Electronic version of an article published at International Journal of Software Engineering and Knowledge Engineering, Vol. 10, No. 4 (2000), 511-525.
DOI: 10.1142/S0218194000000298

Distributed Process Planning Support with MILOS

Sigrid Goldmann, Jürgen Münch, Harald Holz
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany
{sigig, muench, holz}@informatik.uni-kl.de

Abstract

Software development processes are highly creative, and therefore prone to change frequently. In addition, necessary expertise often cannot be found at one development site, so that the necessity arises to distribute projects among several sites, or form "virtual" corporations, where software development is shared between several companies. The first point makes it necessary to support changes by identifying and notifying the people affected by a change. The second point emphasizes this necessity, while also complicating project planning and management: changes in one development site or company might necessitate replanning in several locations. In this paper, we introduce the MILOS approach, which provides concepts to integrate process modeling, planning, scheduling and enactment in one system. Thus dynamic plan changes, as well as automated feedback from execution to the project plan, can be supported.

Keywords
distributed planning and modeling; integrated planning, modeling, and enactment; process support; coordination; change notification

1 Introduction

Software processes represent knowledge about software development activities. They can be used to facilitate human understanding and communication, support process improvement and management, automate process guidance and automate execution support [4]. Capturing, storing and using an organisation's process knowledge is complicated by the following typical characteristics of software development: Software processes are inherently *nondeterministic*, *concurrent,* and *distributed*.

The nondeterminism of software processes results from the fact that the sequence of development steps cannot be predicted in advance. Reasons are the existence of many creative development steps (e.g. design steps), possible choices among different alternative paths for plan execution, and product changes triggered from inside or outside the development organization (e.g. defect detection and repair, requirements changes and subsequent product adaptations).

Concurrency and distribution of software processes result from interacting development activities that can be performed in parallel. Especially, outsourcing of development activities and the pressure to incorporate distributed agents enforces the spatially as well as the temporally distributed enactment of software processes.

Providing appropriate process support for these characteristics requires

- a process support system that allows for alternating modeling, planning, and enactment of distributed processes, and
- the explicit description of decisions and dependencies.

This paper explains the main ideas of MILOS, an approach to integrated design support. In our current implementation, MILOS already supports modeling, planning, and executing the software process, and allows for dynamic plan changes. Notifications are sent to the executing agents whenever a relevant part of the plan changes or products are changed which concern them. We are currently extending the system to provide active support (and change notification) not only for technical roles, but also for planners and modelers.

This paper is organized as follows: The different stages of software process support and their interaction are surveyed in Sec. 2. Sec. 3 identifies techniques for supporting cooperative planning and execution in a distributed development environment. Sec. 4 explains the MILOS architecture. Section 5 discusses a typical distributed process in the context of a representative scenario and thereby illustrates the functionality of MILOS. Sec. 6 gives an overview of the current capabilities and limitations of the prototype implementation. Finally, Sec. 7 summarizes the article and gives an outlook on future work.

2 Stages of Software Process Support

Support for software development projects needs to be provided on three stages:

Process Modeling captures essential aspects of real-

world software development activities. A process model is an abstract representation of a development activity (e.g. coding, testing) which explicitly specifies generic knowledge about a set of real-world activities.

Project planning instantiates and tailors process models to specific project situations, and schedules the project according to project deadlines.

During *project enactment*, the project plan is executed, and products are created.

As mentioned above, the activities associated with each of these stages might be distributed among different project sites, and therefore need to be coordinated. Below, we first describe each of the three stages in detail, and then argue that these stages cannot be handled independently from each other.

2.1 Process Modeling

A software engineering process consists of the following information that can be captured in MILOS process models:[1]

Each process has a *name* and general *description*.

A process has input and output *parameters*. These parameters serve as variables that hold the process' input and output products.

Process *preconditions* and *postconditions* specify which facts need to be true before a process can be enacted, and what requirements should be fulfilled once the process has been finished. Conditions can contain variables, which will be instantiated when the process model is used in a concrete project.

For each process, different *methods* can be specified, describing alternative refinements:

- An *atomic method* defines a way to solve the process directly. For example, code inspection can be done in an ad-hoc way, or using a formal reading technique.
- *Complex methods* refine a process into a number of subprocesses, e.g. a component development process is composed of a component design, a design inspection and a component implementation process.

For each method, a set of skill requirements is defined, which describe the resource qualifications necessary to use that method to work on a process.

