AHEAD: A Graph-Based System for Modeling and Managing Development Processes*

Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel

Aachen University of Technology
Department of Computer Science III
D-52056 Aachen, Germany
{jaeger,schleich,bernhard}@i3.informatik.rwth-aachen.de

Abstract. Management of development processes in different engineering disciplines is a challenging task. The AHEAD system addresses these challenges by providing an integrated environment for modeling and managing development processes. Products, activities, and resources are managed in an integrated way; furthermore, AHEAD supports evolving development processes by seamless interleaving of planning and execution. AHEAD is based on programmed graph transformations; tools are generated from a graph-based specification. Finally, a wide-spread object-oriented modeling language (UML) is employed for acquiring process knowledge from domain experts.

1 Introduction

Development of products in disciplines such as mechanical, chemical, or software engineering is a challenging task. Costs have to be reduced, the time-to-market has to be shortened, and quality has to be improved. Skilled engineers and sophisticated tools for performing technical work are necessary, yet not sufficient prerequisites for meeting these ambitious goals. In addition, the work of engineers must be coordinated so that they cooperate smoothly. To this end, the steps of the development process have to be planned, an engineer executing a task must be provided with documents and tools, the results of development activities have to be fed back to management which has to adjust the plan accordingly, the documents produced in different working areas have to be kept consistent with each other, etc.

The *AHEAD* system (*A*daptable and *H*uman-Centered *E*nvironment for the *A*dministration of *D*evelopment Processes) addresses these challenges. AHEAD supports the management and modeling of development processes. Its key features are the following ones:

* The work described in this paper was partially supported by the Deutsche Forschungsgemeinschaft (Sonderforschungsbereich 476 "IMPROVE" and Graduiertenkolleg "Informatik und Technik").

M. Nagl, A. Schürr, and M. Münch (Eds.): AGTIVE'99, LNCS 1779, pp. 325–339, 2000.

1. Products (documents such as requirements definitions, software architectures, or module implementations), activities (tasks such as design, implementation, and test), and resources (developers assigned to tasks and tools supporting developers) are managed in an integrated way.
2. AHEAD takes care of the dynamics of development processes. Due to numerous factors (e.g., product evolution, feedback, concurrent or simultaneous engineering), development processes cannot be completely planned a priori; rather, they constantly evolve during execution. Thus, planning and execution have to be interleaved seamlessly.
3. AHEAD is human-centered in that it supports both managers and developers through interactive tools. It does not attempt to automate management, as it is done e.g. in many workflow management systems.
4. AHEAD is based on graph transformations. Task nets, version histories, product configurations, etc. may be modeled as graphs in a natural way. Operations on these graphs are declaratively specified by graph rewrite rules. In this way, we obtain a specification of the underlying management model at a high level of abstraction.
5. Tools are generated from graph-based specifications rather than encoded by hand. This does not only reduce implementation effort. In addition, proofs of the correctness of the implementation with respect to the specification are no longer necessary.
6. AHEAD is adaptable to different application domains. To acquire process knowledge from domain experts, we use a wide-spread object-oriented modeling language (UML) rather than graph rewriting systems. Object-oriented process models are automatically translated into graph transformations. Thus, the translation formally defines the semantics of process models in UML.

AHEAD is a successor of a management system which was developed in the SUKITS project that dealt with development processes in mechanical engineering. AHEAD is currently being developed within IMPROVE, a Collaborative Research Council that investigates methods and tools for development processes in chemical engineering (see [19] for a description of both projects). IMPROVE is a research project carried out by engineers and computer scientists at Aachen University of Technology. There are strong contacts to industrial partners, in particular Bayer AG. The AHEAD system will be evaluated both internally by the engineering partners and externally by the industrial partners.

Within the IMPROVE project, AHEAD is being applied to a complex process which deals with the development of a chemical plant for the production of Polyamid-6. In this paper, however, we will stick to a simple example from the software engineering domain in order to explain the underlying concepts in an easily understandable way.

The rest of this paper is structured as follows. Section 3 provides an overview of the AHEAD system. Section 2 describes the underlying model for managing products, activities, and resources. Section 4 presents the environments offered

by the AHEAD system for supporting process modelers, managers, and developers. Section 5 discusses related work. Section 6 concludes the paper.

2 Background

The AHEAD system is based on an integrated model for managing products, activities, and resources. The respective submodels are briefly described below (see [14,24] for more detailed descriptions).

CoMa (*Co*nfiguration *Ma*nagement, [23]) supports version control, configuration control, and consistency control for heterogeneous documents through an integrated model based on a small number of concepts. In the course of development, documents such as designs, manufacturing plans, or NC programs are created with the help of heterogeneous tools. Documents are related by manifold dependencies, both within one working area and across different working areas. The representation of these dependencies lays the foundation for consistency control between interdependent documents. Documents and their mutual dependencies are aggregated into configurations. Since development processes may span long periods of time, both documents and configurations evolve into multiple versions. Versions are recorded for various reasons, including reuse, backup, and coordination of team work. Consistency control takes versioning into account, i.e., it is precisely recorded which versions of different documents are consistent with each other.

