a coupling of workflow management systems with AHEAD for use within an organization. Our approach is characterized by the following properties:

- Within an organization, *AHEAD* serves as the central instance for the planning of the overall process, e.g. it is used for its *global coordination*. The composition of process fragments into a coherent overall process is realized using dynamic task nets, so that the dynamic character of the design process is adequately supported by AHEAD.
- *Predefined parts* of the overall process are executed in *workflow management systems*. Existing process definitions can be reused (a-posteriori integration). This approach addresses the observation, that often in the beginning of a design process only part of it are understood well enough to support them using a workflow management system.
- Through a *view-based integration*, partial processes running in workflow management system can be represented in AHEAD as dynamic task net fragments within the overall design process. Thus the manager can monitor all parts of the process in AHEAD using only one adequate process representation regardless if or how they are executed by other management systems.
- In order to reduce the effort for the integration of multiple workflow management systems, we make use of the *neutral exchange format XDPL* from the Workflow Management Coalition [1059]. The transformation between

processes described in XPDL format to dynamic task nets does not need to preserve the full semantics of both formalisms, because this would lead to very rigid requirements for the systems to be integrated. Moreover, this is not necessary, since the workflow fragments are used in AHEAD for monitoring purposes only, and therefore it seems tolerable if some process information is lost during the generation of the workflow fragments.

There are several alternatives for the mapping of predefined partial workflow processes into dynamic task nets. On the one hand, the whole workflow process can be represented as a single task in the dynamic task net. Its activation reflects the start of the workflow instance within the workflow management system, while the termination of this task reflects the termination of the workflow instance. In this case, the fine-grained activity structure of the workflow process is hidden (*black-box approach*). This simple form of integration is sufficient in many situations, but because of the encapsulation it is impossible to monitor the progress of the activities in the workflow process within AHEAD.

On the other hand, all details of the workflow process can be mapped to a task net (*white-box approach*). This alternative suffers from the following disadvantages. First, this extreme form of transparency is often not feasible, if some details of the workflow should be hidden because of confidentiality reasons. Second, the language of dynamic task nets has to be capable of expressing all aspects and peculiarities of the modeling language used for the definition of the workflow process. Because the mapping is carried out to allow for the monitoring of these processes in AHEAD, we can afford to map only a filtered portion of all details of the workflow process, e.g. control structures. Third, both modeling languages are used on different levels and for different purposes. Workflow definition languages target at the automatic execution of the described workflows within the workflow management system. This requires describing a lot of necessary technical details on a very low abstraction level. In contrast, dynamic task nets describe processes with respect to the coordination of their tasks on a very high abstraction level.

Our mapping approach is in the middle of these two mapping alternatives. Only selected details of the workflow process, which are necessary to represent the coordination aspects of the activities in the workflow, are mapped into a dynamic task net (*gray-box approach*). For example, such workflow fragments do not contain workflow relevant process variables which are only needed internally by the workflow engine to automatically decide which activities to execute next upon the termination of workflow activity.

To illustrate our approach, we revisit our scenario on the design process of a plant for Polyamide-6 (PA6) carried out within a chemical company, which is used throughout this paper. There we can easily identify a static process fragment in the plastics engineering part of the process as a good example, namely the determination of the mixing quality within the extruder through a complex and expensive 3D-simulation (see Fig. 3.86). Because this process fragment is small, static and well-understood, it is feasible to model it as a

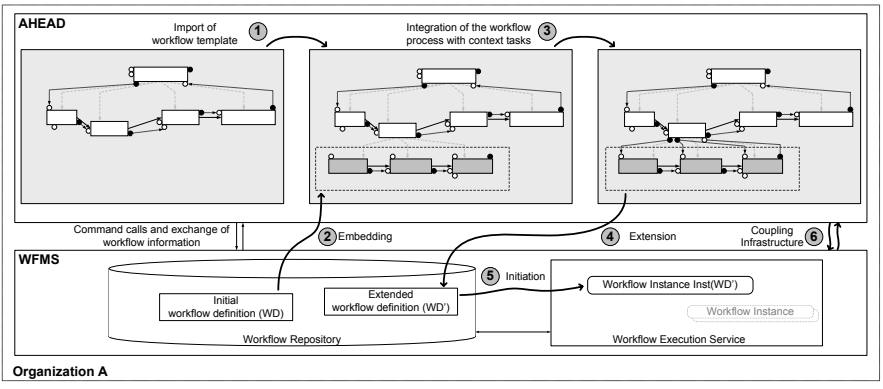

Fig. 3.94. Integration of workflow processes in AHEAD

workflow process and support its execution within a workflow management system.

