ELSEVIER

The Journal of Systems and Software 46 (1999) 183–192

The Journal of Systems and Software

Experience with software process simulation and modeling

Walt Scacchi

USC ATRIUM Laboratory, 2912 Broad St., Newport Beach, CA 92663-4832, USA

Received 10 November 1998; accepted 11 November 1998

Abstract

In this paper, I describe an approach and experiences in developing and applying simulation and modeling technologies to software processes. Processes for both software development and use have been investigated. The focus of this paper is organized around three topics for software process simulation and modeling. First, I describe an approach and examples of software simulation and modeling as investigated with knowledge-based process engineering environment developed at USC. Second, I describe how by focusing on process modeling, analysis and simulation, we are led to expand the scope of work with software processes toward a more comprehensive software process life cycle engineering. Third, I describe some of the lessons learned from applying modeling and simulation concepts, techniques and tools to software processes in a variety of organizational settings. Conclusions then stress the complementary value arising form the use of both qualitative and quantitative technologies for software process simulation and modeling.

Keywords: Software process; Process modeling; Process simulation; Knowledge-based simulation; Process life cycle

1. Introduction

Over the past 10 years, research efforts in the USC System Factory project and the USC ATRIUM Laboratory have investigated various kinds of process modeling and simulation issues. These efforts focus on understanding the organizational processes involved in software system development and use. Many of these efforts have addressed how to use simulation and modeling tools, techniques and concepts to understand and forecast software process trajectories prior to their actual performance. This is done when analyzing both "as-is" processes (i.e., simulated models of the software processes found in some organizational setting) and "to-be" processes (simulated models of new or redesigned software processes intended to operate in the same organizational setting). However, we have also used these same technologies to understand, validate and refine models of software processes in light of empirically grounded observational data from actual process performances that we have collected. The potential also exists for applying these techniques to model and simulate the "here-to-there" processes and transformations that occur in moving an organization from as-is to to-be software processes. This paper highlights the approach, experiences and lessons learned through these efforts.

2. Simulation and modeling software development processes

Models of software processes are particularly interesting when formalized as computational descriptions (Curtis et al., 1992). Accordingly, we can model, interactively browse and symbolically execute them. In simple terms, this is equivalent to saying that simulation entails the symbolic performance of process tasks by their assigned agent using the tools, systems and resources to produce the designated products. For example, in simulating an instance of a software project management process, the manager's agent would "execute" her management tasks. Tasks are modeled according to the precedence structure of sub-tasks or steps specified in the modeled process instance. Agents and tasks can then use or consume simulated time, budgeted funds and other resources 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 preconditions or post-conditions are satisfied at each step. For example, for a manager 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 and then waits for new input or command from the simulation user.

0164-1212/99/$ – see front matter
PII: S0164-1212(99)00011-4

In our work at USC, we have employed two kinds of simulation technologies in studies of software development processes, knowledge-based simulation (KBS) and discrete-event simulation (DES) (Mi and Scacchi, 1990; Scacchi and Mi, 1997). We have built our own KBS tools starting in 1988, while we employed a commercially available DES package. Each is described in turn. Commercial KBS products integrated with DES are now available from companies such as Knowledge Based Systems (www.kbsi.com) and Intelligent Systems Technology (www.intelsystech.com). We have also developed and used tools that allow users to browse, stepwise simulate (traverse) and execute software process models across distributed network environments, as described elsewhere (Noll and Scacchi, 1998; Scacchi and Mi, 1997; Scacchi and Noll, 1997).

2.1. Knowledge-based simulation of software processes

KBS is one kind of technology used to simulate software processes. Many approaches to modeling and simulating complex processes center around the execution of an abstract finite state machine that traverses explicitly or implicitly modeled states and/or events in the simulated process instance. For example, the entity-process approach developed by Kellner (Humphrey and Kellner, 1989) is one that uses the Statecharts[tm] tool to explicitly model and traverse the states declared in a software process.

Simulations also typically apply to instances of a modeled process, where the model is developed then run through the simulation after its parameter values are instantiated. In a KBS, software process states can either be explicitly declared or implicitly represented. Implicit representation means that process states will correspond to values that populate a snapshot of the underlying knowledge-base of interlinked objects, attributes and relations used to record and manage the simulation. KBS can also symbolically evaluate a class of process models, or a process model without instance values. Finally, KBS employ computational reasoning in the form of pattern-directed inferencing that is implemented via a production rule base (Mi and Scacchi, 1990). These rules monitor and update the values of objects, attributes or relations stored in the knowledge-base. The set of all values in the knowledge-base constitutes the implicit state of a process at a given moment. When one or more rules is enabled and fires, then the computational procedure in the rule's action is performed and the state of the process knowledge-based is updated. Thus, these features help to begin to differentiate KBS approaches to software process simulation.

