Using UML for Software Process Modeling

Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel

Aachen University of Technology
Department of Computer Science III
D-52056 Aachen, Germany
{jaeger|schleich|bernhard}@i3.informatik.rwth-aachen.de

Abstract. We examine the benefits of using an object-oriented modeling language for software process modeling. We show how the Unified Modeling Language (UML) can be used to model software processes based on dynamic task nets, which evolve continuously during enactment. We have selected UML for various reasons: it is wide-spread, provides a comprehensive set of diagrams for both structural and behavioral modeling, and supports the early phases of process modeling (analysis and design).

Like many other object-oriented modeling languages, UML has no well-defined semantics. We indicate how a process model described in UML can be automatically transformed into an executable form, i.e., we provide dynamic semantics for UML models. To this end, UML models are transformed into programmed graph rewriting systems which are used to drive a process management environment.

Keywords: Software Process Models, Software Engineering Tools and Environments

1 Introduction

Software processes are highly dynamic. Many changes have to be taken into account while a software process is being executed: changing requirements, feedback to earlier stages of the software life cycle, moved deadlines, shrinking budget, etc. These changes challenge the capabilities of process-centered environments [7].

For modeling software processes, we have proposed dynamic task nets [9], i.e., hierarchies of tasks that are in addition connected by various kinds of horizontal relationships (control flow, data flow, and feedback). The most essential feature of these task nets is that they continuously evolve during the enactment of a software process. This contrasts sharply to the distinction between build time and run time, as it is made in most workflow management systems [13].

Originally, dynamic task nets were defined without any reference to an object-oriented modeling approach. However, the continuous evolution of task nets makes an object-oriented approach particularly attractive. Therefore, we have decided to examine the benefits of using an object-oriented modeling language for software process modeling. For this purpose, the Unified Modeling Language [3] appeared to be a natural choice. Some of the benefits we expected include:
If a wide-spread notation is used, process models can be communicated more easily to a larger number of people.

UML provides a large set of diagrams (class diagrams, object diagrams, collaboration diagrams, state diagrams, etc.) which can be used to define both structure and behavior of dynamic software processes.

Object-oriented modeling supports the earlier phases of process engineering (analysis and design), while most process modeling approaches, in particular those underlying process-centered environments, primarily focus on process programming.

On the other hand, we also expected some problems, in particular because UML is an informal modeling language which does not have a well-defined (dynamic) semantics.

In this paper, we describe how we are using UML for modeling software processes based on dynamic task nets. Section 2 summarizes the main features of dynamic task nets, Section 3 introduces the main components of the DYNAMITE process management environment. Section 4 constitutes the main part of this paper, which is devoted to process modeling in UML. In Section 5, we define the semantics of UML process models by a mapping into a graph rewriting system. Section 6 summarizes the lessons learned. Related work is compared in Section 7. Finally, Section 8 presents a short conclusion.

2 Dynamic Task Nets

The DYNAMITE model [9] introduces DYNAMIc Task nEts for software process management. A task represents a unit of work that is typically performed by a human developer (with tool support). A task may have input parameters and output parameters, which are the documents the task is working on. A task reads its input documents upon activation, works with these documents and finally produces some output documents. Tasks are connected by control flow relationships which describe the order of execution of tasks. In addition, there are data flow relationships which refine control flow relationships. While control flow state only the existence of temporal dependencies, data flows describe the passing of documents between tasks.

Figure 1 shows a dynamic task net which models the process of extending a software system during maintenance. At the beginning only little is known about the process. The request for extending the software system has to be analyzed and the application has to be redesigned. Finally, the modified system has to be installed. The intermediate structure remains unspecified, because it depends on the changes to the design document’s internal structure (part i). According to the new design document produced by the task Redesign Application, modules B and D have to be changed and a new module C has to be introduced. Thus, the project manager introduces two tasks for changing modules B and D together with a task for implementing module C. In addition, he adds subsequent test tasks. The control flow relationships between the test tasks reflect the module hierarchy, i.e., the topmost module D is the last one to be tested (part ii).