Resource models describe types of project resources or agents (i.e. people or tools), with the resources' skills and roles. These models can be used during process modeling to describe the type of resource that is needed to work on a process or apply a method.

The general process information described above is stored in a company-specific "experience base", to be (re-) used in specific projects. Other information, like quality models[2] that have proved accurate in past projects, are also stored in this experience base, and can be used to plan a project. A survey of the MILOS representation schema and language can be found in [6].

2.2 Project Planning

Project planning is essentially instantiation of process models. Planning a project comprises

- selecting appropriate processes from the experience base, and inserting them into the plan,
- selecting applicable development methods according to the characteristics of the organizations (e.g. familiarity with specific methods) and the goals of the project (e.g. budget limitations),
- instantiating variables in pre- and postconditions,
- allocating resources to the processes according to the resource properties specified in the respective process models, and
- time scheduling with these resource allocations in mind.

The resulting plan[3] contains the information necessary to enact the plan. It can also be used for monitoring (and controlling) quality values and other relevant information for project management and quality assurance.

2.3 Project Enactment

Once an initial plan has been built, it can be executed: Agents work on processes assigned to them, thereby creating the corresponding output products. As specified in the model, certain conditions must be fulfilled before a process can be started. These conditions can either be always true, or they become true when a certain stage in the project is reached, e.g. when all of a process' inputs are complete and meet the specified quality requirements.

It can also happen that an already finished process becomes executable again, and needs to be redone, e.g. component implementation has to be redone when the component test detects errors in the code.

2.4 Interleaved Modeling, Planning, and Enactment

In case a process restart becomes necessary during project enactment, the original plan will become obsolete if this loop has not been anticipated and provided for in the plan. This is one of many examples which show that modeling, planning, and execution depend on each other and need to be interleaved. Other examples are:

- During planning, it turns out that a new method is needed for a process. The planner should be able to access the model and define that method.

1. Most of the concepts described below are standard process modeling concepts. However, the concept of alternative methods that are applicable to a process, is MILOS-specific.

2. Quality models map measurable influence factors to quality factors of interest.

3. In this paper, we use the terms "plan" and "schedule" synonymously.

- A process definition itself needs to be tailored to the specific project, e.g. additional attributes are needed, or a new precondition needs to be defined.

These two cases might also occur during execution, if the planner did not have the information necessary to predict that the process cannot be used "as is", but needs to be tailored.

In general, it is not always possible to build a perfect model or plan in advance. In complex projects such as software development, it is generally the case that information necessary for planning will only become available during project execution. It is therefore necessary for a usable support system to allow plan changes during project execution, as well as to be capable of executing plans that are incomplete when plan execution starts, and are refined later when the necessary information becomes available.

3 Techniques for Cooperative Modeling, Planning, and Execution

The increasing complexity of software projects creates the necessity to distribute the project between team members, development sites, and companies. In the past, while projects might have been executed in a distributed way, they generally were planned centrally. With increasing project size, as well as distribution of projects between different companies in virtual corporations, it is no longer feasible to always plan a project centrally. That means that support is needed for coordinating distributed project enactment as well as support for cooperative modeling and planning activities. Below, we first describe techniques we use for coordinating distributed project execution, and then we argue how the same techniques can be adapted to support distributed modeling and planning.

3.1 Execution Support

Execution support in our system has two important tasks:

(1) Provide to-do lists for agents, and generate notifications to the concerned agents whenever a process can be started (or needs to be restarted), or when something of interest (e.g. an input product, a scheduled time, or a process definition) has changed.

(2) Provide feedback to the planner about plan violations during execution, and allow the model and plan to be changed during process execution, triggering change notifications to the concerned agents.

The first point provides guidance to the agents, and thereby helps ensure that the "real world" process conforms with the plan. The second, on the other hand, allows the plan to be adjusted when necessary, and therefore prevents the plan from becoming obsolete when execution does not follow the plan in spite of the guidance our system provides.

In order to meet the first requirement, we generate notification dependencies from the project plan, and allow project participants to express interest in specific information. To implement these notification dependencies, we use Event-Condition-Action (ECA) rules that can be based on product and process-specific events. For example, if the precondition of the component testing process demands that the component requirements document should be complete, our system would automatically generate the following rule:

Event: document *component requirements* completed
Condition: the *component test* process has been assigned to *Agent x*
Action: Notify *Agent x*

The second of the above requirements we meet by keeping a project trace, and checking the trace data (like actual start and end times) against the plan. If a process' execution data deviates from its planned values, the assigned agent as well as the responsible planner are notified, so that necessary replanning can be performed. These notifications are also triggered by corresponding ECA rules. See [5] for more details concerning our use of ECA rules.