KBS is useful to address different types of simulated process execution behavior. We have found four types of process behavior of interest. First, understanding software processes requiring fine granularity. Second, analyzing processes with multiple or mixed levels of process granularity. Third, analyzing patterns of interaction and work flow among software developers (or agents). Fourth, analyzing processes whose structure and control flow are dynamically changing (Mi and Scacchi, 1993). Granularity here refers to the level of detail that we seek to simulate in order to understand gross or fine-level process dynamics or details. The following two figures may help to convey this. Alternatively, we use DES to simulate processes at a coarser level, but when our interest is addressed to understanding the overall performance envelope that results from the simulated executed of a real-world process that is instantiated 5–100+ times. This is described later.

Fig. 1 provides a view of a work station display running a KBS session in the Articulator environment developed at USC (Mi and Scacchi, 1990). The Articulator was implemented as a rule-based multi-agent problem solving system that models, analyzes and simulates complex organizational processes. The top left frame lists the KBS production rules that have fired during the process simulation step. Many of these rules pertain to the operation of the KBS engine, rather than to a simulated software process. The details of these rules are not the focus here, but examples can be found elsewhere (Mi and Scacchi, 1990). The top right frame provides an annotated transcription of the software process events that have occurred as a result of the rule firings. The bottom right frame then provides a description of rules that conflict when concurrently fired, and the resulting conflict resolution results that occur.

Fig. 2 provides another screen display this time focusing on the KBS annotation transcript that is partially occluded in Fig. 1. Furthermore, additional post-processing has been performed by the Articulator to paraphrase the transcript into a more readable, though somewhat stilted form. Classic techniques for paraphrasing the internal semantic form into a sentential form are employed. As before, details on the software process being simulated and automatically documented as shown in Fig. 2 appear elsewhere (Mi and Scacchi, 1990).

Our use of KBS mechanisms support features that are uncommon in popular commercial simulation packages. The provision of paraphrasing capabilities described above is an example. Other examples include using symbolic execution functions to determine the path and flow of intermediate process state transitions in ways that can be made persistent, queried, dynamically analyzed and reconfigured into multiple alternative scenarios. Similarly, the use of a multi-agent problem-solving representation allows use of a distributed environment to model and simulate the interactions and behavior of different agents, skill sets and task assignments. These capabilities for knowledge-based software simulation are described next.

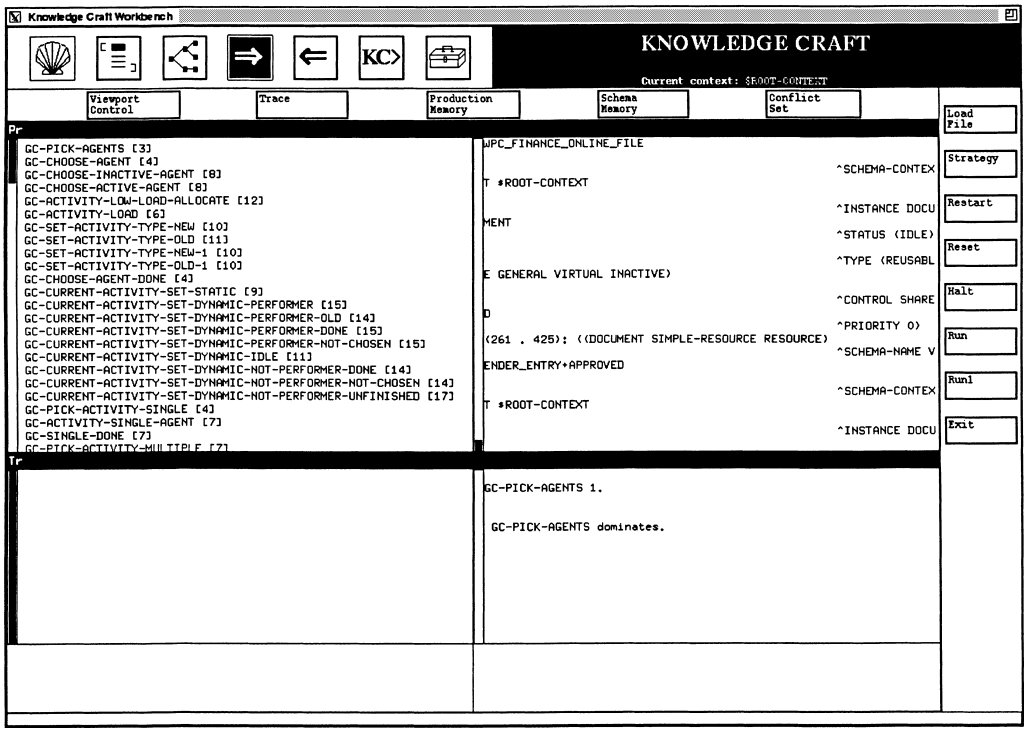

Fig. 1. Screen display of a KBS session running in the articulator environment.