DYNAMITE (*DYNAMI*c *T*ask N*E*ts, [9]) supports dynamic development processes through evolving task nets. Editing, analysis, and execution of task nets may be interleaved seamlessly. A task is an entity which describes work to be done. The interface of a task specifies what to do (in terms of inputs, outputs, pre- and postconditions, etc.). The realization of a task describes how to perform the work. A suitable realization may be determined only at run time, using one of multiple alternative realization types. A realization is either atomic or complex. In the latter case, there is a refining subnet (task hierarchies). In addition to decomposition relationships, tasks are connected by control flows (which are akin to precedence relationships in PERT charts), data flows, and feedback flows.

RESMOD (*RES*ource Management *MOD*el, [15]) is concerned with the *resources* required for executing development processes. This includes both human and computer resources, which are modeled in a uniform way. Complex resources are represented by resource configurations, whose components are connected by dependencies. Prior to the execution of some project, resource requirements may be planned. To this end, RESMOD provides (abstract) plan resources to which actual (concrete) resources may be assigned later on. Finally, RESMOD supports multi-project management, which, for example, includes global balancing of workloads. To this end, the RESMOD model introduces project resources which are allocated from an enterprise-wide pool of base resources.

An example illustrating these submodels and their integration is given in Figure 1. The figure refers to a sample process from software maintenance that

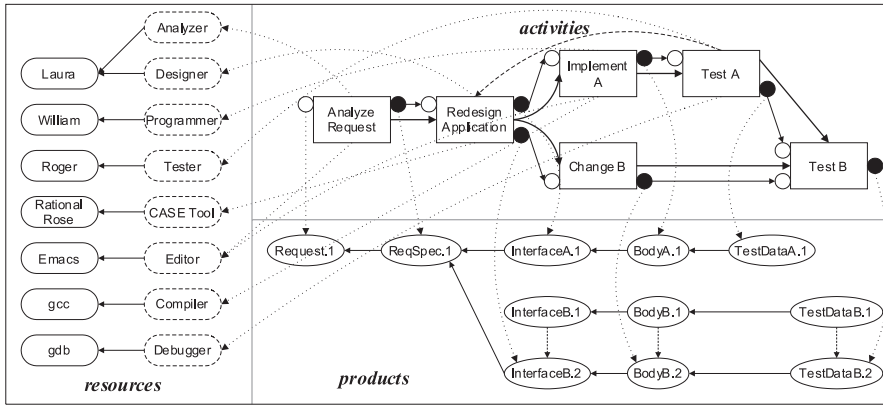

Fig. 1. Integrated management of products, activities, and resources

is used as a running example. In response to a change request for extending the functionality of a software system, the system is redesigned, affected and new modules are changed and implemented anew, respectively, and all modifications are tested in bottom-up order. The task net consists of tasks connected by control flows and feedback flows; moreover, task parameters are connected by data flows. Parameters refer to versions of documents; the evolution history of each document is represented by a version graph. Resources are classified into plan resources and actual resources. Developers are assigned to tasks occurring in the task net; they are assisted by tools such as a CASE tool, a text editor, a compiler, and a debugger.

3 Overview of the AHEAD System

An overview of the AHEAD system is given in Figure 2. The figure is structured into four regions[1]. The horizontal line separates the *definition level* from the *instance level*; the vertical line is used to distinguish between *external* and *internal representations*. Furthermore, there are four environments supporting different kinds of users: the modeling environment for domain experts, the PROGRES environment for specification experts, the management environment for project managers, and the work environment for developers.

At the instance level, the AHEAD system offers tools supporting the management and execution of development processes. The *management environment* supports project managers in planning, analyzing, monitoring, and controlling development projects; the *work environment* assists developers in executing the tasks assigned to them by project managers. Both environments offer external views on the underlying management database. For example, PERT-chart

[1] The figure illustrates only the modeling and management of activities. However, AHEAD covers products and resources as well.