3.2 Modeling and Planning Support

Not only do modeling and planning depend on enactment data, but there are also dependencies between different modeling and planning activities that concern the same process or subplan. Once these activities are performed in a decentralized way, support is needed to coordinate modeling and planning activities as well as execution activities. For example, the removal of an output parameter in a process model might necessitate replanning in case the output is needed somewhere else in the plan. If these different modeling and planning activities have been distributed between different people, notifications have to be triggered in order to inform all concerned people of the necessity of replanning or re-modeling their processes.

As mentioned in [12], modeling and planning can be seen as a different kind of (meta) process, and can be handled as such. If modeling and planning (and re-modeling and replanning) are seen as tasks in the meta process of "managing a software project", these *meta tasks* can be assigned to agents (i.e. modelers and planners), just like development processes are assigned to resources with the appropriate skills. The meta process can be modeled, and ECA rules can be extracted from that meta process. For example, when the output of the system design process is changed from an OMT document to a UML document, the following ECA rule will be triggered (among other rules):

Event: output type for process *system design* has changed

Condition: process *component design* needs *OMT design document* as an input
AND planning task for process *component design* has been assigned to *Agent y*
Action: Notify *Agent y*

In other words, the same mechanisms can be used to coordinate modeling and planning that we already apply to execution: from the notification mechanisms and ECA rules to agendas for meta tasks, all coordination concepts developed for enacting the software process itself can also be utilized in order to facilitate the meta process of modeling and planning the software process.

4 System Architecture

The MILOS system uses a three-tier-architecture, formed by the modeling component, the planning component and the enactment support system (see [9] for a more detailed description of the MILOS architecture).

The *modeling component* lets the user(s) specify process definitions. These include the concepts specified in section 2.1, like product flow and control flow between processes, product and process attributes, and resource models.

The *planning component* provides functionality for planning and scheduling a concrete project. It allows the planner to select processes defined in the modeling component and customize them for a specific project, as described in section 2.2.

To support resource assignment to processes (in the planning component), as well as agent notifications by the enactment component, our system manages a *resource pool* (RP), which stores information about the available resources, their properties (e.g. skills), their utilization, and their email addresses. The planning component accesses the RP in order to find the agents who's skills are appropriate for specific processes, and checks whether their time schedule allows the assignment of processes to them.

The *enactment support component* is a flexible workflow engine that provides task agendas for enactment clients, and manages the current project state, including product versions and the project trace. It sends email notifications (using the agents' email addresses stored in the resource pool) when products become available/change, or when the plan or schedule is changed.

These three tiers of our (as yet centralized) server can be accessed via World Wide Web (WWW) interfaces. We have defined modeling and planning interfaces as well as agent agendas that provide guidance to the agents who enact parts of the plan. In addition to our own planning interface, the planning component can also be accessed using MS-Project as a planning tool.

5 Scenario

In this section, a scenario is given which demonstrates a typical distributed process: a component development due to given requirements, and a subsequent component testing at geographically distributed sites. While showing typical events (such as the violation of quality criteria) and their consequences for a distributed process support environment, we illustrate the functionality of the MILOS system. In particular, the following planning activities are described:

- distributed method selection with corresponding changes of the data- and control flow,
- distributed time- and resource scheduling and its consequences,
- dynamic distributed replanning caused by plan violations.