Persistent storage of simulated events enables the ability to run a simulation forward and backward to any specific event or update to the knowledge-base. Persistence in the Articulator environment was implemented using object oriented data management facilities provided in the underlying knowledge engineering environment (Mi and Scacchi, 1990). Queries to this knowledge-base provide one mechanism to retrieve or deduce, where an event or update of interest occurs. Dynamic analysis can monitor usage or consumption of resources to help identify possible bottlenecks in simulated process instances. Using these functions, it is possible to run a simulation to some point, back up to some previous context, and then create new instance values for the simulated process model. These changes then spawn a new simulated process instance thread. For example, this could allow adding more time to a schedule, more staff to an overloaded work flow, or to remove unspent money from a budget. Such a change would then branch-off a new simulation trajectory that can subsequently be made persistent, and so forth. Furthermore, we can also employ the paraphrasing and report generation functions to produce narrative-like descriptions or summaries of what transpired during a given simulation run. Thus, knowledge-based simulation enables the creation and incremental evolution of a network of event trajectories. These in turn may be useful in evaluating or systemically forecasting the yield attributable to new, interactively designed process alternatives.

The Articulator environment was among the earliest to address issues of modeling and simulating organizational processes enacted by autonomous or managed problem-solving agents (Mi and Scacchi, 1990). Although agent technology was still at a conceptual stage when the Articulator was initially developed (1988–1990), we thought it would be interesting to develop a process simulation capability that could eventually be distributed and run across a local-area or wide-area network. Individually modeled agents acted as placeholders for people working in real-world processes. Agents were capable of individual or collective problem-solving behavior, depending on what problem-solving skills (i.e., processes they "know" how to perform) they were given, and how they were configured to interact (typically to send and receive messages from one another). We found that developing and specifying software processes from an agent-centered perspective helped us to more thoroughly understand the processes under examination.

Soon after, we developed an approach to import and export specifications of agent-centered software process models and instances to integrate and control new/legacy software engineering environments with open system interfaces (Mi and Scacchi, 1992). Thus, we could now prototype and later generate process-driven software interfaces (Mi and Scacchi, 1992). Thus, we could now prototype and later generate process-driven software development or use environments that could be tailored for different users or user roles (Mi and Scacchi,

Fig. 2. Paraphrased transcription resulting from a KBS run on a fine-grain software process.

1992; Garg et al., 1994; Scacchi and Mi, 1997). Similarly, we could monitor, capture, replay and simulate the history of a process instance that was enacted in a process-driven work environment (Scacchi and Mi, 1997). Unfortunately, interest in these matters quickly began to overshadow our interest in just simulating software processes using rule-based problem-solving computational agents. Subsequently, our interest in developing an agent-based simulation environment waned as others interested and exclusively focused on agent technology, rather than software processes, moved into the foreground.

2.2. Discrete-event simulation of software processes

DES allows us to dynamically analyze different samples of parameter values in software process instances. This enables simulated processes to function like transportation networks whose volumetric flow, traffic density and congestion bottlenecks can be assessed according to alternative (heuristic or statistical) arrival rates and service intervals. Using DES, process experts often find it easy to observe or discover process bottlenecks and optimization alternatives. Similarly, when repetitive, high frequency processes are being studied, then DES provides a basis for assessing and validating the replicability of the simulated process to actual experience. Likewise, DES can be used when data on events and process step completion times/costs can be empirically measured or captured, then entered as process instance values for simulation. Thus, DES seems well-suited for use when studying the behavioral dynamics of processes with a large number of recurring process instances and those with relatively short process completion (or cycle) times.