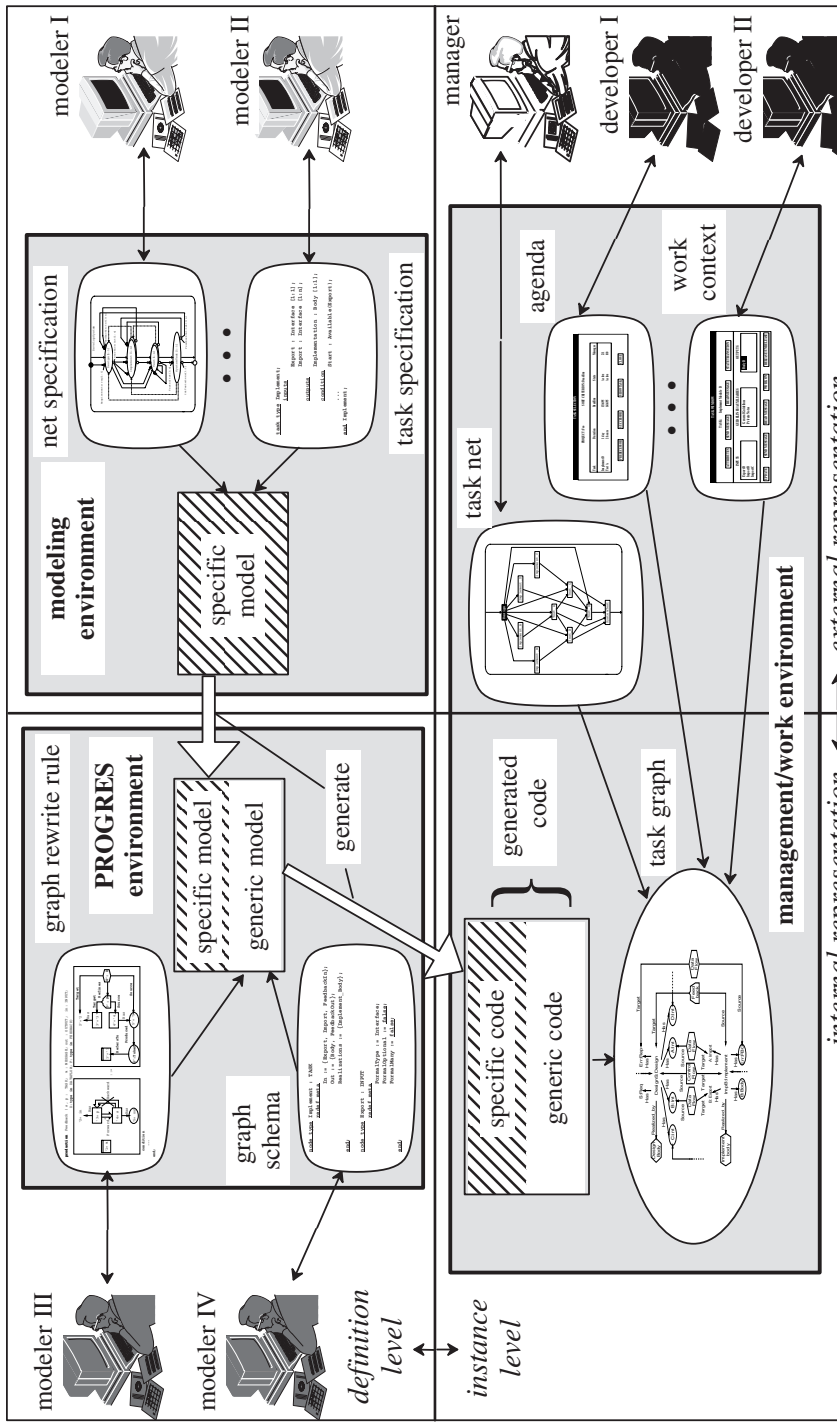

Fig. 2. AHEAD system

views on task nets are presented to project managers; developers are supplied with task agendas and work contexts providing documents and tools. The management environment and the work environment jointly constitute the *process support environment* [8].

Internally, the management data are represented by complex graph structures such as version graphs, configuration graphs, task graphs, and resource graphs. These graphs and the operations provided on them are defined by programmed graph rewriting systems. At the definition level, specifications are edited, analyzed, and interpreted in the *PROGRES environment* [21]. Subsequently, code is generated to obtain management tools operating at the instance level. Each specification is composed of a generic part (meta model), which can be reused for all (or at least a large class of) development processes, and a specific part (model definition), which incorporates domain-specific knowledge. The generic model is modified rarely; a specific model must be supplied for each application domain.

A specific model has to be developed in close cooperation with domain experts. AHEAD addresses different engineering disciplines such as mechanical, chemical, electrical, or software engineering (the latter of which is used for the examples presented in this paper). Clearly, we cannot assume domain experts to be familiar with PROGRES. Instead, we are employing a wide-spread object-oriented modeling language — the *Unified Modeling Language* (*UML* [2]) — for the communication with domain experts [11]. Process models described in UML are automatically transformed into corresponding domain-specific parts of PROGRES specifications [20]. Thus, domain experts are completely shielded from PROGRES. Note, however, that the generic models for product, activity, and resource management must have been specified in PROGRES beforehand: the PROGRES code generated by the UML transformation tool is based on the pre-defined specification of the generic part.

4 Environments

After having provided an overview of the AHEAD system, we discuss the environments which it offers in a more detailed way below.