The AHEAD system and the workflow management system SHARK from ENHYDRA [660] have been integrated with each other to support this scenario. The manager of the chemical company uses AHEAD to manage the overall design process, and a dedicated team of simulation experts is responsible for performing 3D-simulations. Because all 3D-simulations have to follow a best-of-breed practice, developed once within the company, a workflow process has been defined to enforce this simulation procedure. As illustrated in Fig. 3.94, we describe the different phases of workflow processing using an example session:

- *Workflow Embedding.* The manager decides within the compounding part of the process that a 3D-simulation is needed in order to analyze quality-affecting problems of the current parameterization of the extrusion process. He opens a browser displaying all available workflow processes, chooses the 3D-simulation workflow and imports its dynamic task net representation (called *workflow fragment* or *workflow template* below) into the compounding subnet (1). Then the workflow fragment is embedded within the subnet of the compounding task where other compounding subtasks also reside (alternatively it is possible to embed a new single task first and embed the workflow fragment as its subnet) (2). All formal parameters of the workflow regarding the input and output to it are located at the tasks which represent workflow activities processing these parameters. These are the first and last tasks in the workflow fragment.
- *Workflow Context Definition.* The manager connects the isolated workflow fragment with other tasks by adding control flows and data flows to at least the first and last tasks of the workflow fragment (3). After that, he sets the execution state of the workflow tasks to Waiting and subse-

quently all workflow tasks are set to this state, too. An extended workflow definition with additional workflow process data for the workflow context is generated (4). After that, the context definition phase is finished within AHEAD.

- *Workflow Instantiation.* AHEAD automatically contacts the workflow management system and requests the creation of an instance of the corresponding workflow process definition. A new instance of the workflow definition is created and a reference to the instance is handed back to AHEAD (5). After all defined input data is transferred and provided as actual parameters to the workflow instance, the workflow is finally started.
- *Workflow Monitoring.* Workflow activities are assigned to members of the 3D-simulation team. They can accept and start assigned activities, read and update workflow relevant data and finally commit workflow activities. The workflow management system automatically routes the control and data flow to the next workflow activities. All process changes which are relevant for the monitoring within AHEAD, like status changes or document processing, are forwarded to AHEAD via an event-based coupling infrastructure (6). The manager can thus monitor the progress of the workflow process within AHEAD.
- *Process Traceability.* After the termination of the last workflow activity, the workflow instance is terminated in the workflow management system automatically. In AHEAD, the workflow fragment is still visible with all terminated workflow tasks of this fragment. All documents produced during the course of the workflow remain accessible in AHEAD. This allows for traceability of the overall process regardless if a part has been executed in AHEAD or in a workflow management system.

With this approach workflow processes can be embedded within the overall dynamic process. The process manager can monitor and control the execution of workflow fragments within the AHEAD system. The coupling of workflow management systems and the AHEAD system is achieved by an event-based coupling infrastructure. Both systems generate events about relevant process changes and forward them to the coupled system via the coupling infrastructure [471].

Features of the View-Based Cooperation Model

The new view-based cooperation model can be *characterized* and summarized by the following *features*:

- *Dynamic process view model.* Managers can use dynamic process views to configure the access rights to selected parts of the private process by external organizations. Process views are highly flexible and support the provision of different perspectives onto a process fragment according to

individual cooperation needs. With the concept of process views, each process manager can manage autonomously which process parts shall remain private and which process parts shall be published to other organizations.