Many commercially available DES packages now support animated visual displays of simulated process executions or results. We use the animated displays so that process experts can further validate as-is and to-be process simulations under different scenarios. These can be viewed as software development or business process "movies". In addition, the animated process instance

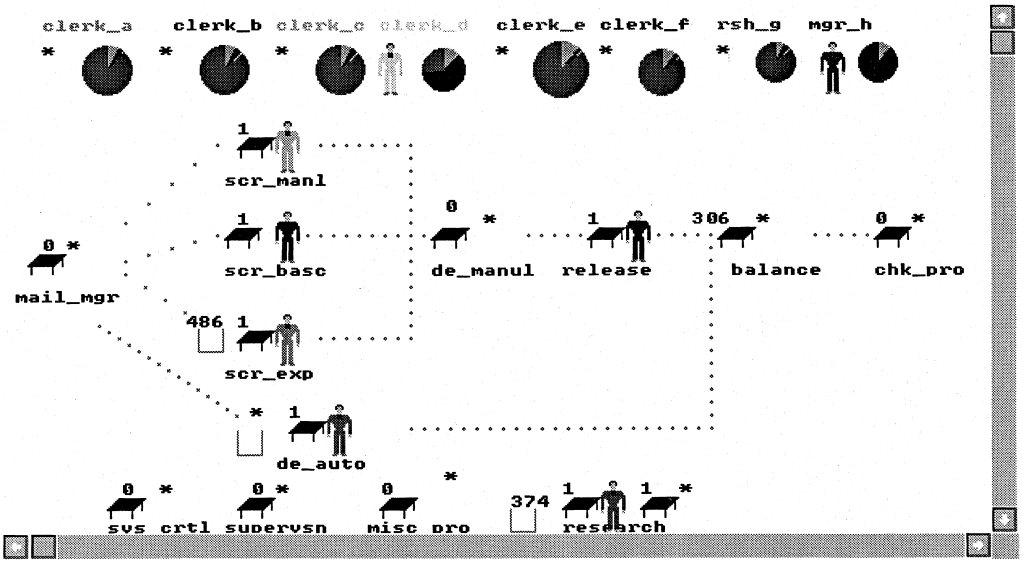

Fig. 3. Visual display form an animated multi-agent simulation.

simulations can be modified, re-executed and viewed like other simulations. Although we cannot conveniently show such animations in printed form, snapshots captured from such a simulation may suggest what can be observed. In Fig. 3, we have modeled an eight-person (or agent) activity for performing an instance of a high frequency end-user process supported by a software system (in this case, for an accounts payable process). The figure depicts the structure of the work flow, agents that perform each task, individual workload pie charts and work units-in-progress quantities (e.g., the backlog of invoices cleared, problematic invoices and checks released).

We have employed both KBS and DES to generate and analyze simulated process performance results. With KBS our attention focused on generating persistent records of the process execution context as it evolved through the course of a simulation. Fig. 2 displays an output format view of such a record. By capturing simulation results in this manner, they could be stored in a database repository, queried and even run backwards. Such capability enabled us to observe or measure the dynamics of interactions between the multiple agents (e.g., software development people working in different roles in a process) over some simulation duration or event stream. In contrast, with DES our interest was often focused on developing answers to questions pertaining to descriptive statistical characterization of simulated process execution events. In this regard, we found that DES could be used to analyze and measure the distribution of time, effort and cost (as in "activity-based costing" and "process-based costing").

Finally, we experimented with different techniques for integrating the KBS and DES software tools. Using the Articulator, we built a transformation utility that could generate a process model specification in the standard API format recognized by the DES. Commercial KBS products now support this feature. Using the output of the DES as input to the KBS is similarly achieved with a different data transformation utility that rewrites selected DES statistics outputs as input values to KBS simulation parameters. Shell scripts are then used to hook the programs and utilities together.

Thus, both KBS and DES play vital roles in helping to understand the behavioral dynamics of software processes that we find in real-world settings.

3. Simulation and modeling in process life cycle engineering

Following the discussion in the preceding section, it becomes increasingly clear that modeling and simulating software processes is not an end onto itself. To the contrary, experience in software process modeling and simulation has led to a better understanding the range of activities that can be instigated and supported under the heading of *process life cycle engineering* (Garg et al., 1994; Heineman et al., 1994; Scacchi and Mi, 1997). To help make this clear, consider the following set of activities that emerged as a result of our experience with knowledge-based process engineering (Garg et al., 1994; Mi and Scacchi, 1990, 1992, 1993, 1996; Scacchi and Mi, 1997; Noll and Scacchi, 1998). In addition, comparisons are provided to highlight differences in the use of knowledge-based approaches in contrast to how discrete-event and system dynamics (SD) approaches for each process life cycle activity.

