The SMART Approach for Software Process Engineering

(Research Paper)

Pankaj K. Garg[*], *Peiwei Mi*[†], *Thuan Pham*[*],
Walt Scacchi[†] *and Gary Thunquest*[‡]

Abstract

In this paper we describe a methodology for software process engineering and an environment, SMART, that supports it. SMART supports a *process life-cycle* that includes the modeling, analysis, and execution of software processes.

SMART's process monitoring capabilities can be used to provide *feedback* from the process execution to the process model. SMART represents the integration of three separately developed process mechanisms, and it uses two modeling formalisms (object-oriented data representation and imperative-style programming language) to bridge the gap between process modeling, analysis, and execution.

SMART demonstrates the *meta-environment* concept, using a process modeling formalism as input specification to a generator that produces Process-Centered Software Engineering Environments (PSEEs). Furthermore, SMART supports a team-oriented approach for process modeling, analysis, and execution.

1 Introduction

In a typical large-scale software engineering effort, a variety of activities are carried out by several people over extended time periods. *Process models*, as representations of these activities and their characteristics, provide the following benefits:

- With the use of a process model, the steps that need to be carried out for a project can be made explicit to answer questions such as, *what should be done next?*.
- Often, to ensure the quality of a software system, *quality guidelines* suggest that a set of activities be carried out in a particular order. A process model can help in determining the relationship between the quality guidelines and the process, e.g., whether the process is in conflict with the quality guidelines or not.
- A process model can be *analyzed* with respect to its internal consistency, completeness, and correctness. For example, we can find out the maximum number of parallel activities that can be carried out in a process (for scheduling purposes), or whether there are any redundant activities in a process (the outputs of which are not being used).
- Usually, a software process involves routine, automatable activities that are mixed with activities requiring creative thought. Based on a process model, such routine activities can be extracted from the process and embedded within a computing environment such that they can be *automated*.
- A large-scale software effort requires *coordinating* the work of several people over extended time periods. An explicit process model helps team members understand and coordinate *who* is doing *what* and *when*. Similarly, *communication* between people can be improved as individuals can better understand the information needed by other activities and individuals.
- With the use of a process model, process objectives can be developed, and *measurements* defined, to collect data to analyze the process execution against these objectives. These can be used for both continuous process improvement or to radically change the process.

A common hypothesis within the process modeling research community is that the benefits of process models will be easier to realize with the use of a multi-formalism approach to process modeling (for example, see [2, 8]). In order to experiment with this hypothesis,

[*]Hewlett-Packard Labs., Palo Alto, CA 94304, USA.
[†]Information and Operations Mgmt. Dept., University of Southern California, Los Angeles, CA 90089, USA.
[‡]Software Engineering Systems Division, Hewlett-Packard Company, Fort Collins, CO. Now at Eriksen, McClure & Associates Inc., 4840 Pearl East Circle, Suite 301E Boulder, CO 80301

0270-5257/94 $3.00

we have developed the SMART approach for software process engineering. SMART combines two formalisms for process modeling within a single framework.

Within the SMART approach, we use both an object-oriented knowledge-based formalism for process modeling and an imperative-style programming language for process programming. The roles of the formalism are made clear with a methodology supporting the *process life-cycle*. The object-oriented representation is used mainly for process analysis and quality assurance, while the programming language representation is used to provide process guidance, automation, and measurement.

SMART provides a mechanism to derive the process programming language representation from the object-oriented representation. This is akin to the code-generation capabilities of common application generators. In this regard, SMART is a *meta-environment* that accepts a process model as its input specification, then generates a process program which produces an executable PSEE [14]. However, since there is additional information available in the process program representation regarding activities, this step cannot be fully automated in all cases.

Early results of working with SMART have successfully demonstrated the utility of the approach. For example, we have used SMART for modeling the change management process that has been used in the research community [15], as well as more complex processes, such as those conforming to MIL-STD-2167A. In each case, the programming language version of the process model was automatically generated from the object-oriented representation.

In this paper we describe the various concepts underlying SMART and their implementation. We start in Section 2 with a description of the SMART process engineering life-cycle. An overview of the architecture of SMART is given in Section 3. In Section 4, we describe the SMART support for the various stages of the process life-cycle, including: process modeling, analysis, embedding, execution, and feedback. We discuss related work in Section 5. Finally, we conclude with some suggestions for future work along these directions in Section 6.