- *Process view evolution on definition-level and instance-level.* Planning and enactment of dynamic task nets may be interleaved seamlessly so that the private processes are constantly changed (process evolution). Consequently, our process view concept allows for the dynamic evolution of the process views as well. Process view instances are always kept consistent with their underlying process by incrementally updating the view contents according to the process view definition upon process changes.
- *Uniform modeling of processes and process inter-connection.* Intra- and interorganizational processes are uniformly modeled by re-using elements of dynamic task nets (e.g. tasks, parameters, control flows, data flows, and feedback flows). In this way, process managers do not need to use different modeling languages for the modeling of intra- and interorganizational process fragments.
- *Contract-based support for different cooperation scenarios.* The concept of process views allows to support different cooperation relationships between organizations. For instance, monitoring relationships, interaction relationships, or outsourcing relationships across organizations can be configured within the same process management system. Contracts can be established to fix all necessary agreements between the partners, like the different organizational roles with rights and duties, the involved process views, as well as different kinds of cooperation policies for changes of the contract or related process view definitions. Additional parameters can be stored in contracts as well (e.g. cost or time schedules).
- *Conformance monitoring and inconsistency toleration.* Another important feature of our approach (not presented here) is the monitoring and control of the inter-organizational cooperation. Upon modification, each process view is checked by the management system for conformance with the process meta-model of dynamic task nets. Detected violations of structural and behavioral constraints are reported to the process managers. They may either modify the process views in order to re-establish consistency or tolerate the violation.
- *Integration of workflow processes.* Workflow processes can be embedded into the overall dynamic task net in order to monitor and control their execution from within the AHEAD system. To achieve the desired integration, workflow processes are mapped to dynamic task nets and the resulting workflow fragments are subsequently integrated with the dynamic parts of the process. Therefore, all aspects of the design process within all of its static or dynamic parts are represented in a unique process modeling formalism. On the technical level, workflow management systems are coupled with the AHEAD-System using an event-based coupling infrastructure. At process runtime, both management systems exchange events to keep each other informed about relevant process changes.

System Architecture for Interorganizational Cooperation Support

Both the delegation-based and the view-based cooperation model are realized on the basis of an *event-based coupling mechanism* [208]. The *graph-based realization* of the coupling concept is described in Fig. 3.95. Two AHEAD systems are coupled together using a communication server.

Let us first concentrate on the AHEAD system on the left-hand side. Each AHEAD *system* consists of a graphical user *interface*, the AHEAD core (containing the application logic library and the UPGRADE framework) and the underlying graph database. The task net shown in the graphical user interface is created step by step by invoking special user interface *commands*, for example, to insert a new task or a new control flow relationship between two tasks. Each user interface command calls a graph *transaction* of the application logic in the AHEAD core. The execution of a graph transaction leads to the *manipulation* of the *graph data* stored in the graph database. In the example, at the graphical user interface a task T1 is displayed. Invoking a user interface command to activate task T1 leads to a *change* of one of the *attributes* of the corresponding graph node in the database. The database propagates all changes on the graph data back to the AHEAD core. According to these *change events* the current state of the graphical user interface is updated.

If one of the AHEAD systems is temporarily disconnected, the communication server stores the events for subsequent delivery. In the coupled system, corresponding graph *transactions* in the AHEAD core are called for each of these *change events*. Accordingly, the graph data stored in the graph database is manipulated and the graphical user interface is updated. Therefore, changes regarding the monitored task T1 are also displayed in the GUI on the right hand side. Every AHEAD system can at the same time act as a producer of change events regarding all elements which are monitored in coupled systems and as a consumer of change events regarding all elements which are executed elsewhere and only monitored locally.

The realization of the view-based cooperation model has required a number of *extensions* to this *coupling mechanism* with respect to the coupling of AHEAD systems and workflow management systems [129, 471]. Mainly, the application logic of AHEAD was substantively changed and extended in order to realize the new view-based concepts for process views, cooperation relationships, flexible configuration support, as well as the needed user interfaces for a view editor environment.