Meta-modeling: Constructing and refining a process concept vocabulary and logic for representing families of

processes and process instances in terms of object classes, attributes, relations, constraints, control flow, rules and computational methods. A knowledge-based software process meta-model was developed for this purpose (Garg and Scacchi, 1989; Mi and Scacchi, 1990, 1996). In contrast, DES packages employ a statistical network flow meta-model, while a SD approach employs a continuous system of differential equations as its meta-model. A meta-model conceived and implemented with software processes as the modeling goal minimizes the representational compromises that arise when using, adapting or overloading statistical or numerical schemes. The process meta-model supported by the Articulator is an open representation by design that serves as a framework for integrating and interoperating with other software technologies (Scacchi and Mi, 1997). Its meta-model and modeling ontology can be extended as needed, whereas popular DES and SD packages employ closed representations within proprietary systems.

Modeling: Eliciting and capturing informal process descriptions, then converting them into formal process models or process model instances. Knowledge-based software process models can be incrementally specified and partially evaluated (Mi and Scacchi, 1990). In contrast, popular DES and SD packages require that software process models must be fully specified before they can be evaluated.

Analysis: Evaluating static and dynamic properties of a process model, including its consistency, completeness, internal correctness, traceability, as well as other semantic checks (Mi and Scacchi, 1990). Analysis also addresses the feasibility assessment and optimization of alternative process models. Analysis should be applicable to both classes and instances of process models, as is supported by the articulator. In contrast, available DES and SD packages used for modeling and simulating software processes do not discuss whether the available processing mechanisms can assess the consistency, completeness or correctness of process models built in their native representation schemes.

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. Using knowledge-based simulation it is possible to support incremental simulation, persistence of simulation execution traces, reversible simulation computation, query-driven simulation and reconfigurable simulation space during execution (Mi and Scacchi, 1990). In contrast, common DES and SD packages only support monolithic simulation and do not offer support for persistence, reversible computation or query processing. However, these are not necessarily systematic shortcomings of DES or SD approaches, but rather they are shortfalls in available implementations.

Redesign: Reorganizing and transforming the structure of relationships within a process to compress completion time, as well as reduce or eliminate the number of steps, handoffs, or participants. Knowledge-based process redesign can support automated transformations or optimizations on internal semantic representations of process models (cf. Scacchi and Mi, 1997; Scacchi and Noll, 1997). No explicit support is provided by DES or SD packages for automated process redesign, thus process redesign is a manual activity with these packages.

Visualization: Providing users with graphic views of process models and instances that can be viewed, navigationally traversed, interactively edited and animated to convey an intuitive understanding of process statics and dynamics. Popular DES and SD packages provide mature facilities for creating and displaying graphic or animated views of software processes. Similarly, they provide mechanisms for creating user-defined plots and charts that display characteristics of various event or state distributions. However, improvements are needed in a variety of areas including automated layout of alternative views of process models, graphical query, highlighting of user-defined properties and layout (e.g., highlighting the critical path during execution) and visualizing large-scale process representations (cf. Fig. 4).

Prototyping, walk-through and performance support: Incrementally enacting partially specified process model

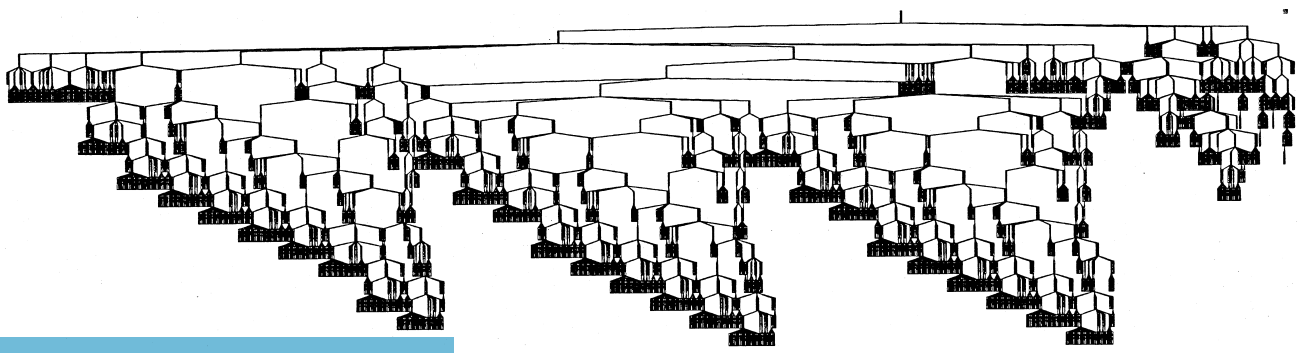

Fig. 4. Decomposition view of a large software life cycle process model.

instances in order to evaluate process presentation scenarios through the involvement of end users prior to performing tool and data integration. These capabilities are easily provided with an open knowledge-based process engineering environment, as well as providing mechanisms to access and use these capabilities over the Internet and Web (Scacchi and Mi, 1997; Scacchi and Noll, 1997). None of the DES or SD packages that have been used for software processes readily support these capabilities.