2 Process Life-Cycle

One approach to process modeling is to consider the models as *programs* in the traditional programming sense [20]. An important benefit of process programs is that they can be machine executable and therefore

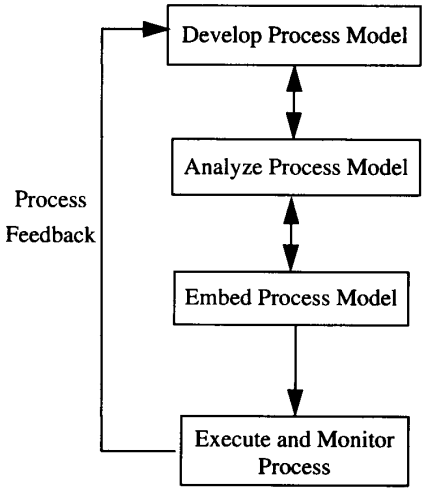

Figure 1: The SMART Software Process Engineering Life-cycle

automated. However, much like the development of complex software systems entails more than programming, similarly the development of complex software processes–those needed to support the development of large or very large software systems–entails more than process programming. As such, our work has led to an initial formulation of a software process engineering life-cycle that is founded on the incremental development and iterative refinement of software process models, as shown in Figure 1.

Four stages of the software process engineering life-cycle, which we focus in this paper are process modeling, analysis, embedding, and execution with monitoring.

- **Process Modeling**

  Process modeling involves eliciting and capturing informal process descriptions and converting them into formal process models. The concepts used in defining a process model usually depend on the considerations that are important for the organization, and the process model is best developed in conjunction with the people who are participants in, or are affected by, the process. Therefore, at this stage of the process life-cycle, it is important that the concepts used in the language for process modeling be familiar to the people affected by the process and well understood

by them. For this reason, we advocate the use of *meta-modeling* wherein a process modeler can specify the vocabulary and concepts used for process modeling.

- **Process Analysis**

  This involves the evaluation of the static and dynamic properties of a process model, including its consistency, completeness, internal correctness, and traceability. Examples of useful static analysis are to find out: the maximum number of activities that can be carried out in parallel within a process, the number of activities that use the output of a particular activity, and other descriptive statistics.

  At this stage, one might want to carry out a simulation that involves symbolically executing process models in order to determine the path and flow of intermediate state transitions in ways that can be made persistent, replayed, queried, dynamically analyzed, and reconfigured into multiple alternative scenarios. For example, hypothetical agents and resources can be assigned to the process and the process engine started. During the execution, one can discover dependencies between activities and agents, e.g., $Agent_1$ cannot start any work unless $Agent_2$ has finished the *requirements* activity.

  Multiple graphic views or visualizations of the software process at this stage help in understanding process flow relationships. For example, sometimes it is useful to view the process from an activity viewpoint, while at other times it might be useful to view it from a data-flow or role-specific viewpoint.

- **Process Embedding**

  Once a process has been successfully analyzed for various properties, it can be embedded and executed within a software engineering environment. This involves assigning and scheduling specified users, tools, and data objects to the process.

  A shift in process representation needs to occur at this stage. While the process model in the modeling and analysis stages of the life-cycle is mainly used for communication, understanding, and analysis, the process model at the embedding stage is mainly used to derive an executable PSEE. Therefore, specific software tools, such as Emacs, need to be associated with the process model, whereas in the earlier stages it would have been sufficient to say that a tool of class "text-editor" is required. Similarly, in the earlier stages we could have modeled a "Requirements Document" in the abstract, while at this stage that will be bound to some data object identifier, which eventually resolves to a path specification or named file in the local network file system.

- **Process Execution and Monitoring**

  Finally, the process is executed within the organization. The PSEE that was generated in the stage above is used to guide or enforce the process. While the process is being executed, it can be monitored by the PSEE such that information regarding the ordering and duration of activities can be tracked. In addition, any departures from the specified process can be collected. Such information can be abstracted and fed back to the first stage of developing the process model.

