MODELING, INTEGRATING, AND ENACTING THE DESIGN OF SOFTWARE PRODUCTION PROCESSES

Walt Scacchi
Information and Operations Management Dept.
University of Southern California
Los Angeles, CA 90089-1421 USA
(213) 740-4782, (213) 740-8494 (FAX)
scacchi@gilligan.usc.edu

ABSTRACT

I describe an approach and mechanisms to support the design and engineering of software production processes throughout their life cycle. I describe what activities are included in the process life cycle. I then describe our approach, computational mechanisms, and experiences in supporting many of these life cycle activities.

1. Introduction

Software process redesign, enterprise integration, and coordinated teamwork support are among the current generic goals for advanced software process engineering (SPE) technology within organizations. Organizations are looking for ways to respond to competitive pressures by redesigning and continuously improving their software system production and operation processes. Such endeavors may therefore address complex organizational processes that entail tens, hundreds, or even thousands of organizational participants, as well as support the integration of a heterogeneous collections of SPE tools and application systems, whether new or pre-existing. Thus, we are faced with the problem of how to realize these goals in a coherent, scalable, and evolutionary manner.

In this paper, I describe an approach and supporting mechanisms I have been investigating in an effort to solve this problem and realize these goals. As such, I describe an approach to engineering complex software production processes throughout their life cycle. Accordingly, I describe some of the SPE technologies we have developed within the System Factory Project at USC [16].

2. Supporting the Software Process Engineering Life Cycle

In simplest terms, we see that support for software processes entails more than the modeling of processes and creation of executable process programs. Our view is that the goal should be to support software process engineering across the process life cycle. Much like the way that the development of complex software systems entails more than programming, so does 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 at USC has led to the initial formulation of a software process life cycle that is founded on the incremental development, iterative refinement, and

ongoing evolution of software process descriptions. In this way, the software process life cycle spiral includes activities that address process:

- meta-modeling: constructing and refining a process concept vocabulary and logic (an ontology) for representing families of processes and process instances in terms of object classes, attributes, relations, constraints, control flow, rules, and computational methods.

- modeling: eliciting and capturing of informal process descriptions, and their conversion into formal process models or process model instances.

- analysis: evaluating static and dynamic properties of a process model, including its consistency, completeness, internal correctness, traceability, as well as other semantic checks.

- simulation: symbolically enacting 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.

- visualization: providing users with graphic views of process models and instances that can be viewed, navigationally traversed, interactively editted, and animated to convey process statics and dynamics.

- prototyping: incrementally enacting partially specified process model instances in order to evaluate process presentation scenarios to end users, prior to performing tool and data integration.

- administration: assigning and scheduling specified users, tools, and development data objects to modeled user roles, product milestones, and development schedule.

- integration: encapsulating or wrapping selected tools, repositories, and data objects that are to be invoked or manipulated when enacting a software process instance.

- environment generation: automatically transforming a process model or process instance specification into a process program producing a process-based environment that selectively presents integrated tools/objects to end-users for enactment.

- enactment: executing a generated process program and resulting environment by a process engine that guides or enforces specified users or user roles to enact the process as planned.

- monitoring, recording, and auditing: collecting and measuring process enactment data needed to improve subsequent process enactment iterations, as well as documenting what process steps actually occurred in what order.

- replay: graphically simulating the re-enactment of a process, in order to more readily observe process state transitions or to intuitively detect possible process enactment anomalies.

- articulation: diagnosing, repairing, and rescheduling actual or simulated process enactments that have unexpectedly broken down due to some unmet process resource requirement, contention, availability, or other resource failure.

- evolution: incrementally and iteratively enhancing, restructuring, tuning, migrating, or reengineering process models and process life cycle activities to more effectively meet emerging user requirements, and to capitalize on opportunitistic benefits associated with new tools and techniques.

While such a list might suggest that engineering a software process through its life cycle proceeds in a linear or "waterfall-like" manner, this is merely a consequence of its narrative presentation. In practical situations where these activities and associated process mechanisms have been initially tried out (e.g, at AT&T Bell Laboratories [18], Northrop Corporation, and elsewhere [17]), it quickly becomes clear that software process engineering is a dynamic team-based endeavor that can only lead to mature processes through rapid process prototyping, spiraling development, and the reengineering of ad hoc process task instances and models. To no surprise, many of our efforts addressing these life cycle activities and supporting prototype mechanisms have been described in greater detail elsewhere [11, 9, 12, 7, 8, 17]. As such, I now turn to briefly describe our approach to some of these activities.