Administration: Assigning and scheduling specified users, tools and development data objects to modeled user roles, product milestones and development schedule. KBS, DES and SD support these capabilities to some degree.

Integration: Encapsulating or wrapping selected information systems, repositories and data objects that can be invoked or manipulated when enacting a process instance (Garg et al., 1994; Mi and Scacchi, 1992). This provides a computational work space that binds user, organizational role, task, tools, input and output resources into "semantic units of work" (Garg and Scacchi, 1989). None of the DES or SD packages that have been used for software processes readily support process-based integration of externally developed software tools or legacy repositories.

Environment generation: Automatically transforming a process model or instance into a process-based computing environment that selectively presents prototyped or integrated information systems to end-users for process enactment (Garg et al., 1994; Mi and Scacchi, 1992). None of the DES or SD packages readily support this capability.

Instantiation and enactment: Performing the modeled process using the environment by a process instance interpreter that guides or enforces specified users or user roles to enact the process as planned. None of the DES or SD packages that have been used for software processes readily support these capabilities. In contrast, knowledge-based techniques have been demonstrated and used to support these capabilities using external process execution support environments (e.g., Garg et al., 1994; Mi and Scacchi, 1992; Noll and Scacchi, 1998).

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. KBS, DES and SD support these capabilities to some degree.

History capture and replay: Recording the enactment history and 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 or improvement opportunities. While this capability has been employed in a knowledge-based process engineering environment (Scacchi and Mi, 1997), such a capability could be added to common DES and SD package implementations.

Articulation: Diagnosing, repairing and rescheduling actual or simulated process enactment that have unexpectedly broken down due to some unmet process resource requirement, contention, availability, or other resource failure. While this problem-solving capability has been developed and demonstrated within a knowledge-based approach (Mi and Scacchi, 1993), it appears that the meta-model underlying DES and SD representations may not ever be able to support such a capability except in an ad hoc manner.

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 opportunistic benefits associated with new tools and techniques. KBS, DES and SD support these capabilities to some degree, though a knowledge-based approach foreshadows the of automated transformations and mechanisms to support such a capability.

Process asset management: Organizing and managing the collection of meta-models, models and instances of processes, products, tools, documents and organizational structures/roles for engineering, redesign and reuse activities. Only a knowledge-based approach has been developed and demonstrated (Mi, Lee and Scacchi, 1992), while such a capability is outside the scope of popular DES and SD packages.

While these process life cycle activities have been addressed above, they are described in more detail in the cited works. Clearly, many other scholars in the software process research community have made important contributions in these areas. But one goal that we sought was to identify and address of process life cycle engineering activities as we encountered them (cf. Mi and Scacchi, 1993; Scacchi and Mi, 1997). Similarly, our approach and environment for studying software processes was designed and implemented to support a tight integration of modeling, analysis and simulation (Garg et al., 1994; Mi and Scacchi, 1990).

Central to our approach to achieving this integration was the use of a knowledge-based process meta-model (Mi and Scacchi, 1990, 1996). Our research team could easily tailor this meta-model to various processes or process application domains. We believe this was possible because the foundations of the meta-model were grounded in empirical studies of software development and use processes (cf. Mi and Scacchi, 1990; Scacchi and Mi, 1997). Likewise, the meta-model was based on the theoretical conceptualizations of the resource-based ontology that we developed and refined (Garg and Scacchi, 1989; Mi and Scacchi, 1990, 1996).

Use of process meta-modeling techniques enabled our team to develop a number of associated tools that could be rapidly integrated with our established modeling,

analysis and simulation capabilities. Similarly, it enabled us to rapidly integrate externally developed or commercially available tools, whose selected inputs and outputs could be interfaced with the modeling and simulation facilities available to us (Garg et al., 1994; Scacchi and Mi, 1997; Scacchi and Noll, 1997). Accordingly, this capability enabled us, among other things, to interoperate and automatically transform models of complex organizational processes that we had captured and formally modeled into input formats required by the KBS and DES tools we employed (Mi and Scacchi, 1996; Scacchi and Mi, 1997). Thus, development and use of a process meta-model was a key component contributing to the success of our approach and effort.

4. Experience in industrial settings

We used our capability and facilities for software process simulation in a variety of industrial settings, as highlighted elsewhere (Scacchi and Mi, 1997; Scacchi and Noll, 1997). Our experiences were shaped in a substantial way by the interactions and feedback we encountered from our organizational sponsors and their staff (typically people at work in the settings being studied).