- **Process Feedback**

  An important aspect of our process engineering life-cycle is that it does not assume that once a process model has been developed it remains fixed forever. On the contrary, we anticipate a process engineering life-cycle in which the process models can evolve to accommodate changes in the execution environment of the process. An example change in the environment could be when a new way of doing a particular aspect of the process is discovered. For instance, suppose that programmer modifies code by using an edit-compile-debug cycle, and that this has been modeled in the embedding knowledge-base. Therefore, whenever there is a task of modifying code, it is embedded within the PSEE with the activities of edit, compile, and debug. At some point during the execution of these activities, a programmer may discover a static analysis tool and start using it with the edit activity. Therefore, the task of modifying code becomes into an edit-analyze-compile-debug cycle. The new activity of analyzing the code can be recognized by the PSEE, and an appropriate message can be sent to the process embedding tool.

  This feedback can be utilized by the process embedding tool to improve its transformation of future process models. Therefore, future programmers need not discover the use of the static analysis tool themselves, since knowledge of its use will be available for them from past experience with the process embedding tool.

In this manner, the process life-cycle is an evolutionary life-cycle in which the processes developed are incrementally enhanced and continuously improved.

3 SMART Architecture and Implementation

The high-level architecture of SMART is shown in figure 2. The major components of SMART are:

- A **Team Database** that maintains the process model developed during the modeling and analysis stages of the process life-cycle. This is a multi-user object-oriented database.
- A set of workspaces called **Workshops** [5] that maintain a role-specific process model for each person on the process modeling team.
- A set of **Editors** and **Browsers** for each person on the process modeling team that allows them to manipulate the process model.
- The **SynerVision** process execution and monitoring tool connected to a host of **SoftBench** compatible tools through a Broadcast Message Server (BMS). SynerVision, SoftBench, and BMS are commercially available products from Hewlett-Packard. However, a large number of CASE vendors now provide tools that are compatible (i.e., encapsulated to run) with SoftBench, thus the range of possible PSEEs built with SMART is substantial.

SMART represents the integration of three separately developed process mechanisms: SynerVision from HP's SESD product division, Matisse from HP Laboratories, and the ARTiculator from USC. SynerVision is a process execution and monitoring tool that operates with the SoftBench programming environment. Matisse is a knowledge-based team programming environment [10]. The Articulator is a knowledge-based process modeling, analysis and simulation system [18]. This combination was facilitated by three main characteristics of the systems being combined: (1) the *extensibility* of Matisse [10], (2) the Articulator's meta-modeling formalism, and (3) the *openness* of SynerVision. The Matisse team programming environment maintains an object-oriented team information base of software related information. This object hierarchy was extended to incorporate the concepts and mechanisms of the Articulator meta-model

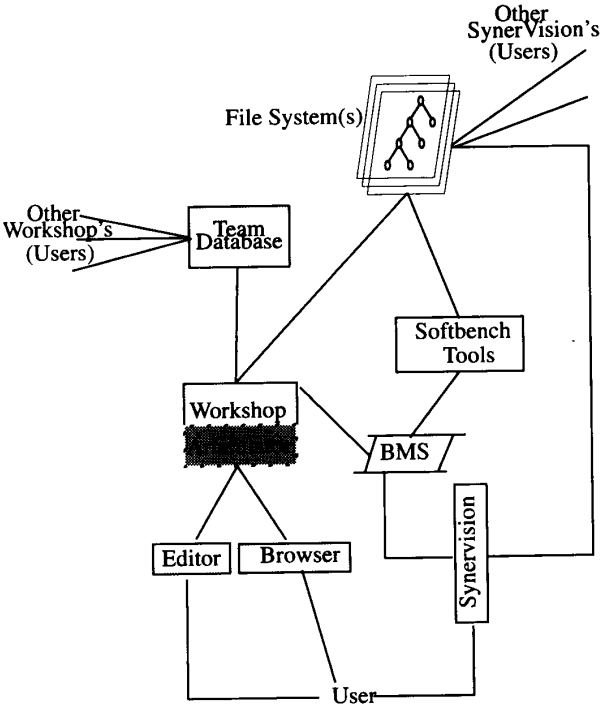

Figure 2: The Architecture of SMART

formalism. In this way, all process modeling and analysis functionality, as well as all process models operational with the Articulator's formalism, were ported with little effort. Last, SynerVision is an open tool that broadcasts information about its activities on a Broadcast Message Server. Appropriate messages can therefore be sent to SynerVision and messages from SynerVision can be used for process feedback.

