The Design of a Next-Generation Process Language*

Stanley M. Sutton, Jr. and Leon J. Osterweil

Department of Computer Science
University of Massachusetts
Amherst, MA 01003-4610

Abstract. Process languages remain a vital area of software process research. Among the important issue for process languages are semantic richness, ease of use, appropriate abstractions, process composability, visualization, and support for multiple paradigms. The need to balance semantic richness with ease of use is particularly critical. JIL addresses these issues in a number of innovative ways. It models processes in terms of steps with a rich variety of semantic attributes. The JIL control model combines proactive and reactive control, conditional control, and more simple means of control-flow modeling via step composition and execution constraints. JIL facilitates ease of use through semantic factoring, the accommodation of incomplete step specifications, the fostering of simple sub-languages, and the ability to support visualizations. This approach allows processes to be programmed in a variety of terms, and to a variety of levels of detail, according to the needs of particular processes, projects, and programmers.

1 Introduction

Process language research was an early emphasis of software process studies. It has remained vital for several reasons. First, no language has gained general acceptance or widespread use. This is not just a linguistic problem, as the use of languages depends also on organizational, methodological, and technological support. Second, first-generation languages generally have obvious limitations. This is in part because many of these languages were based on existing paradigms that were not particularly well adapted to the domain of software process [11, 24, 33, 26, 15, 4, 25]. Finally, research in other areas of software process has affected our ideas about what can and should be done with process languages. In this paper we report on the design of a “next-generation” process language that is intended to capitalize on lessons learned from first-generation languages, overcome limitations of those languages, and explore issues emerging from ongoing process research.

Section 2 identifies our primary language design goals, which are based on our experience with first-generation process languages. Section 3 describes the

* This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract F30602-94-C-0137.
design of JIL, our next-generation process language, including examples based on the Booch object-oriented design process. Section 4 discusses the multi-modal interpretation of JIL programs. An assessment of the JIL approach is presented in Section 5, and our status is discussed in Section 6.

2 Language Design Goals

Process programming proposes that it is feasible and valuable to represent software processes using programs written in compilable, executable coding languages [28, 29]. Our experience with APPL/A [33] has validated this proposal. We now take many properties of coding languages as fundamental to representing software processes, including formal syntax, well-defined semantics, executability, analyzability, object management, and consistency management. These issues have been the focus of much previous work (e.g., [31, 23, 11, 24, 6, 12, 33]), and they should continue to be addressed by second-generation process languages. Our focus here, however, is on the issues outlined below.

2.1 Semantic Richness

Software processes are multi-faceted and technically challenging applications. To support this domain, a process programming language must provide many kinds of interrelated semantics. This pressure for semantic richness is reflected in first-generation process languages. Many of these are based on extensions of conventional programming languages or paradigms, including functional languages ([11, 26]), rule-based or reactive languages ([24, 23, 12, 6]), imperative languages ([33]), and Petri nets ([4, 15]). Conversely, where process languages have neglected certain areas of semantics (e.g., reflexivity, resource modeling), process programs have suffered. Process language semantics must be both rich and rigorous. They must cover an adequate range of process semantics, and they must do so with appropriate models that support reasoning about processes.

2.2 Ease of Use

Ease of use is an important requirement for process programming languages because the individuals and organizations responsible for defining software processes are often not experienced at programming. The semantic richness of process languages means, however, that significant software engineering skills are required to program in them effectively. This is an impediment to the widespread adoption of process languages. A key issue for process languages is thus balancing the need for technical rigor with this need for ease of use.

2.3 Appropriate Abstractions

The clear and concise representation of software processes requires appropriate kinds and levels of abstraction. The development and maintenance of process programs is complicated if the user must construct process-specific abstractions from lower-level abstractions, as with process languages that are based on