4.1 PROGRES Environment

The management models introduced in Section 2 are fairly complex. To describe them formally, we use *attributed graphs* and *graph rewriting systems*. Complex management configurations may be represented by graphs in a very natural way. Using graph rewriting, we may specify all changes to these graphs in a uniform way. This includes creation of new versions of documents, construction of product configurations, planning and execution of dynamic task nets, assignment of resources, etc.

More specifically, we are using the specification language *PROGRES* [21] to formalize the management model. PROGRES combines concepts from database

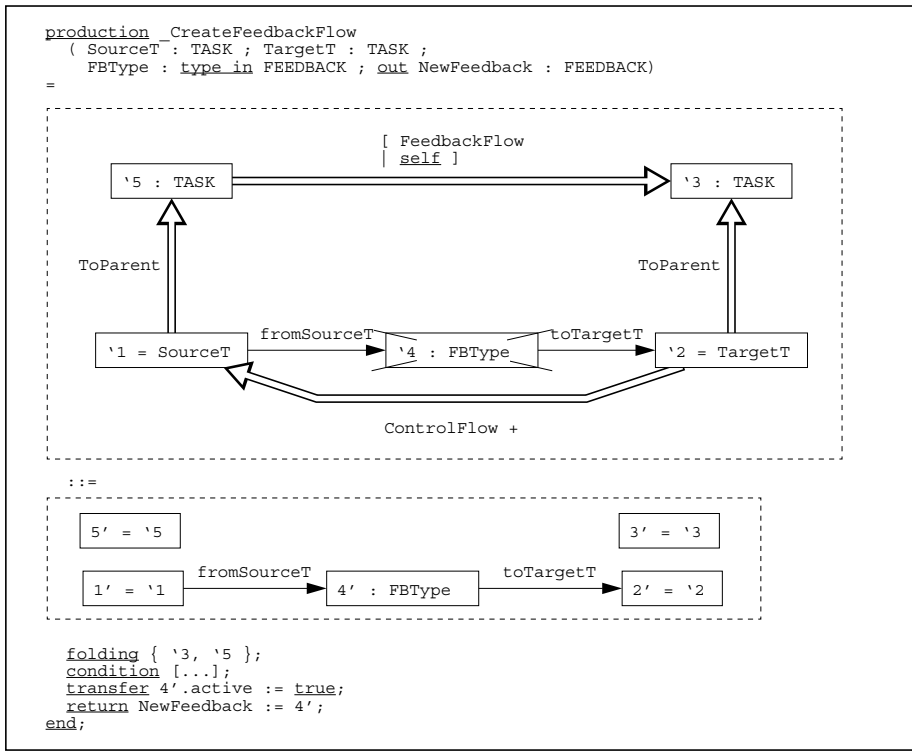

Fig. 3. Graph rewrite rule for creating a feedback flow

systems, knowledge-based systems, graph rewriting, and procedural programming into a coherent language.

A graph schema defines the types of nodes, edges, and attributes in a similar way as a database schema. Derived attributes and relationships (paths) can be defined in a graph schema as well. Graph transformations are specified by high-level graph rewrite rules operating on the level of graph patterns rather than on the level of single nodes and edges. Rewrite rules can be combined by control structures to form more complex transformations. Thus, the procedural programming style is supported as well.

The PROGRES specifications for CoMa, DYNAMITE, and RESMOD are both large and complex. In total, the process meta model covers about 200 pages of PROGRES; model definitions for specific applications may even exceed this size. For more detailed information on the specifications of CoMa, DYNAMITE, and RESMOD, the reader is referred to [23], [9], and [15], respectively.

Due to space restrictions, we merely present a single example for illustrating the application of PROGRES to process modeling. The example, which is taken from the DYNAMITE model, describes the insertion of a feedback flow into a task net (Figure 3). A *graph rewrite rule* is used to specify this transformation.

The rule is supplied with input parameters fixing the source, the target, and the type of the feedback flow to be created, and returns the new feedback flow as output parameter. The left-hand side (shown above the right-hand side) describes a graph pattern to be searched in the task graph. Nodes `‘1` and `‘2`, which are fixed by the corresponding input parameters, are connected by a feedback flow on the right-hand side. The other parts of the left-hand side define application conditions. First, source and target must not yet be connected by a feedback flow (negative application condition represented by the crossed node `‘4`). Furthermore, the source must be reachable from the target by a control flow path (double arrow from `‘2` to `‘1`) because feedback flows must be oriented oppositely to control flows. Finally, the parents of source and target must either be siblings, or they must coincide (`folding` clause below the right-hand side). In case all application conditions are satisfied, the flow is created and marked as active (`transfer` part for assigning attribute values).

The *PROGRES environment* offers tightly integrated, syntax-aided tools for editing, analyzing, browsing, and interpreting specifications (see [21] for details). Within the AHEAD system, these tools are used to develop the process meta model. In contrast, model definitions are created automatically (see below).