3.4.6 Related Work

AHEAD Core System

In the following, we will discuss the state of the art of *tool support* for managing design processes. From the previous discussion, we derive a set of crucial requirements for management tools for design processes:

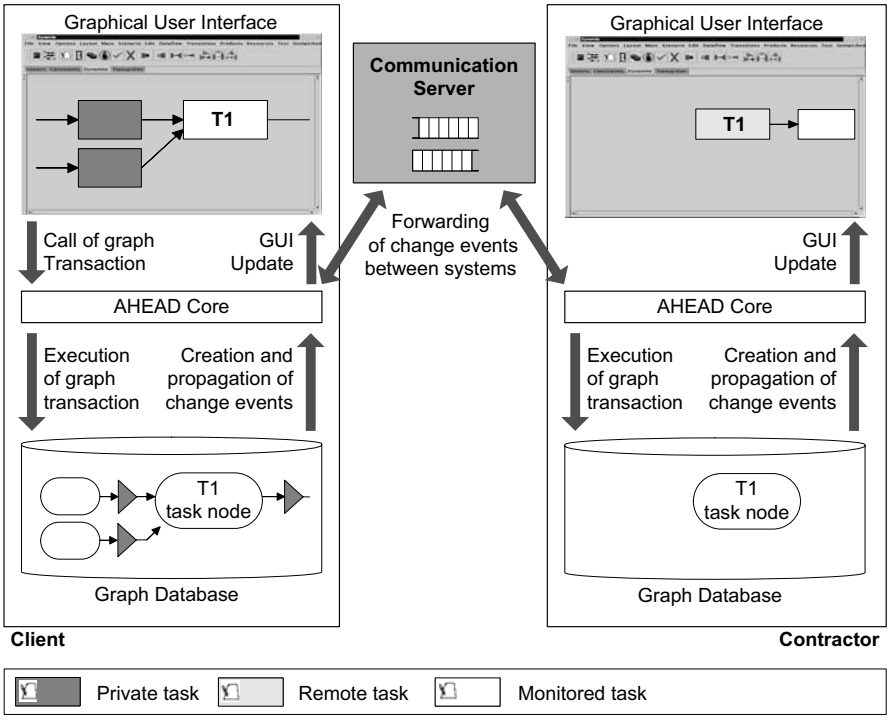

Fig. 3.95. Realization of the coupling of two AHEAD systems

- *Medium-grained representation.* The management of design processes has to be supported at an appropriate level of detail.
- *Coverage and integration at the managerial level.* Management tools have to deal equally with products, activities, resources and their relations.
- *Integration between managerial and technical level.* Managerial activities have to be coupled with technical activities: Designers have to be supplied with the documents they to be manipulated, as well as with the corresponding tools.
- *Dynamics of design processes.* Design processes evolve continuously during execution (product evolution, feedback, simultaneous/concurrent engineering).
- *Adaptability.* Management tools have to be adapted to a specific application domain and they must provide domain-specific operations to their users.

AHEAD meets all of these requirements. In industry, a variety of *commercial systems* is being used for the management of design processes, including systems for project management, workflow management, and product management, see below. All of these systems only partially *meet* the *requirements* stated above (Table 3.3):

Table 3.3. Comparison of AHEAD with commercial management systems

	AHEAD	**project management systems**	**workflow management systems**	**product management systems**
granularity of representation	medium-grained	coarse-grained	medium- and fine-grained	medium-grained
coverage at the managerial level	products, activities, resources	activities, resources	activities, resources	products, (activities)
integration with technical level	tool integration, document storage	not supported	tool integration	document storage
support for dynamic design processes	full support of process evolution	evolving project plans	limited (fixed workflows)	version control for documents
adaptability	UML models	not supported	workflow definitions	database schema

Project management systems [777] such as e.g. Microsoft Project support management functions such as planning, organizing, monitoring, and controlling. The project plan acts as the central document which may be represented in different ways, e.g., as a PERT or GANTT chart. It defines the milestones to be accomplished and provides the foundation for scheduling of resource utilization as well as for cost estimation and control. Project management systems are widely used in practice, but they still suffer from several limitations: project plans are often too coarse-grained, products (documents) are not considered, project plans are not integrated with the actual work performed by engineers, and there is no way to define domain-specific types of project plans.