As shown in the figure, SMART is a multi-user system that spans and supports the process life-cycle. At the process modeling and analysis stages, the multi-user capabilities are provided by the maintenance of the process model within the Matisse team programming environment. Matisse supports multiple users by providing optimistic concurrency control on a *shared information space* with each user having their own *individual information spaces*. The information spaces (both shared and individual) contain medium-grained versioned software objects with automation and consistency rules on their property modifications [10]. At the process execution and monitoring stages, SynerVi-

sion provides multi-user capabilities by using the capabilities of the Network File System (NFS). SynerVision supports both individual and team tasks as well as task delegation. Therefore, SMART provides role-specific workflow management for all team members with process guidance (or enforcement), automation, and performance feedback.

4 SMART Support for Process Engineering Life-Cycle

Based on the description of the process engineering life-cycle in Section 2, it is clear that the life-cycle is as complex as, or perhaps more complicated than, the traditional software product life-cycle. Therefore, much as the software product life-cycle requires a comprehensive complete solution [11] so does the PE life-cycle. The SMART approach represents a step in this direction by providing support which spans the process life-cycle.

In this section, we present an overview of the support capabilities of SMART for the four stages of the process engineering life-cycle identified earlier.

4.1 Modeling, Analysis and Simulation

SMART utilizes the knowledge-based Articulator approach for modeling, analyzing, and simulating complex organizational processes [18]. The Articulator utilizes an object-oriented knowledge representation scheme for process modeling.

4.1.1 Modeling

The Articulator's resource taxonomy, explained in detail elsewhere [18], serves as a *process meta-model* that provides an ontological framework and vocabulary for constructing *software process models* (SPMs). At the base level, the process meta-model states that software processes can be modeled in terms of *agents* who perform *tasks* using *tools or systems* that consume (utilize) or produce (modify) *resources*. Further, agents, tools, and tasks are resources (i.e., resource subclasses), which means they can also be consumed or produced by other agents and tasks. For example, a project manager may produce staff through staffing and allocation tasks that consume departmental budgets. These staff may then assigned to other routine or creative production tasks using the provided resources (e.g., computer workstations, CASE tools, desktop publishing packages, schedules, and salary) to construct the desired products or services (e.g., application programs and documents). *Instances of SPMs* can then be created by binding values of corresponding real-world entities to the classes of corresponding entities employed in the SPM. For instance, Mary may be the project manager who is responsible for getting a set of documents produced for an external client, and she is authorized to assign 2-3 individuals in her department to use their desktop Unix workstations that run Motif 1.2 and FrameMaker software in order to get the reports produced by the end of the week.

The agents, tasks, product resources, tools, and systems are all hierarchically decomposed into subclasses of arbitrary depth that inherit the characteristics of their parent classes. Further, these resource classes and subclasses are interrelated in order to express relationships such as: control-flow relationships (sequential, iterative, conditional, optional, or concurrent), task/resource pre- and post-conditions, authority relationships among agents in different roles, product compositions, SE tool/system aggregations, and others [18]. Thus, in using these classes of process modeling entities, we are naturally led to model SE processes as a web of multiple interacting tasks that are collectively performed by a team of developers using an ensemble of tools to consume and produce products [16].

In addition, the meta-model enables us to model other complex phenomena associated with organizational processes, such as agents' resource sovereignties (i.e., the set of resources under the control of an agent), authority relationships among agents, articulation strategies [19], technology transfer strategies, etc. Accordingly, these relationships are defined in the meta-model, used and then instantiated in the SPMs. Then, we can use SMART to query, analyze, and simulate process models (cf. [18]).

4.1.2 Analysis

As the process meta-model provides the semantics for SPMs, we can construct computational functions that systematically analyze the consistency, completeness, traceability and internal correctness of SPMs [18]. These functions represent batched or interactive queries to the knowledge base. At present, we have defined a few dozen parameterized query functions that can retrieve information through navigational browsing, direct retrieval, or deductive inference, as well as what-if simulations of partial or complete SPMs [18]. Further, most of these analysis functions incorporate routines for generating different types of reports (e.g., raw, filtered, abstracted, para-

phrased, or publication format) that can be viewed interactively or incorporated into publishable documents.