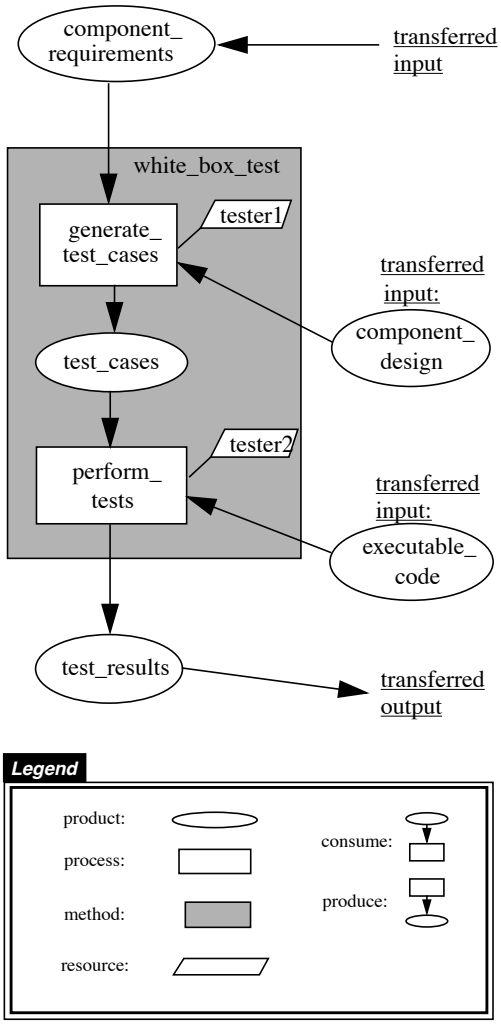

Figure 1 Plan excerpt (test site)

Initially, two processes - *component_test* and *component_development* - have to be instantiated in a global project plan. We assume that these two processes are to be enacted in geographically distributed locations (i.e. a test site and a development site). The instantiated global project plan can be interpreted by the MILOS workflow engine in order to enact the processes.

Planners for both sites are selected: Susan = planner for the test site, John = planner for the development site.

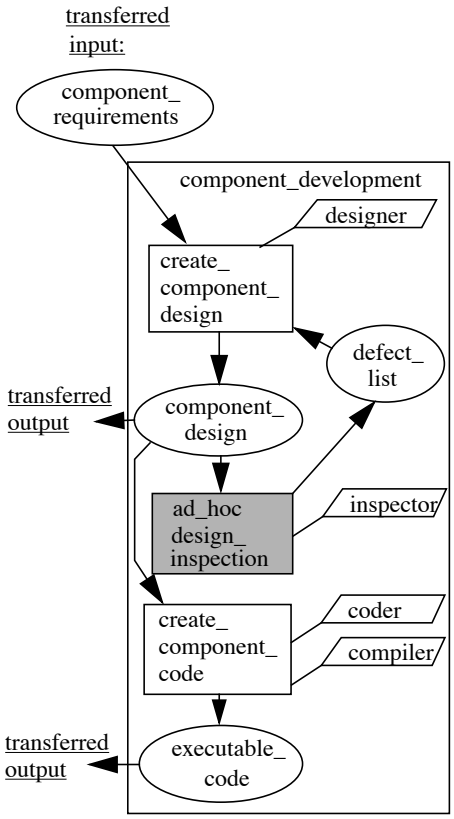

Figure 2 Plan excerpt (development site)

As depicted in Fig. 1 and Fig. 2, Susan and John have already selected a development method for each process before the project starts[1]; for example, Susan (the planner of the test site) has chosen *white_box_test* as test method because there is sufficient experience with this method in the organization. Especially the relationship between the complexity (due to the McCabe complexity measure) and the expected test effort is known with respect to the organization's characteristics. This is explicitly expressed in a predictive quality model (see Fig. 3).

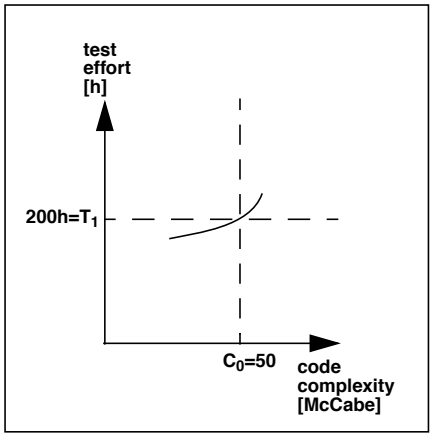

Figure 3 Predictive quality model

During planning, variables (such as effort) in the pre- and postconditions have to be instantiated with desired values due to project goals and characteristics. For our scenario the conditions for the process *generate_test_cases* are:

Precondition:
(component_requirements.status = complete) AND
(component_design.status = complete)
Postcondition:
(component_test.effort ≤ 200h) AND
(component_design.path_coverage ≥ 70%) AND
(test_cases. status = complete)

The desired effort value results from budget limitations, and the desired statement coverage value results from a demanded reliability goal. The planners in both organizations must communicate and adapt the product and control flows.