4.2 Modeling Environment

Encoding process knowledge into a PROGRES specification is a task that probably cannot be mastered by a domain expert. To gather knowledge from a domain expert, we need a modeling language that is easier to understand and is much more wide-spread than PROGRES. In addition, it should allow for expressing knowledge at an informal level.

We have selected the *Unified Modeling Language* (*UML* [2]) to satisfy these requirements. UML is a language that serves as a standard notation for object-oriented modeling. It offers a comprehensive set of diagrams for object-oriented modeling, including e.g. use case diagrams for documenting typical scenarios of using the system to be constructed, class diagrams for structural modeling, state diagrams for behavioral modeling, collaboration diagrams for describing the interactions among objects, etc.

For the purpose of process modeling, we have adapted and restricted UML according to our requirements [11]. For example, we have tailored class diagrams such that they may be used to define task nets at the type level. Moreover, we are using only a subset of the diagrams offered by UML. So far, we have mainly focused on *class diagrams* for structural modeling, as well as *state diagrams* and *collaboration diagrams* for behavioral modeling.

An example of a collaboration diagram is shown in Figure 4. The collaboration diagram describes an *event handler* that reacts on the creation of a feedback flow. While the graph rewrite rule of Figure 3 is part of the generic process meta model, the event handler shown here is specific to the change request process introduced earlier. The event handler assumes that a feedback flow has been created from the test of some newly implemented module to the redesign task.

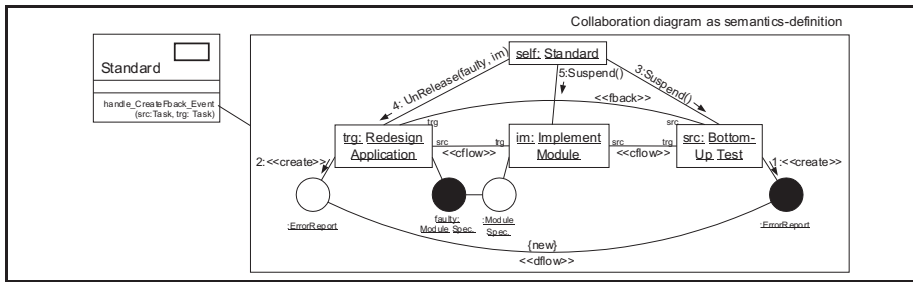

Fig. 4. Collaboration diagram for feedback handling

A collaboration diagram consists of objects and links that are required to be available when the method is executed. Additionally, objects and links can be created and destroyed during the execution of the specified method. The communication between objects can be defined through messages which may refine links. In the figure, messages `1` and `2` are used to create task parameters for an error report which are connected by a data flow; messages `3`–`5` are used to suspend tasks affected by the feedback and to unrelease the faulty design.

The formal semantics of UML diagrams is defined by an automatic transformation from UML to PROGRES. The details of this transformation are provided in a companion paper [20]. Figure 5 shows the PROGRES code generated for the collaboration diagram of Figure 4. Firstly, a *graph test* is performed. Subsequently, graph transformations are sequentially executed in a *transaction*. These graph transformations correspond to the method calls occurring in Figure 4.

For process modeling in UML, we use a commercial CASE tool (Rational Rose). The transformation tool traverses the UML diagrams and generates a text file containing PROGRES code. This text file is then parsed by the PROGRES environment.

4.3 Process Support Environment

The process support environment operates on a *management graph* which is composed of a product graph, a task graph, and a resource graph. The code for operating on the management graph is generated from the PROGRES specification (the PROGRES compiler generates C code). Thus, the domain-specific process model defined in UML is transformed in two steps into program code driving the process support environment.

The PROGRES compiler only takes care of the internal data manipulated by the process support environment; it is not concerned with the user interface. To generate tools from graph-based specifications, we are using the *UPGRADE* framework (*U*niversal *P*latform for *GR*aph-Based *A*pplication *DE*velopment [10]), which is currently under development. UPGRADE is implemented in Java, based on standard libraries and both public-domain and commercial components (ILOG JViews). It mainly focuses on graphical tools,