4.1.3 Simulation

Simulation entails the symbolic performance of process tasks by their assigned agents using the tools, systems, and resources to produce the designated products. Using the previous example, this means that in the simulation, Mary's agent would "execute" her project management tasks according to the task precedence structure specified in the SPM instance, consuming simulated time and effort along the way. The simulation makes progress as long as task preconditions or post-conditions are satisfied at each step (e.g., for Mary to be able to assign staff to the report production task, such staff must be available at that moment, else the simulated process stops, reports the problem, then waits for new input or command from the simulation user).

We have used the Articulator environment to model, analyze, and simulate a variety of large-scale SE processes, including those in use in industrial organizations (e.g., [22]). However, at present, our focus in on supporting symbolic rather than analytical (e.g., discrete event) simulation capabilities, whereas analytical simulation capabilities are required to simulate a large sample of process instantiations.

4.2 Process Embedding

The process embedding stage involves the semi-automated transformation of the process representation from the object-oriented representation, of the analysis stage, to a representation suitable for execution within a PSEE. Figure 3 shows this transformation process.

As shown in the figure, the object-oriented process model is first automatically transformed into a **process template** via the **model transformer**. This template is a description of the process in an extended Bourne shell script language [4] that is understood by SynerVision [13]. The process template can be modified by the **process-engineer** to provide additional information, perform name substitutions, etc. These are typically once-only modifications; changes that are more commonly used can be specified as rules to the model transformer.

The process template is then *instantiated* within SynerVision, resulting in a process-centered software engineering environment. In this manner, SMART illustrates the capabilities of a meta-environment [14],

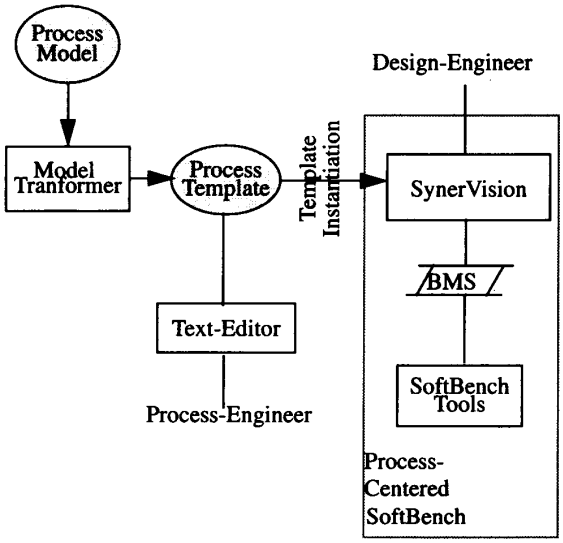

Figure 3: The *Embedding* of a Process Model in a Process-Centered Software Engineering Environment.

i.e., an environment for the generation of an environment.

The model transformer needs to convert the various abstract representations, i.e., process model classes, to actual instances of environment components. For example, whereas in the analysis stage it would suffice to say that a **text-editor** is required for the **modify-code** activity, for the template we need to specify the exact editor from the environment, e.g., SoftEdit from SoftBench, that will be required for the process execution. Similarly, we need to associate the object identifiers (e.g., file path name) that will represent the various modules of the target system.

Some of the knowledge required by the generator can be coded into the object hierarchy for the process models. For example, an organization can *a priori* specify the various text-editors that are available for use. In cases where the constraints identify only one choice for a class of objects, that choice can be made automatically by the generator; otherwise, a multiple choice menu can be generated for the process engineer's assistance.

In this manner, over a period of time, an organization can evolve a rich knowledge-base for automating future processes. If a project has embedded, success-

fully used, and refined process models through SMART, then subsequent projects can be aided by the choices made in prior projects. Similarly, the generation of a changed process model that has been evolved within SMART can benefit from the choices made in the previous embeddings of the model. Thus, SMART is designed to both facilitate process improvement and to iteratively mature process models.

4.3 Process Execution and Monitoring

Process execution within SMART is supported through HP's process execution product called SynerVision [13]. Details on this tool can be obtained from the technical literature on the product. As such, we present highlights to illustrate the process execution and monitoring capabilities of SynerVision.