In our scenario, the planner for the component development process needs to know the planned effort for the test process to predict the maximum complexity of the component design using the quality model. The conditions for the process *create_component_design* consequently are

Precondition:
(component_requirements.status = complete)
Postcondition:
(create_component_design.effort ≤ 250h) AND
(component_design.complexity ≤ 50 [McCabe]) AND
(component_design.status = complete)

1. The depicted plan excerpts are only partially instantiated because further instantiation steps (such as resource assignments) still have to be performed and will be described later.

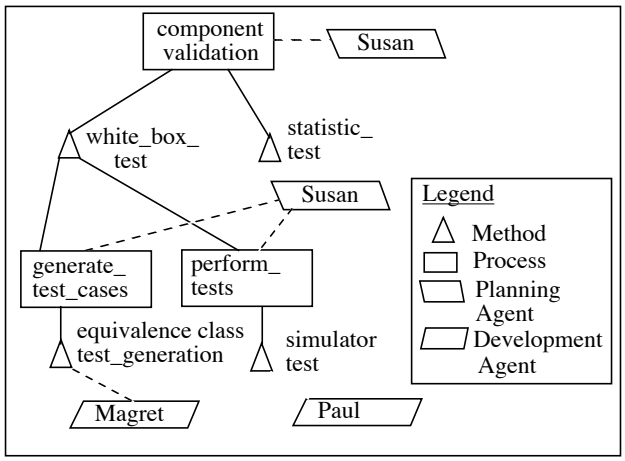

Figure 4 Process and method refinement

The time scheduling is performed in MILOS by adding planned start and finish dates to the processes. These scheduled times are derived by the planners using the processes' pre- and postconditions and external constraints (such as delivery deadlines). The planners perform the initial resource assignment from the resource pool due to available and qualified personnel and tools. As mentioned above, Susan is the responsible planning agent for the component validation process and assigns herself as planning agent for the subtasks of the method *white_box_test*. Then she assigns Magret and Paul as development agents to the atomic methods (see Fig. 4). Fig. 5 shows the resulting plan in MS-Project with the following assignments: tester1 = Magret, tester2 = Paul, designer = Colin, inspector = Vic, Coder = Tom, Compiler = Sun Java Compiler.

Figure 5 Global plan in MS-Project

Now let's start project execution. The first situation under consideration is that the finish date of the process *create_component_design* is exceeded by two days because the actual effort has surpassed the predicted effort by 16 hours. The reaction is, that John, the planner of the development site, reschedules the start and finish times for the subsequent processes *design_inspection* and *create_component_code*. Furthermore, he notifies Vic and Tom about the delay. Additionally, Susan, the planner of the test site, should be notified about a delayed delivery of the product *component_design* which is input for the process *generate_test_cases*. This process may have to start later than originally planned, so that it is not guaranteed that the existing personnel is available during the new time slots. Reassignments and notifications may be necessary.

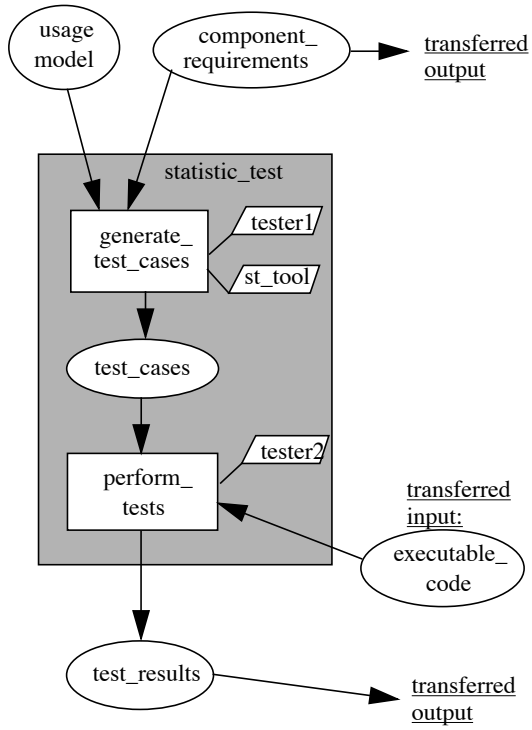

Figure 6 Replanned test method

Let us now assume that the complexity of the developed component design is too high. If there were enough time, it would be possible to redesign the component with a complexity that fits the postcondition of the process. However, since there already is a time delay, it is not possible to redesign the component. Several other possibilities exist: First, the planner on the test site can increase the test effort due to the quality model, and reschedule time and reassign personnel as described above. Second, the reliability requirements for the test process can be reduced (e.g. reduction of the minimum path coverage to 30%), so that the time limitations can be fulfilled. Third, a different test method can be chosen, e. g., statistic testing (see Fig. 6). This implies

a) a rejection of the method *white_box_test*,

b) changes of the product flow (additional input product *usage_model*; *component_design* is no longer needed for testing),

c) a change of the pre- and postcondition (e.g. replace the path coverage value by reliability and significance values),

d) a reassignment of the resources (do Paul and Margret have experience with stochastic testing?, assignment of the additional test tool st_tool), and

e) notification of the involved personnel.