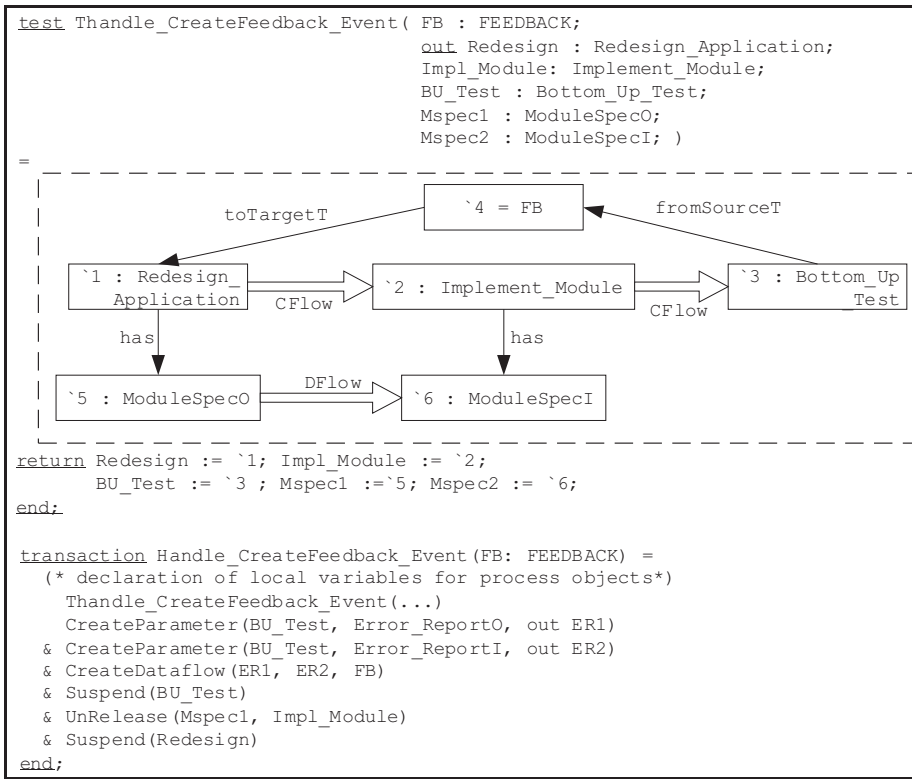

Fig. 5. Translation of the collaboration diagram

but it also supports e.g. tabular and tree representations. Graphical tools provide external views on the underlying management graph, hiding all of the technical details of the internal representation. Graph transformations defined in the PROGRES specification are offered as user commands. A constraint-based incremental layout algorithm automatically positions nodes and edges after the application of a graph transformation. The user may still improve the generated layout manually.

Using the UPGRADE framework, the process support environment is built with minimal effort. For example, command menus and windows for entering command parameters are generated from the operations defined in the PROGRES specification. External views on the management graph are defined by filtering nodes and edges based on type information. Furthermore, views may be based on derived data (derived attributes and relationships) that may already be defined at the PROGRES level. As a consequence, only small parts of the process support environment have to be coded manually.

Figure 6 shows a screen shot taken from the management environment. The project manager is supplied with a graphical view on a task net for our sample

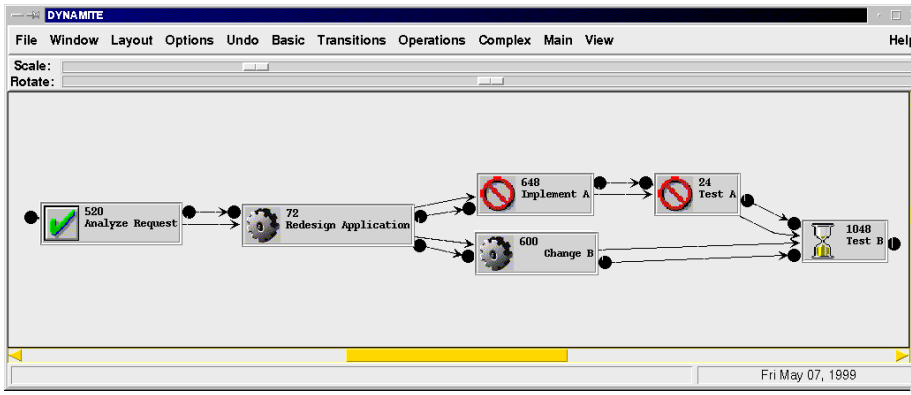

Fig. 6. Management environment

extension request process. Using such graphical views, the manager may analyze the current project state, assign tasks to developers, handle feedback, update the plan according to changes in the product structure, etc.

The work environment (not shown in a figure) provides developers with a tabular agenda of tasks to be done. From the agenda, a developer selects a task to work on. Then, the task's workspace is displayed, consisting of all documents relevant for this task. External development tools such as CASE tools, editors, compilers, etc. may be started on these documents. To this end, the work environment offers commands for tool activation. An external tool is started by means of a wrapper which supplies the document(s) to work on, prepares the operating system environment, and invokes the tool with appropriate parameters. In this way, developers are shielded from the details of tool activation.

4.4 Summary of the Tool Construction Process

Using AHEAD, a domain-specific management system is constructed in the following steps:

1. The modeling environment is used to define the domain-specific process model in terms of UML diagrams.
2. The transformation tool generates PROGRES code from the domain-specific UML model.
3. The domain-specific PROGRES code is compiled into C code with the help of the compiler being part of the PROGRES environment.
4. External development tools are integrated with the AHEAD system with the help of wrappers.
5. The UPGRADE framework is compiled with the generated C code and the tool wrappers, resulting in a domain-specific process support environment.