3. Modeling, Analysis and Simulation

We have developed a knowledge-based computing environment, called the Articulator, for modeling, analyzing, and simulating complex organizational processes [11]. This environment utilizes an object-oriented knowledge representation scheme for modeling interrelated classes of organizational resources. These classes in turn characterize the attributes, relations, rules, and computational methods associated with a taxonomy of organizational resources. Thus, using the Articulator, we can construct or prototype knowledge-based models of organizational processes.

3.1. Modeling

The resource taxonomy we have constructed, explained in detail elsewhere [4, 11, 13], serves as a process meta-model which provides an ontological framework and vocabulary for constructing software process models (SPMs) [2]. In simplest terms, our process meta-model states that software processes (or other organizational processes) can be modeled in terms of (subclasses of) agents that perform tasks using tools and systems which consume or produce resources. Further, agents, tools, and tasks are resources, 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 workstations that run Motif 1.2, Emacs, and FrameMaker software in order to get the reports produced by the end of the week. For a more complex SPM, such as one based on MIL-STD-2167A, then a multi-level process task decomposition must be specified (see [10] for examples).

Although space limits a more detailed presentation here, the agents, tasks, product resources, tools, and systems are all hierarchically decomposed into subclasses that inherit the characteristics of their parent classes, for economy in representation. Further, these resource classes and subclasses are interrelated in order to express relationships such as precedence among tasks (which may be sequential, iterative, conditional, optional, or concurrent), task/resource pre- and post-conditions, authority relationships among agents in different roles, product compositions, tool/system aggregations, and others [11, 13]. Thus, in using these classes of process modeling entities, we are naturally led to model software production 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 composed products/artifacts [6].

In addition, the meta-model enables us to model other complex phenomena associated with organizational processes, such as agents' resource sovereignties (ie, the set of resources under the control of an agent), authority asymmetries (ie, relationships among agents), multiple belief systems, articulation strategies [9, 7], etc. Accordingly, these relationships are defined in the meta-model, used and then instantiated in the SPMs. Then, we use the Articulator to query, analyze, and simulate modeled processes as described below.

3.2. 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 [11]. These functions represent batched or interactive queries to the knowledge base through its representational schemata. At present, we have defined a few dozen paramaterized 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 [11]. Further, most of these analysis functions incorporate routines for generating different types of reports (e.g., raw, filtered, abstracted, paraphrased, publication format) which can be viewed interactively or incorporated into desktop publication documents.

3.3. Simulation

Finally, since process models in our scheme are symbolic descriptions, we can simulate them using the Articulator. In simple terms, this is equivalent to saying that 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. Since tasks and other resources can be modeled at arbitrary levels of precision and detail, then the simulation makes progress as long as task pre-conditions 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 organizational processes, although most are targeted to the domain of large-scale software production processes. In this regard, we have constructed SPMs and instances for organizations within our research sponsors' business units, including those involving team-based software product design and review processes, as well as department and division-wide software production and support processes that include tens to hundreds of participants. Such SPMs typically include dozens of classes of agents, tasks, resources, and products, but a small number of software tools and information systems, while the SPM instantiation may include 1-10+ instances of each class. Our experience to date suggests that modeling existing processes can take 0.5 to 3+ person-months of effort, analysis routines can run in real-time or acceptable near-real-time, while simulations can take seconds to hours (even days!) depending on the complexity of the SPM, its instance space, and the amount of non-deterministic process activities being modeled. Note however that simulation performance is limited to available processing power and processor memory, thus suggesting better performance can be achieved with (clusters of) high performance computing platforms.

4. Visualization, Prototyping, and Enactment

As we improve our ability to construct and redesign plausible models of different organizational processes, we have found that it is increasingly important to be able to quickly and conveniently understand the structure and dynamics of complex SPM instances. As such, we have developed a graphic user interface (GUI) for visualizing and animating SPM instances. This process-based user interface (PBI) is coupled to a another computational facility to which SPM instances developed with the Articulator can be automatically transformed into process programs, then downloaded into a process driver. In turn, the process driver and GUI enable SPM developers to prototype or enact process-driven software development environments. These capabilities can be used to reflect, guide, try-out, and support how users work with process-driven environments. These capabilities are described next.