6 State of Implementation

The current implementation of our system can satisfactorily handle the above scenario by providing the modeling and planning functionality necessary to build the global plan described in section 5, and by notifying all involved agents of changes during project enactment, as described in section 2.4. However, MILOS does not yet provide support to explicitly model the meta process of modeling and planning the project, and therefore the different meta tasks cannot be assigned to the appropriate planners. With respect to the above scenario, this means that while the enactment agents are notified individually of the changes that concern them, MILOS cannot yet identify which of the two planners at the different sites needs to be notified of the necessity of replanning. The current system version handles this problem by notifying *all* involved planners whenever the necessity of replanning occurs.

The MILOS system has been implemented in Java, using the object-oriented database GemStone/J 2.0 as an Enterprise Java Bean (EJB) server that provides transaction management and persistency services. This server manages the process model and project plan, and provides support for project enactment. Clients are responsible for modeling, planning and executing software development processes. They are stand-alone applications or Java applets which access the server via HTTP or using a Java Remote Method Invocation (Java RMI) interface. The data exchange with MS-Project has been implemented using the MS-Project API, which allows import and export of MS-Project plans in a predefined exchange format. Our system is able to import and export plans, and to identify similarities and differences between imported plans.

Our system also includes an interface to a software metrics tool [16]. This tool can be automatically triggered to take process and product measurements at specified points in the project (e.g. when a process is started/finished, or when a product becomes available).

7 Conclusions and Future Work

MILOS is a process modeling and enactment approach which not only supports modeling and enactment in the same system, but also provides project management functionality in the form of planning and scheduling support. This allows us to guide project execution according to the project plan and process model, as well as to keep the plan up-to-date by feeding back enactment information into the plan. MILOS' flexible workflow engine allows the model and plan to be changed during project enactment, and provides support for process restarts whenever necessary. (See [10] for an example problem that can be supported and facilitated using the MILOS system.) We support distributed project enactment by allowing the clients access to the workflow engine via the WWW. We are currently extending our approach to allow off-line execution of partial plans.

In order to provide more active support for distributed modeling and planning, we are currently extending MILOS to provide notifications services for planners and modelers by explicitly modeling the meta process (i.e. the management activities that have to be done in addition to the actual development process, which include planning and modeling), and allowing these meta tasks to be assigned to individual people. We are currently identifying the dependencies inherent in this meta process which can be extracted in order to generate the appropriate ECA rules to support it. We are also planning to address the idea of versioning plans and models (and linking them to the project trace and the products produced during execution), in order to be able to store different plan versions for later reuse.

8 Related Research

Our work bears similarities to several areas of research, particularly project management tools, workflow management approaches, process modeling and enactment research.

Commercially available project management tools like MS-Project and Autoplan support project planning and scheduling, but provide little or no enactment support. A project management system that does provide both planning and execution support is the Mesa/Vista Enterprise tool. Mesa/Vista Enterprise is an environment for collaborative project execution and management. It provides distributed access to project data, as well as version and configuration management, but it does not include any change notification services.

Workflow management tools like Staffware, FlowMark, or TeamWARE concentrate on project execution and provide little or no support for process modeling and project planning. In particular, plan changes during enactment require a complete restart of the project in most workflow management tools.

The approaches most similar to our work can be found in the area of process modeling and enactment research, e.g. Endeavors [2], Serendipity [7], OzWeb [8], EPOS [11], and SPADE [1]. Most approaches in that area provide (web-based) modeling and enactment functionality, as well as some support for dynamic plan changes and change notifications. However, most of these approaches do not

provide project planning and management support, like resource allocation and time scheduling for tasks in the project. See [10] for an overview over these approaches.