SynerVision presents the user with an agenda-based window that lists the tasks for that user and possibly the tasks of other people in the project team. The tasks are displayed with a set of attributes that are determined by the user using a rich filter mechanism. The tasks can be displayed either in a graphical task hierarchy or they can be presented as a nested list.

The user clicks the mouse at task representations to perform various operations on them which check and/or modify task attributes. The semantics of an attribute modification determine the actions that need to be taken for the PSEE. For example, an attribute modification of the status of a task from **New** to **Execute** means that work on the task is started. SynerVision checks to make sure that this transition does not violate any constraints that may have been specified in the process. Also, a menu called **Actions** is activated with a list of activities available with the currently executing task. The user can then use that menu to *instantiate* process actions. A simple action may be to invoke another tool in the PSEE using the Broadcast Message Server (BMS). A complicated action may involve the execution of a Unix Shell script [4] that may run over a period of time. Finally, users can also add new tasks to a process template already instantiated in SynerVision. However, tasks in an instantiated process template that a user chooses not to perform can be marked as "abandoned" but cannot be deleted.

SynerVision allows process users to both estimate and track time for task completion. A clock mechanism tracks the amount of time spent in "executing" a task. In order to ensure privacy of time data, the time a user spent on a particular task is visible to that user only. The user can at her discretion choose to share that information with other team members. Timing information is also generated as a log file that can be edited for sharing purposes. Thus, SynerVision helps process users identify variances in estimated versus actual time allocation, whereby high variances indicate tasks or task steps whose definition could be revised or otherwise improved.

There are pre-defined attributes that characterize each tasks, e.g., status, owner, duration, etc., and additional attributes can be added by the user. Tasks can be delegated to other members of the project by modifying certain task attributes (e.g., changing the owner), then dispatching the task. In turn, users who receive delegated tasks can then accept or refuse them. Tasks also have a text attribute called **Notes**. Users of the PSEE can use this field to enter rationale for actions taken or feedback about the process, e.g., how useful a particular task description was, whether a task needs to be decomposed into several smaller activities, useful hints that help when performing the task, etc.

The execution of the process is tracked by SynerVision for various information such as time spent on an activity, tools invoked, users who have worked on different tasks, and any user notes or feedback. This data is made available to the users in a variety of standard report formats.

4.4 Process Feedback

Process feedback is implemented within SMART by exchanging messages through the Broadcast Message Server (BMS) between SynerVision and the process modeling knowledge-base. For example, when a new action is added to a task, or a new person is added to the access list for a particular task, SynerVision broadcasts a message on the BMS indicating that the particular modification has been made to the task. This message can be unparsed by the modeling tool to add to its knowledge base. When a new action has been added, the knowledge-base adds this fact to its task decomposition. When a new user is added to the access list, the knowledge-base adds this information to its agent specification for the task. Over a period of time, this information can be used to help characterize how the process evolved, which is useful for subsequent model development.

Another useful feedback mechanism is the process activity logs. As each activity is performed by a user, SynerVision's messages can be used to log the frequency with which each activity has been executed, the ordering of activities with respect to each other, and the objects on which the activities have been invoked. This information can then be analyzed by an

automated tool to infer redundant activities of the process model, incorrect dependencies or ordering of activities in the model, and so forth, all of which can be used to revise and improve modeled processes.

5 Related Work

The PRISM project [17], which has a methodology for developing process models and a PSEE, is the work most closely related to ours. The process engineering life-cycle that we suggest is similar to the four vertices of the PRISM approach. However, the PRISM approach does not support the concept of *meta-modeling*, which is an important aspect of our approach. Using meta-modeling, a process modeler can define the concepts that will be used for process modeling. In the PRISM approach, the modeler uses the concepts of FUNSOFT nets for process modeling. An important aspect, missing from the PRISM approach described in [17], is the stage of process execution feedback for improving the process model and aiding in the development of future process models.

