Humans in the Process: Architectural Implications

Dewayne E. Perry

Software and Systems Research Laboratory
AT&T Bell Laboratories,
Murray Hill NJ 07901
dep@research.att.com

1. Introduction

There are a variety of ways in which humans may be treated in a process-centered environment. At one end of the spectrum, the human is treated as a subroutine that is invoked by the environment in the execution of the process model in much the same that any tool is invoked within that process. At the other end of the spectrum, the process-centered environment provides a virtual machine upon which the human executes some portion of the process model. The first case represents a rather static view of process in much the same way that a program represents a static view of a computation. The execution of the various activities and the invocations of both humans and tools are under the control of the process model. The second case, on the other hand, represents a much more dynamic view of the process in which the instance of a particular model is created and adapted by the human ‘‘on the fly’’. While the model defines to some degree the general order of activities in the model, the human exercises a much larger measure of control over the planning and execution of those activities.

The degree to which a process formalism leans towards one or the other has a significant impact on certain aspects of the environment architecture. While many of the demands on a process-centered environment by these two positions are quite similar (for example, support for process evolution, automation of mundane activities and invocations of tools, process monitoring for measurement, analysis, feedback and optimization, etc.), the more dynamic emphasis of the second approach stresses two particular aspects of architectural support: dynamic instantiation of process fragments; and reification of the process model, process state and process history.

2. Dynamic Instantiation

The static and the dynamic approaches are similar in approach to basic instantiation of process models: both provide the ability to define generic process models that can be then customized by instantiating them with project-specific arguments. What distinguishes the two approaches is the time of binding the arguments to the generic models. The static approach tends to do most of the binding when the model is initiated — that is, the models are instantiated in much the same way that Ada generic packages are: at the time of model elaboration. The dynamic approach may do that as well — that is, provide for binding at the time of elaborating the model — but it requires a much more elaborate dynamic binding mechanism as well.

Part of this difference stems from the different degree of specification that is found in representative examples of these two different approaches. A process formalism that treats the human as a subroutine tends to provide more detailed descriptions of the process (as for example represented by the APPL/A version of the ISPW6 and ISPW7 examples). The process definition tends to be spelled out in specific details for the human processor. Customization, then, tends more towards parameterization, rather than amplification. A process formalism that treats the human as the controller of the process tends towards underspecified models (as for example in the Interact ISPW6 and ISPW7 examples). While Interact/Intermediate provide for parameterized customization, it also enables the human processor to amplify and elaborate the process model during its execution.

An example of this dynamic kind of process elaboration is provided by Interact/Intermediate. In the process modeling language Interact, the model definer may choose the level of detail to be be provided in the

structure of process activities. Facilities exist for providing the same level of specification as the static approach — that is, descriptions down to the last detail. However, the preferred approach is to specify only the various possible goals (and resulting obligations) for the activity leaving the implementation structure of the activity as *primitive*. One is also able to provide ‘‘highlights’’ of an implementation as guidance where there still much to be filled in to make it a complete implementation. In both of these cases, the actual implementation is amplified and elaborated at activity execution time.

This process of amplification and elaboration requires dynamic binding that takes more than just arguments for parameters. The human processor is building an implementation of the activity, either from scratch or from the guideposts provided by the abbreviated activity structure.

3. Model, State, and History Reification

Guidance is one of the aspects of process modeling that is paramount regardless of the approach taken. With the static approach, the detailed guidance is provided by the detailed specification of the model implementation. With the dynamic approach, however, much more elaborate support is required in order to be able to examine the current state of the process model, the current state and its relationship to the current model, and the history of what has been done in the past.

Given that the details of how an activity is to be done may not be specified, but at best implied by the activity description, the human processor requires elaborate support to determine what implementations are possible for a given activity. There are several ways in which this might be done. First, one might inspect other instantiations of the activity in question to see how others have elaborated the implementation. This solutions require the ability to retrieve previous instances of activity elaborations. Alternatively, one might inquire as to what activities provide the various results defined by the activity in question and use the support environment to interactively plan various strategies for elaborating the activity structure. This solution requires the ability to inspect the current state of the model definition.

The current state of the project, product and process is of paramount importance, for both the project manager and individual developers, in determining progress within that model as well as determining what activities can be done next. It is clearly the case that both approaches require access to current state. There are differences, however, as to the degree to which that state must be exposed. In the static approach, the process’ dynamic aspects require knowledge of the current state, much the same as a program at any given point requires data generated by the preceding computation – the current state is the basis for control within the process execution. In the dynamic approach, the humans controlling the process require access to the current state for guidance, both in terms of understanding the process and in terms of planning the continuation of that process. Clearly, reification of the history of the process is required for the same reasons.

In the case of the static approach, exception handling is typically built into the process descriptions in much the same way that forward recovery is built into fault tolerant programs. In an approach such as that exemplified by Interact/Intermediate, this level of detail is not provided in the descriptions and indeed is not provided in the process modeling language itself. We require then a much more elaborate interface to the process state in which error recovery can be done dynamically by the human controller in response to perceived problems in the process execution. Thus we need facilities to back out and restore the process state in various ways on the fly — that is, we need recovery facilities.

4. Conclusions

The place of the human in the process has significant architectural implications for process-centered environments. We have focused on two such aspects and distinguished the differences between a static approach where the human is considered to be a subroutine and a dynamic approach where the human uses the environment as the virtual machine on which to execute his or her process. In the latter approach, the environment must support much more elaborate dynamic binding and instantiation facilities, as well as more elaborate model, state and history reification facilities.