Workflow management systems [763, 803], e.g., Staffware, FlowMark, or COSA, have been applied in banks, insurance companies, administrations, etc. A workflow management system manages the flow of work between participants, according to a defined procedure consisting of a number of tasks [836]. It coordinates user and system participants to achieve defined objectives by set deadlines. To this end, tasks and documents are passed from participant to participant in a correct order. Moreover, a workflow management system may offer an interface to invoke a tool on a document either interactively or automatically. Their most important restriction is limited support for the dynamics of design processes. Many workflow management systems assume a statically defined workflow that cannot be changed during execution. This way, dynamic design processes can be supported only to a limited extent (i.e., the statically known fractions can be handled by the workflow management

system). Recently, this problem has been addressed in a few university prototypes (see e.g. [628, 688]).

In the context of this paper, the term *product management system* refers to all kinds of systems for storing, manipulating, and retrieving the results of design processes. Depending on the context in which they are employed, they are called engineering data management systems (EDM), product data management systems (PDM [722]), software configuration management systems (SCM [1000, 1049]), or document management systems. Documentum and Matrix One are examples of such systems which are used in chemical engineering. Documents such as flowsheet, steady-state and dynamic simulation models, cost estimations, etc. are stored in a database which records the evolution of documents (i.e., their versions) and aggregates them into configurations. In addition, product management systems may offer simple support for the management of activities (e.g., change request processes based on finite state machines), or they may include workflow components, which suffer from the restrictions already discussed above. Their primary focus still lies on the management of products; in particular, management of human resources is hardly considered.

The approaches cited above do not depend on a certain application domain. For instance, workflow management systems can be applied to business processes in different disciplines, and product management systems can be used in different engineering disciplines. Only a few approaches target the domain of chemical engineering directly. For example, KBDS [524] allows to manage different design alternative together with the change history; n-dim [1047] supports distributed and collaborative computer-aided process engineering. But these approaches do not really support the integrated management of processes, products, and resources. Moreover, they are restricted to their single application domain and cannot be used in different domains.

Process Evolution and Parametrization

The need for a *wide spectrum approach* to process management was recognized as a research challenge in [963]. It is explicitly addressed in GroupProcess [742], 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 [727] and FLOW.NET [773] 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 [712] may be used to deal with errors and special cases. However, the main focus still lies on highly or moderately 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*. [616] and [861] both deal with in-

consistencies between process definitions and process instances. In PROSYT [616], 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 [727, 764, 772, 792], 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 [598, 1044]. 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 [772, 792] and schema versioning [584]. 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 [584]. We believe that the underlying base mechanisms must be as flexible as possible. For example, in [727, 772, 792], 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., [526, 929]). 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.

Interorganizational Coordination

A lot of workflow management systems deal with *distributed processes*. However, a distributed process need not be interorganizational as addressed in this paper. The term "interorganizational" refers to cooperation between different enterprises, while the term "distributed" can be used to describe processes where tasks are distributed either within a single enterprise or across enterprises. For instance, the workflow management system *Mentor* [1055] supports distributed processes by providing multiple workflow servers. In this approach, work is distributed within a single enterprise among workflow servers, according to a sophisticated load balancing algorithm.

[1012] provides an overview of paradigms for *interorganizational processes*. Among others, the following paradigms are identified:

- *Process chaining.* From some process p, a process q is launched to continue the overall process. The only interaction between p and q occurs when q is started. Subsequently, p and q perform independently of each other.
- *Subcontracting.* A task t of the overall workflow is passed to a subcontractor, which executes t and passes the results back to the contractor. From the perspective of the contractor, t appears to be atomic. The contractor and the subcontractors interact both at the start and at the end of the execution of the subcontracted process.
- *Loosely coupled processes.* Processes are executed in parallel in different organizations. Occasionally, they interact at pre-defined communication and synchronization points.
- *Case transfer.* The workflow is seen as a case which has to be transferred among different organizations. Transferring the case includes transfer of documents and transfer of the current state of execution. Only one organization at a time may execute the case.