4.1. Visualization

PBI provides graphic visualizations of task precedence structure on a role-specific basis for each user (ie, agent instance) [12]. Since process tasks can be modeled and hierarchically decomposed into subtasks of arbitrary depths, then PBI provides users with a subtask window and an associated (cached) workspace. Since a subtask precedence structure appears as a directed graph, we associate a development "status" value (ie, none, allocated, ready, active, broken, blocked, stopped, finished) with each subtask step. For ease of understanding, these status values are represented in the PBI as colors, so

that the current "state" of a process task can be observed as a color pattern in the directed graph. Further, as PBI also incorporates a facility for recording and replaying all changes in process task state, evolving process state histories can be maintained and visualized as an animation of changing task step status colors. Subsequently, we have found that project managers in industrial organizations can quickly browse such a PBI display to ascertain the current status of an arbitrarily complex production process to varying degress of detail. The interested reader should consult [12] to see a number of examples.

4.2. Prototyping

The process driver that backs PBI can also accept an SPM as its input. Since SPMs need not include instance details, then it is possible to use these SPMs to create prototype mock-ups of process-driven environments. These prototypes show users the look-and-feel of how the emerging process-driven environment would appear. That is, the SPM serves to provide role-specific views of process task precedence structure, which in turn guides users in their use of tools, systems, and data resources. Thus, since the Articulator accomodates partially decomposed SPMs, then these SPMs can also be downloaded into the process driver to visually display and interactively walkthrough role-specific usage scenarios. We find this extremely useful in supporting an SPM construction effort that is iterative, incremental, and improvement-oriented in an evolutionary sense. Further, this prototyping capability can also be used to support training situations, which is especially important when introducing new users to the concepts and mechanisms that support process-driven software environments.

4.3. Integration and Enactment

The process driver and PBI provides software tools, application systems and associated data resources (e.g., objects, files, databases, spreadsheets) which are served to users at the bottom level subtask actions so that they can perform their work. This process enactment capability enables users to perform or enact the modeled process tasks, subtasks, or actions assigned to them. Accordingly, during process enactment, Software tools, information systems, and data resources are delivered to them at their displays and fingertips when needed. Examples of process enactment views which provides the tools, data objects, and internal workspace appropriate for different developer roles appear elsewhere [12].

Process enactment is a computational activity. It interprets an SPM or SPM instance as its input. The SPM or instance output from the Articulator represents a process enactment specification that is automatically transformed into an object-oriented OS shell scripting language, which serves as our process programming language [15]. In this sense, our process programs are produced by a special-purpose application generator [5]. Accordingly, the process enactment specification can incorporate any OS command, system invocation script, virtual mouse selections, or canned user input, as well as access protocols to distributed data/object repositories and the data models therein [14]. Tool integration is supported using encapsulation wrappers compatible with both the process modeling notation and the generated process programs. Thus, through process integration [12], is possible for users to

perform complex information processing tasks through a process-based interface that integrates access to local/networked data resources and tools/systems through a common GUI presentation [16].

Finally, our experience in supporting or enforcing process guidance has led to the following result. We find that different developers want different levels of process guidance, ranging from "none to almost none" by expert users, to "complete conformance" by project managers uncertain about their development staff's ability. Accordingly, we elected to provide what we call a guidance policy enforcement mode variable which allows varying degrees of process guidance/conformance to be turned on, off, or bypassed depending on the role, sub-task, or action. The choices for what levels of guidance are needed are then made by the developers' team, rather than by us. However, when enforcement is turned down or off, more forms of process breakdown are possible, but some forms of breakdown recovery and repair (i.e., process articulation) are enabled as well [9, 7]. Thus, our recommendation at this time is that if the eventual goal of process enactment is to increase process maturity and effectiveness, then it is better to keep process guidance turned on, provide tractable bypasses for expert users, and employ process articulation mechanisms for tracking and assimilating excursions outside of modeled process.