The construction of the domain-independent part of the AHEAD system — i.e., the "infrastructure" into which the domain-specific parts are implanted — involves the following steps:

1. The PROGRES specifications of the generic models for product, activity, and resource management are created with the help of editor, analysis, and interpreter tools provided by the PROGRES environment.
2. The user interface of the management system is developed with the help of the UPGRADE framework. This involves the definition of views and tools for both managers and developers.
3. The code compiled from the specification is combined with the user interface for the process support environment. As a result, we obtain a generic process support enviroment, being able to handle generic types of products, activities, and resources.
4. The modeling environment is implemented by adapting a commercial CASE tool (Rational Rose). The transformation tool accesses the database of the CASE tool and generates a text file containing PROGRES code.

Note that the generic process support environment can be employed meaningfully when there is no process knowledge available yet. In such a situation, the process support environment can be used in an "ad hoc" mode to gather experience that can then be turned into a domain-specific process model. This approach considerably reduces the start-up time for applying the process support environment.

5 Related Work

Numerous management systems have been designed and implemented to support the coordination of development processes. *Project management systems* [13,12] support management functions such as planning, organizing, monitoring, and controlling. *Engineering data management systems* (EDM [18]), *product data management systems* (PDM [7]), and *software configuration management systems* (SCM [22,25]) assist in managing the products of development processes in different engineering disciplines such as electrical, mechanical, and software engineering, respectively. *Workflow management systems* [16] have been applied in banks, insurance companies, administrations, etc. to manage the flow of work between participants according to a defined procedure consisting of a number of tasks [17]. Finally, *process-centered software engineering environments* (PSEE [4,5,3]) support the execution of software processes, driven by process models which define the activities to be executed, inputs and outputs, the resources required for execution, etc.

AHEAD differs from these systems with respect to all of its key features briefly described in the introduction:

1. Existing systems do not equally cover products, activities, and resources. Project management systems, workflow management systems, and PSEEs primarily focus on activities, while EDM, PDM, and SCM systems address product management.
2. The dynamics of development processes is not adequately taken into account. In particular, workflow management systems sharply distinguish between build time (definition of a workflow) and run time (workflow execution). Changes to workflows at run time are at best supported to a limited extent.
3. While AHEAD is human-centered and provides interactive management tools, PSEEs and workflow management systems tend to automate managers by replacing them with process programs. This approach does not work because of the inherent dynamics of development processes, which requires many human decisions during execution.
4. Most existing systems lack a formal definition of their underlying models for managing products, activities, and resources. Concerning the management of activities, however, there are several systems which are based e.g. on Petri nets, which do have formally defined semantics [1,6]. However, the semantics of Petri nets deals only with activities. In contrast, we employ graph transformations for specifying products, activities, and resources. Moreover, the evolution of Petri nets is described outside of the Petri net formalism, while the evolution of task nets in AHEAD is also formally described by graph transformations.
5. AHEAD is based on a framework for generating tools from graph-based specifications. Thus, the underlying process meta model (consisting of CoMa, DYNAMITE, and RESMOD) may be changed rather easily with modest effort. This does not apply to other systems with hard-coded process meta models.
6. For communicating with domain experts, we are using a wide-spread object-oriented modeling language. So far, our experiences concerning the expressiveness of UML have been positive. In contrast, numerous specialized process modeling languages in particular have been defined for PSEEs even though their underlying concepts are often similar. In addition, process modeling languages for PSEEs and workflow management systems tend to focus on process programming, while AHEAD also addresses the early phases of process engineering.

6 Conclusion

We have presented a graph-based system for managing and modeling development processes. Currently, the AHEAD system is still under development. We expect to complete the implementation in spring 2000. To give an impression of the development effort involved, let us present some numbers: The PROGRES system, which has been available for several years and is reused as an important component of the AHEAD system, comprises about 700,000 lines of code (loc). The specifications of the generic models for managing products, activities, and

resources cover about 200 pages of PROGRES code. The transformer from UML to PROGRES was implemented in 13,000 loc. Finally, the UPGRADE framework currently consists of about 40,000 loc, and the specific extensions required for the AHEAD system will require about 10,000 loc.

Before starting our work on the AHEAD system, we implemented a management system for development processes in the mechanical engineering domain within the SUKITS project [19,24]. This system relied on predecessor versions of the models for managing products, activities, and resources. The SUKITS management system was fully functional and was integrated with about a dozen tools for design, manufacturing planning, NC programming, etc. For evaluation and demonstration purposes, the system was applied to several development processes, including e.g. the development of a drill.