Some of these *aspects* are investigated in *literature*: The work of [1012] primarily focuses on case transfer and an extended variant thereof. In [1013], the same author discusses loosely coupled processes. The interaction paradigms process chaining and subcontracting are supported by the standards defined by the Workflow Management Coalition (WfMC [803]). In addition, subcontracting was introduced as early as 1987 by the Istar system [641] into the software engineering domain.

The *delegation model* of the AHEAD system adds a new paradigm to the classification scheme presented above. It differs from process chaining inasmuch as the contractor and the subcontractors do interact while the delegated subprocess is being executed. The delegation-based approach also differs from the case transfer model because both parties perform their parts of the overall process in parallel: The contractor is not suspended when a subprocess is delegated to a subcontractor. Delegation constitutes a significant extension of subcontracting because subprocesses rather than single tasks may be delegated in general. Like loosely coupled processes, contractor and subcontractor may interact during the execution of the delegated subprocess rather than merely at the start and the end, respectively. Delegation differs from loosely coupled processes because there is a hierarchical relationship between contractor and subcontractor (while loosely coupled processes are peer to peer in general). Finally, the delegation-based approach supports dynamic changes, while loosely coupled processes have been introduced for statically defined workflows.

Besides this work, we have extended AHEAD to provide additional support for the paradigm of *loosely coupled* process integration mentioned above and we introduced a new cooperation layer above the execution-oriented process view and private process layers. In the following, we restrict ourselves to

highlighting related work addressing similar view-based approaches to support interorganizational processes.

Some other researchers like Finkelstein [669] use the concept of a view in different way than we do. These approaches focus on the consistent integration of these views in order to maintain a consistent and up-to-date representation of the whole development process by superimposition of all views. While these approaches focus on the problems of view-based process definition that arise with modifiable views, we use views which usually are not modified by anyone else than the view publisher, so we do not face problems of consistent integration to that extent. Because we do not use different modeling formalisms for all process views (we always use dynamic task nets in all process views), we do not face the problem that two views onto the same process part model different aspects of it in a conflicting way.

In our application domain of development processes in chemical engineering, we put more focus on the processes at the instance level rather than on the definition level when interorganizational cooperation is concerned. Using a view-based approach to process coupling, the views published for a process instance and the process instance can easily become inconsistent upon modifications because the processes evolve with the time. Process views are directly embedded into the private processes of other organizations (no integration process are used), where remote elements and local elements are connected with control flows, feedback flows, or data flows.

Several approaches target the modeling of the integration aspects between separate processes. To model the interconnection of existing workflow processes, the used workflow modeling language can be extended with additional modeling elements. For example, new modeling elements can be introduced to express the publication and interception of events which are exchanged between workflows processes of different organizations (like the approach described by Casati and Discenza [585]). Alternatively, explicit *synchronization points* can be modeled, as proposed by Perrin et al. [905]. In this case, the existing workflow modeling language is not extended and a separate modeling language is introduced. This approach allows to replace one of the two used modeling languages by another modeling language without affecting the other modeling language.

Van der Aalst [1011] focuses on independently running but loosely coupled interorganizational workflow processes, modeled in a language based on Petri-Nets. This approach is based on a predefined communication structure between the private partner processes which cannot be changed during runtime. Another approach is to split a workflow into several workflow fragments which are executed by the cooperation partners afterwards. Here, definition-time and run-time are strictly separated. In these two approaches, a top-down approach is used which is feasible if the overall process structure is known in advance. In our application domain of dynamic development processes, this is not feasible, since development processes cannot be planned fully in advance. New integration points between already existing partial processes of the part-

ners should be creatable and modifiable whenever needed. So, a mixed top-down and bottom-up approach is more feasible. But at the same time it is important to ensure that all partial processes can be managed autonomously by the process managers of the cooperating organizations.

Three important view-based approaches of interorganizational workflows have been proposed by Liu and Shen [821], Chiu et al. [595], and Tata, Chebbi et al. [590, 993]. All three concepts provide support for routine business processes and they separate definition-time from run-time. Workflow definitions can be re-used as view definitions to model the public workflow parts which are accessible by other organizations. These view definitions cannot be changed after the overall workflow has been started. The workflow definitions usually do not need to be modified frequently, because the modeled business processes are not changed too often. For example, Liu and Shen use additional process definitions ("integration processes"), which contain the coupling of private workflow definitions and foreign view workflow definitions. This eases rapid composition of business processes from pre-existing processes as further goal of these approaches. In contrast, in our view-based approach the process views represent processes at the instance-level (not on the definition-level). Process views are directly embedded into private processes of other organizations (no integration processes are necessary). Furthermore, the other mentioned approaches do not focus on the interleaved definition and execution of process and views.