5. Other Advanced SPE Technologies

In addition to the the computational mechanisms described so far, our approach utilizes mechanisms not described here. These include mechanisms for

- process scheduling and administration [7],
- diagnosing, replanning, and rescheduling processes that unexpectedly breakdown or fail [9, 7],
- software re-engineering processes and environment [1], and
- knowledge-based process model repository [10]

Thus, our approach allows us to construct and demonstrate a computational framework for modeling, enacting, and integrating team-oriented process-driven work environments for redesigned software development organizations. As such, we are now working to prototype and demonstrate a small number of process-driven environments in different industrial application domains that incorporate commercial off-the-shelf software tools, internally developed systems, and prototype research mechanisms, all operating on Unix workstations over local-area and wide-area networks.

Although not shown here, we have been able to demonstrate supporting mechanisms for the process life cycle activities described above. 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 (order fulfillment, new product development, business planning), albeit in a radically innovative way [3].

6. Conclusion

This report provides a brief introduction to our approach and computational mechanisms to modeling, enacting, and integrating organizational processes that involve SE tools, systems, and data resources. These include a knowledge-based environment for re-engineering complex organization processes, and other facilities for realizing and executing these processes. We are using our results to help redesign existing organizational processes that employ large teams, and provide a coherent, scalable SE infrastructure for enacting and integrating advanced software processes.

7. References

1. Choi, S.C. and W. Scacchi. "SOFTMAN: AN Environment for Forward and Reverse Computer-Aided Software Engineering". Information and Software Technology 33, 9 (November 1991), 664-674.

2. Curtis, B., M.I. Kellner, and J. Over. "Process Modeling". Communications ACM 35, 9 (1992), 75-90.

3. Davenport, T. Process Innovation: Reengineering Work through Information Technology. Harvard Business School Process, Cambridge, MA, 1993.

4. Garg, P.K. and W. Scacchi. "On Designing Intelligent Software Hypertext Systems". IEEE Expert 4 (1989), 52-63.

5. Karrer, A. and W. Scacchi. "Meta-Environments for Software Production". Intern. J. Soft. Engr. and Know. Engr. 3, 1 (March 1993), 139-162.

6. Kling, R. and W. Scacchi. "The Web of Computing: Computer Technology as Social Organization". Advances in Computers 21 (1982), 1-90. Academic Press, New York.

7. Mi, P. Modeling and Analyzing the Software Process and Process Breakdowns. unpublished dissertation, Computer Science Dept., USC, Los Angeles, CA, 1992.

8. Mi, P. and W. Scacchi. "Evolution of a Software Meta-Process: A Case Study". (submitted for publication) , (October 1992).

9. Mi, P. and W. Scacchi. Modeling Articulation Work in Software Engineering Processes. Proc. First Intern. Conf. Soft. Process, IEEE Computer Society, 1991, pp. 188-201.

10. Mi, P., M-J. Lee, and W. Scacchi. A Knowledge-Based Library for Process-Driven Software Development. Proc. 7th. Annual Conf. Knowledge-Based Software Engineering, IEEE Computer Society, 1992, pp. 122-131.

11. Mi, P. and W. Scacchi. "A Knowledge Base Environment for Modeling and Simulating Software Engineering Processes". IEEE Trans. Knowledge and Data Engineering 2, 3 (1990), 283-294.

12. Mi, P. and W. Scacchi. "Process Integration for CASE Environments". IEEE Software 9, 2 (March 1992), 45-53.

13. Mi, P. and W. Scacchi. "The Articulator Meta-Model". IOM Technical Report 93-07 September, (1993), .

14. Noll, J. and W. Scacchi. "Integrating Diverse Information Repositories: A Distributed Hypertext Approach". Computer 24, 12 (1991), 38-45.

15. Osterweil, L. Software Processes are Software Too. Proc. 9th. Intern. Conf. Software Engineering, IEEE Computer Society, April, 1987, pp. 2-13.

16. Scacchi, W. "A Software Infrastructure for a Distributed System Factory". Software Engineering Journal 6, 5 (September 1991), 355-369.

17. Scacchi, W. and P. Mi. Experiences with Process Modeling, Analysis, and Simulation of Formalized Process Models. position paper circulated at the 8th. Intern. Soft. Process Work., Dagstuhl, Germany, February, 1993, pp. .

18. Votta, L. Comparing One Formal to One Informal Process Description. position paper circulated at the 8th. Intern. Soft. Process Work., Dagstuhl, Germany, February, 1993.