References

1. S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software process model evolution in the SPADE environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, Dec. 1993. 337
2. G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, Reading, Massachusetts, 1998. 330, 332
3. J.-C. Derniame, A. K. Baba, and D. Wastell, editors. *Software Process: Principles, Methodology, and Technology*. LNCS 1500. Springer-Verlag, Berlin, Germany, 1998. 336
4. A. Finkelstein, J. Kramer, and B. Nuseibeh, editors. *Software Process Modelling and Technology*. Advanced Software Development Series. Research Studies Press (John Wiley & Sons), Chichester, UK, 1994. 336
5. P. Garg and M. Jazayeri, editors. *Process-Centered Software Engineering Environments*. IEEE Computer Society Press, Los Alamitos, California, 1996. 336
6. V. Gruhn. Validation and verification of software process models. Technical Report 394/91, University of Dortmund, Dortmund, Germany, 1991. 337
7. S. B. Harris. Business strategy and the role of engineering product data management: A literature review and summary of the emerging research questions. *Proceedings of the Institution of Mechanical Engineers, Part B (Journal of Engineering Manufacture)*, 210(B3):207–220, 1996. 336
8. P. Heimann, C.-A. Krapp, and B. Westfechtel. An environment for managing software development processes. In *Proceedings of the 8th Conference on Software Engineering Environments*, pages 101–109, Cottbus, Germany, Apr. 1997. IEEE Computer Society Press. 330
9. P. Heimann, C.-A. Krapp, B. Westfechtel, and G. Joeris. Graph-based software process management. *International Journal of Software Engineering and Knowledge Engineering*, 7(4):431–455, Dec. 1997. 327, 331
10. D. Jäger. Generating tools from graph-based specifications. In J. Gray, editor, *Proceedings First International Symposium on Constructing Software Engineering Tools*, pages 97–107, Los Angeles, May 1999. University of South Australia, School of Computer Science. 333
11. D. Jäger, A. Schleicher, and B. Westfechtel. Using UML for software process modeling. In O. Nierstrasz and M. Lemoine, editors, *Software Engineering — ESEC/FSE '99*, LNCS 1687, pages 91–108, Toulouse, France, Sept. 1999. Springer-Verlag. 330, 332

12. V. Jungbluth. Alles im Griff: Projektmanagementsysteme im Vergleich. *c't*, (7):178–189, July 1997. 336
13. V. Jungbluth. Teamwork: Einführung in die EDV-gestützte Projektplanung. *c't*, (7):172–177, July 1997. 336
14. C.-A. Krapp, S. Krüppel, A. Schleicher, and B. Westfechtel. Graph-based models for managing development processes, resources, and products. In G. Engels and G. Rozenberg, editors, *TAGT '98 — 6th International Workshop on Theory and Application of Graph Transformation*, LNCS, Paderborn, Germany, Nov. 1998. Springer-Verlag. To appear. 327
15. S. Krüppel and B. Westfechtel. RESMOD: A resource management model for development processes. In G. Engels and G. Rozenberg, editors, *TAGT '98 — 6th International Workshop on Theory and Application of Graph Transformation*, number tr-ri-98-201 in Series in Computer Science, pages 390–397, Paderborn, Germany, Nov. 1998. Department of Computer Science, University of Paderborn. 327, 331
16. P. Lawrence, editor. *Workflow Handbook*. John Wiley & Sons, Chichester, UK, 1997. 336
17. J. McCarthy and W. Bluestein. *The Computing Strategy Report: Workflow's Progress*. Forrester Research, Inc., 1991. 336
18. K. G. McIntosh. *Engineering Data Management — A Guide to Successful Implementation*. McGraw-Hill, Maidenhead, England, 1995. 336
19. M. Nagl and B. Westfechtel, editors. *Integration von Entwicklungssystemen in Ingenieuranwendungen*. Springer-Verlag, Heidelberg, Germany, 1998. 326, 338
20. A. Schleicher. Formalizing UML-based process models using graph transformations. In M. Nagl and A. Schürr, editors, *AGTIVE — Applications of Graph Transformations with Industrial Relevance*, LNCS, page 17 p., Castle Rolduc, The Netherlands, Sept. 1999. Springer-Verlag. 330, 333
21. A. Schürr, A. Winter, and A. Zündorf. The PROGRES approach: Language and environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools*, volume 2, pages 487–550. World Scientific, Singapore, 1999. 330, 332
22. W. F. Tichy, editor. *Configuration Management*, volume 2 of *Trends in Software*. John Wiley & Sons, New York, 1994. 336
23. B. Westfechtel. A graph-based system for managing configurations of engineering design documents. *International Journal of Software Engineering and Knowledge Engineering*, 6(4):549–583, Dec. 1996. 327, 331
24. B. Westfechtel. *Models and Tools for Managing Development Processes*. LNCS 1646. Springer-Verlag, Heidelberg, Germany, 1999. 327, 338
25. D. Whitgift. *Methods and Tools for Software Configuration Management*. Wiley Series in Software Engineering Practice. John Wiley & Sons, New York, 1991. 336