The process engineering life-cycle that we suggest, as part of the SMART approach, is similar to the improvement paradigm suggested by Basili and Rombach, in the TAME approach [3]. Both the SMART and the TAME approach share the idea of building up an experience base that can be useful for future software project planning. However, the emphasis in the TAME approach is on collecting metrics based on the Goal-Question-Metric (GQM) paradigm; the emphasis in the SMART approach has been to provide a general framework in which different kinds of analysis are possible. In this regard, we hypothesize that a GQM paradigm can be implemented in SMART, using its meta-modeling capabilities. Moreover, the supporting environment described in the TAME approach was quite weak, a reflection of the state-of-the-art in Software Engineering Environments at that time [3]. The SMART support environment with SynerVision, Matisse, Articulator, and SoftBench, overcomes this limitation.

Another tool that uses multiple formalisms to support the different stages of the process life-cycle is Process WEAVER, built by Cap Gemini Innovation of France [9]. However, the multiple formalisms in Process WEAVER are used for different purpose - they are used to model different aspects (and details) of a process. Therefore, automatic generation of a process template, or concepts of process feedback are not relevant within the Process WEAVER context. Moreover, Process WEAVER uses a modified Petri-net based approach (transition nets) for modeling the activities of a process, which limits it to an activity-centered view of the process. The object-oriented, rule-based representation of SMART can simultaneously provide a product-centered, activity-centered, or a resource(agent)-centered view of the process. As with SMART, Process WEAVER supports the integration with other tools in the environment using the services of a BMS, and Process WEAVER provides an *agenda* based view of a user's work context.

The MELMAC [7] environment also supports the notion of using multiple formalisms for process representation. In the MELMAC environment, multiple application views or layers of the process model are represented in the same intermediate level. The intermediate layer is represented using an extended Petri-net formalism of FUNSOFT nets. MELMAC also supports the ideas of process visualization, simulation, and to some extent feedback. However, in the SMART approach, the process models are not simply views of the process-in-execution, but they are a totally different representation of the process. In this manner, the SMART models can be quite different from their process-in-execution counterparts, thereby permitting a greater degree of freedom in supporting the different stages of the process life-cycle.

Several research projects in Europe are also advocating the use of some extended form of Petri-nets for modeling and analyzing software processes [1, 12]. In contrast, we advocate a knowledge-based approach to process modeling and analysis, for several reasons. The knowledge-based approach provides for incremental specifications of the process models such that we can query, analyze, and reason about a partial specification of a process model. Using a knowledge-based approach, information about the organizational setting can be captured once and encoded in the knowledge-base. Subsequently, that information does not need to be repeated for each new process model. Finally, a knowledge-based approach supports abstraction mechanisms for data products, processes, and organizational structures much more easily than Petri-nets can. On the other hand, Petri-nets are best at representing issues about concurrent activities and their analysis.

To summarize, none of the prevailing software process engineering environments today supports the full process engineering life-cycle. However, we have been able to demonstrate supporting mechanisms for the process life-cycle activities described in Section 2. Similarly, it should be noted that though our focus is

targeted at software process engineering, our approach can also be applied to other engineering domains (e.g, Electronic Design Automation, Agile Manufacturing) and to conventional business processes, albeit in a radically innovative way [6].

6 Status and Future Work

The SMART prototype is currently operational. We have used it to model several processes and generate process programs from them. So far we have not experimented with it for executing and monitoring any actual industrial software projects, which we hope to do in the future. Such experiments will be able to test the veracity of the feedback mechanisms that we have built.

Early trial experiments, within our group, suggest that an important support aspect missing from SMART is that of *process acquisition*. The SMART approach assumes that a process definition exists that can be encoded in at least a tabular form with activities, their inputs, outputs, and resource requirements. However, in real situations, the issues of which activities to include in the model, and at what level of granularity, are very complicated. This is compounded by the fact that different participants of the process have different viewpoints on the process. Organization design theorists have developed some manual techniques to address this issue, for example see [21]. We need to incorporate such techniques within frameworks like SMART to more fully support the process engineering life-cycle. Thus, this represents an important area for future research.

Acknowledgments

We thank Martin Griss and Kevin Wentzel of HP Labs for their support of this project. We thank Ralph Hyver and Joe O'Brien from HP's University Affairs department who have significantly contributed in simplifying the logistics of our collaboration and providing support for it. We thank Tom Christian, Dave Pugmire, and Chung Tung of SESD for their support of this project.