We used the Articulator environment to model, analyze, or simulate upwards of one hundred or more organizational processes. In this regard, we have constructed software process models and instances for organizations within different industries and government agencies. We have focused, for example, on activities involving team-based software product design and review processes, as well as department and division-wide information system operations and user support processes that include from tens to hundreds of participants. These models typically include dozens of classes of agents, tasks, resources and products, but a small number of software support tools and systems, while the process instantiation may include 1–10+ instances of each class.

In one extreme situation, we were asked to model, analyze and simulate a large-scale software development life cycle process that was to be used in building a next-generation telecommunications services environment. Through modeling and analysis, we discovered the process had 19 levels of (process) decomposition and management delegation. This is displayed in Fig. 4. Our advice to the company was that they needed to seriously rethink what they hoped to accomplish with such an onerous and cumbersome life cycle process, rather than have us spend time and resources trying to simulate this process. The company elected not to do this and our effort ended. However, we were not surprised to read in a trade newspaper some time later that the company had killed the software development effort guided by this process after spending more than $200M while failing to develop the target software system. Our lesson from this is that modeling, analysis and simulation can help improve software processes, but they cannot overcome management decision-making authority.

Our experience to date suggests the following: Modeling existing processes can take from 1–3 person-days to 2–3 person-months of effort. Process analysis routines can run in real time or acceptable near real-time. Software development process simulations can take from seconds to hours (even days) depending on the complexity of the modeled process, 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. This suggests that better performance can be achieved with (clustered) high performance computing platforms.

Overall, we found that use of KBS could be overwhelming to people unfamiliar with such advanced technology. Our ability to potentially model and simulate individual or small group interactions, then summarize the results of these interactions in a (stilted) narrative format was sometimes viewed with some skepticism. This may have been due to our inability to sufficiently communicate what was being modeled and stimulated (e.g., person-role-tool-task interactions with other person-role-etc.) versus what was not being modeled or simulated (how this person thinks or acts in response to interactions with another person).

The Articulator environment as a platform for software process modeling, analysis and simulation was designed to address the research interests of a small group of enthusiastic software process researchers. It was not designed to be a tool provided to end-users in other organizational settings. The complexity of the user interface displayed in Fig. 1 should underscore this. Furthermore, its implementation as a rule-based multi-agent problem-solving system with more than 1000 production rules, most of which are actually small programs implemented in Common Lisp should make clear that evolving it into a robust, user-friendly system would be impractical. Perhaps with recent advances in agent technology designed to operate over the Internet, some of the KBS concepts pioneered in the articulator environment can be reinvented and re-implemented in a more robust and intuitive manner.

Alternatively, we found the use and output of DES results to be much more easily accepted by more people. The functional capabilities and technical features of the DES package we used (WITNESS[tm]) were significantly less when compared to our KBS. However, the DES package did provide facilities for generating interactive graphic animations of simulated processes. These movies or animated cartoons of simulated processes were

informing and entertaining. More importantly, we repeatedly found people could observe and understand the dynamics of an animated process quite readily and intuitively. It appears that this aspect is particularly relevant. Specifically, when people could quickly grasp what was being portrayed in a simulation movie they were watching, they could often recognize process pathologies (e.g., work flow bottlenecks) as well as suggest alternative process redesigns or improvement opportunities (Scacchi and Mi, 1997; Scacchi and Noll, 1997). In turn, this feedback could then be used to further engage and empower these people to take control over the redesign of their as-is process, or the design of their to-be process. We took these to be a better indicator of customer satisfaction with our research approach and results in software process engineering. The apparent value to end-users of intuitive interfaces to software process modeling and simulation environments is therefore a valuable lesson learned.

5. Conclusions

Overall, our experience with these simulation capabilities can be summarized according to the kind of mechanisms employed. We found that knowledge-based simulation was of greatest value in supporting exploratory analysis of software process models. Knowledge-based simulation technology may be "rocket science" to most people. But it is useful when focused on understanding fine-grained or deep causal relationship of low frequency or loosely structured software processes. This can help facilitate *qualitative* insights into the operation of software processes. In contrast, discrete-event simulation was of greatest value in validating and assessing shallow process model instances of high frequency, short cycle-time processes using coarse-grain and statistically representative data samples. This helps facilitate *quantitative* insights into the operation of alternative software process instances. Thus, together we find that both knowledge-based and conventional discrete-event simulation capabilities are helpful when used to complement the strengths of one another.

The starting thesis of this paper was framed in terms of simulating and modeling software processes. Experience with the technologies gave rise to enhancements, extensions and the integration of many additional systems. Consequently, we found the software process modeling, analysis and simulation are best understood and practiced as part of an overall engineering of the software process life cycle. Furthermore, it seems that the use of a process meta-model has demonstrated a strategy for how to successfully integrate new tools and techniques for software process life cycle engineering.