The view-based cooperation model in AHEAD also has related work in the research field of communication-oriented interorganizational cooperation. For example, Weigand and de Moor [1040] work on workflow modeling that considers both *customer relations* and *agency relations* to chart complex organizational communication situations. Here, "agency" means that a relation between a principal and some agents exists where both roles have different rights and duties. An agent acts for the benefit of someone, the beneficiary, and at the same time conducts an operation on behalf of someone else, the principal. The authors propose a modeling method with the following steps: (1) the process is defined and all process tasks can be decomposed into subtasks, (2) selected tasks can be delegated to intra-organizational resources for execution (introducing new *agency relations*), and (3) selected tasks can be outsourced to other organizations (introducing new *customer relations*). The authors present an extended workflow loop model to separate between the *workflow execution task* and the *control task*. This extended model is used for modeling both the agency and customer relations. In AHEAD, we deal with all three mentioned aspects of decomposition and composition of processes as well as intra-organizational and interorganizational cooperation relationships. Our new cooperation layer introduces three different cooperation relationships (monitoring, interaction, and outsourcing) as well as contracts for defining the formal guidelines structuring the cooperation.

3.4.7 Conclusion

In this section, we argued that design processes in chemical engineering are hard to support because they are highly creative, many design alternatives are explored, and both unexpected and planned feedback occurs frequently. These difficulties are taken into account by the *reactive* management system AHEAD which has been developed as the main contribution of the subproject B4 of IMPROVE. AHEAD addresses the management (or *coordination*) of complex and *dynamic design processes* in chemical engineering and supports the planning, execution and control of design processes, which continuously evolve during process execution. Design processes, e.g. for the design of a chemical plant, are represented as process model instances and process model definitions for the description of classes of design processes are created in order to adapt AHEAD to different application domains.

The system has a number of outstanding features which contrasts it from competing process management systems: First, AHEAD supports seamless *interleaving* of planning and execution which is a crucial requirement which traditional workflow management systems usually do not meet. Second, AHEAD *integrates* products, activities, and resources, and their mutual relationships on a medium-grained level. Third, process *evolution* is supported with respect to both process model definitions and process model instances; changes may be propagated from definitions to instances and vice versa (round-trip process evolution). Fourth, in addition to local processes, *interorganizational* design processes are addressed by providing flexible and configurable cooperation support. These contributions on the conceptual level have been demonstrated by several research prototypes. Summing up, the AHEAD system in its current state is the result of one habilitation project and four dissertation projects carried out by the members of the subproject B4.

Another important aspect of reactive management of design processes is the *incorporation* of process *knowledge* contained within *application* models which are developed by our partners in IMPROVE. We have not covered this topic explicitly here, because it is addressed in more detail in Sect. 6.4.

We have applied the AHEAD system successfully to the reference *scenario* studied in the IMPROVE project, which was elaborated in cooperation with industrial partners. But AHEAD is a research prototype which cannot be applied immediately in industry (i.e., in a production environment) for various reasons. In addition to deficiencies with respect to stability, efficiency, and documentation – problems which are faced by many research prototypes –, an important prerequisite of industrial use constitutes the integration with other management tools which are used in *industry*. Therefore, we have integrated AHEAD with several commercial systems for workflow, document, and project management in order to prepare the *technology transfer* into industrial practice as the ultimate goal of the research activities carried out within the subproject B4 of IMPROVE (see Sect. 7.7). Since we are convinced that the developed concepts and mechanisms in the AHEAD system can contribute

significantly to the state-of-the-art of commercial process support tools, we will investigate in the future how dynamic processes can best be supported on the basis of existing management systems. Together with our partners from industry, this research is planned to be carried out within the transfer project.