

A PROCESS SERVER

Dennis Heimburger
Department of Computer Science
University of Colorado
Boulder, Colorado 80309-0430

Much of the research into process programming has been concerned with the formalisms needed to model and support processes. These formalisms are typically made explicit through process programming languages (PPL's), whose purpose is to support the definition of specific processes. These formalisms, and hence the associated PPL's may be divided into two classes: modeling and execution (or enaction). As a rule, modelling formalisms emphasize concise descriptions of the normal operation of a process. Models intentionally ignore many details of a process in order to achieve a concise and clear description of process. This is consistent with the idea that process formalisms for modeling are used primarily for explanation education, and analysis.

Process formalisms for execution are designed to drive so-called *process-centered* (or *process-driven*) environments. A process-centered environment is one in which the programmer is guided in the task of producing software according to some methodology. Such an environment extends the more traditional tool-oriented environment by adding the capability to specify the process by which software is to be constructed. This is in contrast to a typical tool based environment in which the programmer is presented only with a collection of tools and is given no help in deciding how to apply those tools to produce a software product.

To date, most of the work in process programming has been concerned with the definition of appropriate process languages and with the construction of example process programs to test out the utility of those languages. What has been missing from this work is a consideration of how, concretely, such languages can be used to *drive* an environment. There is agreement that the environment must have some form of process "component", but the exact nature and role of that component has not been decided. Additionally, there is some dispute about the correct style of programming to be used in executable process programs: prescriptive versus proscriptive. Yet, no-one has proposed a method by which multiple styles and multiple process languages can usefully co-exist in an environment.

I wish to propose the use of a process server approach as a concrete mechanism for solving (and in some cases bypassing) many of these problems. The *ProcessEngine* (with apologies to [GS91]) is the name of the prototype server under construction at the University of Colorado. The basic idea behind the ProcessEngine is to separate the process programming language from the process instance. Rather than focusing on the process programming language, the process server stores the state of a process in execution. It says little about how a process state is constructed and instead focuses on structure of the process state and the legal modifications that can be applied to that state.

In effect, the ProcessEngine shifts attention from formalisms for describing process code to formalisms for describing process states. This separation allows combining different code formalisms (i.e, mixing different process languages) as long as they adhere to a common state structure. In some sense, I am abandoning concern for process programming as embodied in code and instead

treating a process as yet another product in the environment, albeit one with special importance.

The idea for the process server comes from observations on the solutions to the ISPW6 Software Process Example[KFF⁺91] and its solutions. That example required the solutions to be able to dynamically create and abort various engineering tasks as part of the process. A common approach used in the solutions was to define and manage an explicit representation of the as engineering tasks as part of the process program. In the APPL/A solution, for example, each task in the problem was represented by a corresponding Ada task in the program, thus providing a direct representation of the process elements in code. Additionally, the program included several APPL/A relations that stored information about the attributes and structure of these tasks. In effect, the APPL/A program contained two distinct representations of the process: (1) APPL/A tasks for directly executing the process code, and (2) APPL/A relations to provide directly manipulable task representations. Consistency between those two representations was maintained by embedding maintenance operations in the APPL/A tasks and by using triggers to propagate information between relations and between relations and the APPL/A tasks. A number solutions using other languages such as AP5[Coh88] and Marvel[KBS90], produced similar solutions.

The process server idea elaborates on the use of an explicit process state. The key difference is to recognize that this explicit state can be described and maintained independently of the languages which manipulate it.

Acknowledgement: This material is based upon work sponsored by the Defense Advanced Research Projects Agency under Grant Number MDA972-91-J-1012. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

- [Coh88] Don Cohen. *AP5 Manual*. Univ. of Southern California, Information Sciences Institute, March 1988.
- [GS91] William Gibson and Bruce Sterling. *The Difference Engine*. Bantam Books, 1991.
- [KBS90] Gail E. Kaiser, Naser S. Barghouti, and Michael H. Sokolsky. Preliminary Experience with Process Modeling in the Marvel Software Development Environment Kernel. In Bruce D. Shriver, editor, *23rd Annual Hawaii International Conference on System Sciences*, volume II, pages 131–140, 1990. Kona, Hawaii, January, 1990.
- [KFF⁺91] Marc I. Kellner, Peter H. Feiler, Anthony Finkelstein, Takuya Katayama, Leon J. Osterweil, Maria Penedo, and Dieter Rombach. Software Process Modeling Example Problem. In Takuya Katayama, editor, *Proceedings of the 6th International Software Process Workshop: Support for the Software Process*. IEEE Computer Society Press, 1991. Hakodate, Hokkaido, Japan, October, 1990.