We have had a variety of experiences and learned much for the use of simulation and modeling tools, techniques and concepts in understanding software development processes. Knowledge-based simulation capabilities now seem to have a limited future unless they can be reinvented using agent technologies. Discrete-event simulation packages, especially those that can produce animated visualizations of simulated process execution seem attractive. Either way, provision of robust and intuitive simulation capabilities seems essential if the goal is to get more people involved in software process modeling and simulation, or if the goal is to explore some of the emerging topics that were identified at the end of this paper.

Finally, we found that all of what we have learned, and much of the technical capabilities we initially developed for studying complex software development processes, can be readily applied to other organizational processes where software systems are used (Scacchi and Mi, 1997; Scacchi and Noll, 1997). This is also an important result that researchers may want to consider and exploit when investigating how simulation and modeling technology can be used for understanding and engineering software processes.

Acknowledgements

The research results described in this paper benefited from the collaborative contributions of many people. In particular, Peiwei Mi and John Noll played major roles in developing and evolving the software technologies described here. Pankaj K. Garg, M.J. Lee and Mark Nissen also contributed to the research efforts described in the paper. Their contributions are greatly appreciated.

References

Curtis, B., Kellner, M.I., Over, J.W., 1992. Process modeling. Communications ACM 35 (9), 75–90.

Garg, P.K., Mi, P., Pham, T., Scacchi, W., Thunquest. G., 1994. The SMART approach to software process engineering. Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, pp. 341–350.

Garg, P.K., Scacchi, W., 1989. ISHYS: Designing intelligent software systems. IEEE Expert 4 (3), 52–63.

Heineman, G., Botsford, J.E., Caldiera, G., Kaiser, G., Kellner, M.I., Madhavji, N.H., 1994. Emerging technologies that support a software process life cycle. IBM Systems J. 32 (3), 501–529.

Humphrey W.S., Kellner, M.I., 1989. Software process modeling: Principles of entity process models. Proceedings of the 11th International Conference on Software Engineering. IEEE Computer Society, Pittsburgh, PA, pp. 331–342.

Mi, P., Lee, M.-J., Scacchi. W., 1992. A knowledge-based software process library for process driven software development. Proceedings of the Seventh Annual Knowledge-Based Software Engineering Conference. IEEE Computer Society, Washington DC, pp. 122–131.

Mi, P., Scacchi, W., 1990. A knowledge-based environment for modeling and simulating software engineering processes. IEEE Trans. Knowledge and Data Engineering 2(3), 283–294. Reprint-

ed in (1991) Nikkei Artificial Intelligence 20 (1), 176–191 (in Japanese).

Mi, P., Scacchi., W., 1992. Process integration in CASE environments. IEEE Software 9 (2), 45–53. Reprinted in: Chikofski, E. (Ed.), Computer-aided Software Engineering, 2nd ed. IEEE Computer Society, 1993.

Mi, P., Scacchi., W., 1993. Articulation: An integrated approach to the diagnosis, replanning, and rescheduling of software process failures. Proceedings of the Eighth knowledge-Based Software Engineering Conference, Chicago, IL. IEEE Computer Society, pp. 77–85.

Mi, P., Scacchi, W., 1996. A meta-model for formulating knowledge-based models of software development. Decision Support Systems 17 (3), 313–330.

Noll J., Scacchi., W., 1998. Supporting software development in virtual enterprises. Journal of Digital Information (http://journals.ecs.soton.ac.uk/jodi/) 1 (4).

Scacchi, W., Mi, P., 1997. Process life cycle engineering: A knowledge-based approach and environment. Intelligent Systems in Accounting, Finance, and Management 6 (1), 83–107.

Scacchi, W., Noll, J., 1997. Process-driven intranets: Life-cycle support for process reengineering. IEEE Internet Computing 1 (5), 42–49.

Walt Scacchi received his Ph.D. in Information and Computer Science from UC Irvine in 1981. He was on the faculty in Computer Science and in the Marshall School of Business at USC from 1981 to 1998. During the 1980's he founded and directed the USC System Factory project. This was the first software factory research and education project in a university setting. During the 1990's he founded and directed the USC ATRIUM laboratory (www.usc.edu/dept/ATRIUM) to focus on the research and development of advanced technology resources for investigating and understanding managed processes. He has published over 100 papers and has served as a principal investigator on 25 research contracts and grants. He is also an active industry consultant. He can be contacted via email at Wscacchi @rcf.usc.edu.