Multi-view approaches in the area of process modeling allow distributed modeling of objects in different styles and representations (e.g. control-flow view, abstraction hierarchy view, role-oriented view). These approaches can be classified according to their integration mechanisms. One class is characterized by separate modeling of different views and subsequent integration. A representative of this class is the MVM approach (multi-view modeling) [15]. This approach is based on role-specific views, which are modeled independently using the formal process modeling language MVP-L [3]. Finally, the integration of views is performed with similarity and consistency analyses and the creation of a comprehensive software process model.

The other class of approaches is characterized by the distributed modeling of a common model. This implies the permanent application of consistency checks and updating operations. A typical approach of this class is the MUVIE approach [13]. Here, each view defines a focus on an underlying graph structure model. The modeler only handles those parts that pertain to a specific view. The underlying semantics which guide incremental changes are expressed by a graph model and graph replacements.

Acknowledgments

The MILOS system was developed and implemented in cooperation with Prof. Dr. Frank Maurer' research group at University of Calgary. Barbara Dellen, Boris Kötting, and Fawsy Bendeck were involved in the conceptual work as well as the implementation of the MILOS system. The work was supported by NSERC, Nortel, the University of Calgary, and the DFG with several research grants.

Literature

1. S. Bandinelli, A. Fuggetta, S. Grigolli: *Process Modeling-in-the-large with SLANG*. In IEEE Proceedings of the 2nd International Conference on the Software Process, Berlin (Germany).
2. G.A. Bolcer and R. N. Taylor: Endeavors: *A Process System Integration Infrastructure*. in Proceedings of the Fourth International Conference on the Software Process, Brighton, England, December 1996.
3. A. Bröckers, C. Lott, H. Rombach, M. Verlage: *MVP-L language report version 2*. Technical Report 265/95, Department of Computer Science, University of Kaiserslautern, Germany, 1995.
4. Bill Curtis and Marc I. Kellner and Jim Over: *Process Modeling*. Communications of the ACM, Vol. 35, No. 9, September 1992.
5. B. Dellen, F. Maurer: *Change impact analysis support for software development processes*. Journal of Applied Software Technology, International Academic Publishing, 1998.
6. B. Dellen, F. Maurer, J. Münch, and M. Verlage: *Enriching Software Process Support by Knowledge-based Techniques*. In Int. Journal of Software Engineering and Knowledge Engineering, Volume 7, No. 2, pp. 185-215, 1997.
7. J.C. Grundy and J.G. Hosking,: *Serendipity: integrated environment support for process modeling, enactment and work coordination*. Automated Software Engineering: Special Issue on Process Technology 5(1), January 1998, Kluwer Academic Publishers, pp. 27-60.
8. G.E. Kaiser, St.E. Dossick, W. Jiang, J. Jingshuang Yang and S.X. Ye,: *WWW-based Collaboration Environments with Distributed Tool Services*. World Wide Web, Baltzer Science Publishers (to appear).
9. F. Maurer, B. Dellen: *A Concept for an Internet-based Process-Oriented Knowledge Management Environment*. Proceedings of the KAW'98, Banff, Canada, 1998
10. F. Maurer, G. Succi, H. Holz, B. Kötting, S. Goldmann, B. Dellen: *Software Process Support over the Internet*. submitted to ICSE99, Formal Demonstration Track.
11. M.N. Nguyen, A.I. Wang, R. Conradi: *Total Software Process Model Evolution In EPOS*. Submitted paper for 4th ICSP, 1996, Brigthon, UK.
12. Ch. Petrie, S. Goldmann, A. Raquet: *Agent-Based Project Management*. to appear in Springer LNAI 1500 special volume on Artificial Intelligence Today.
13. Peter Rösch, "User Interaction in a Multi-View Design Environment". IEEE Symposium on Visual Languages (VL'96), Proceedings, pp. 316-323, Boulder, Colorado, Sept. 3-6, 1996. USA, IEEE Computer Society Press, ISBN 0-8186-7508-X.
14. S. Sutton, L. Osterweil, D. Heimbigner: *APPL/A: a language for software process programming*. IEEE Transactions on SE and Methodology, Vol. 4, No. 3, p. 221-286, 1995.
15. Martin Verlage, "An Approach for Capturing Large Software Development Processes by Integration of Views Modeled Independently". 10th International Conference on Software Engineering and Knowledge Engineering (SEKE98), June 1998.
16. G. Succi , Webmetrics project. URL: http://www.sern.enel.ucalgary.ca/Charmi/ResearchProjects/Webmetrics/