References

[1] S. Bandinelli and A. Fuggetta. Computational Reflection in Software Process Modeling: the SLANG Approach. In *Proceedings of the* 15^{th} *International Conference on Software Engineering*, pages 142–144–153, Baltimore, MD, May 1993. IEEE Computer Society.

[2] S. Bandinelli, A. Fuggetta, C. Ghezzi, and A. Morzenti. A Multi-Paradigm Petri Net Based Approach to Process Description. In *Proceedings of the* 7^{th} *International Software Process Workshop*, Yountville, CA, October 1991. IEEE Computer Society Press.

[3] V. R. Basili and H. D. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*, 14(6):758–773, June 1988.

[4] S. R. Bourne. The UNIX Shell. *The Bell System Technical Journal*, 57(6):1971–1990, July-August 1978.

[5] G. M. Clemm. The Workshop System: A Practical Knowledge-Based Software Environment. In *Proceedings of the 3rd ACM software engineering environments conference*, pages 55–64, December 1988.

[6] T. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, Cambridge, MA, 1993.

[7] W. Deiters and V. Gruhn. Managing Software Processes in the Environment MELMAC. In *Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 193–205, 1994.

[8] W. Deiters, V. Gruhn, and W. Schäfer. Process Programming: A Structured Multi-Paradigm Approach Could be Achieved. In *Proceedings of the* 5^{th} *International Software Process Workshop*. IEEE Computer Society Press, September 1989.

[9] C. Fernstrom. Process WEAVER: Adding Process Support to Unix. In 2^{nd} *International Conference on Software Process*, pages 12–26, Berlin, Germany, 1993. IEEE Computer Society.

[10] P. K. Garg, T. Pham, B. Beach, A. Deshpande, A. Ishizaki, K. Wentzel, and W. Fong. Matisse: A Knowledge-based Team Programming Environment. Technical Report HPL-92-104, Hewlett-Packard Company Labs, Palo Alto, August 1992. To appear in *International Journal of Software Engineering and Knowledge Engineering*.

[11] P. K. Garg and W. Scacchi. A Hypertext System to Manage Software Life Cycle Documents. *IEEE Software*, pages 90–98, May 1990.

[12] V. Gruhn. *Validation and Verification of Software Process Models.* PhD thesis, University of Dortmund, Germany, 1991.

[13] Hewlett-Packard Company, Palo Alto, CA. *Developing SynerVision Processes*, May 1993. Part number: B3261-90003.

[14] A. S. Karrer and W. Scacchi. Meta-Environments for Software Production. *International Journal on Software Engineering and Knowledge Engineering*, 3(1):139–162, 1993.

[15] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. D. Rombach. Software Process Modeling Example Problem. In *Proceedings of the* 6^{th} *International Software Process Workshop*, Hakodate, Hokkaido, Japan, October 1990.

[16] Rob Kling and Walt Scacchi. The Web of Computing: Computing Technology as Social Organization. In M. Yovits, editor, *Advances in Computers*, volume 21, pages 1–90. Academic Press, Inc., 1982.

[17] N. H. Madhavji, V. Gruhn, W. Dieters, and W. Schäfer. PRISM = Methodlogy + Process-Oriented Environment. In *Proceedings of the* 12^{th} *International Conference on Software Engineering*, pages 277–289, March 1990.

[18] P. Mi and W. Scacchi. A Knowledge Base Environment for Modeling and Simulating Software Engineering Processes. *IEEE Trans. Knowledge and Data Engineering*, 2(3):283–294, 1990.

[19] P. Mi and W. Scacchi. Modeling Articulation Work in Software Engineering Processes. In 1^{st} *International Conference on Software Process*, pages 188–201, Los Angeles, CA, 1991. IEEE Computer Society.

[20] Leon Osterweil. Software Processes are Software too. In *Proceedings of the* 9^{th} *International Conference on Software Engineering*, pages 2–13, April 1987.

[21] G. A. Rummler and A. P. Brache. *Improving Performance: How to Manage the White Space on the Organization Chart.* Josey-Bass Publishers, San Francisco, 1990.

[22] L. G. Votta Jr. Comparing One Formal to Informal Process Description. In W. Schäfer, editor, *Proceedings of the* 8^{th} *International Software Process Workshop*, Wadern, Germany, March 1993. IEEE Computer Society Press.