

A Revisionist Approach to Process Change

Dennis Heimburger
Department of Computer Science
University of Colorado, Boulder, CO 80309-0430
(dennis@cs.colorado.edu)

The PSEA Process Change Process

Changing an executing process on-the-fly has often been called out as a capability that distinguishes process languages and systems from programming languages and systems. At the recent Process-Sensitive Environment Architectures (PSEA) Workshop, one working group was devoted to looking at the issues of process change and came up with a generic process change process. This process identifies two basic classes of change.

Definitional Changes: This is the case where the process definition is recognized to be in error or otherwise inadequate. The problem is to change the definition and then propagate the changes to all executing instances of that definition.

Instance Changes: This is the case where, for unexpected reasons, the process to be followed must be altered, but the definition of the process is still considered to be adequate. An example might be that a design review must be bypassed because of time constraints. Note that a repeated occurrence of some instance change can imply the need for a definitional change.

The PSEA change process assumes that a change typically starts with a change to the process definition, propagates it to the executing state and then propagates it to the project user (in the sense that the user needs to be notified about some, but not necessarily all, changes). Instance changes are accommodated by allowing the change process to start at the state and propagate to the project user. This leads to the diagram of Figure 1.

Additionally, the PSEA process defines three general sub-steps for each of these “levels” of definition, state, and user.

1. Decide on the changes to be made, possibly based on changes propagated from a previous level.
2. Analyze the effects of the changes within the level. For example, changing a type definition at the definition level might have effects on various other parts of the definition. This can result in additions to the list of changes to be made.
3. Actually perform the changes.

For the project user, change generally will refer to the notifications to be delivered to the user as a result of changes to the state.

Some Requirements on Instance Change Mechanisms

If one accepts the PSEA model as valid, then it seems clear that changes to the process execution state are a central element of any process change. I claim that the mechanism for changing execution state should support at least the following capabilities.

Support for Hypothetical State: It is often the case that a change may have unintended consequences and so it is convenient to be able to “try out” a change and if it is not desirable, to undo it with *minimal* disruption.

Maintenance of State History: It is desirable to keep versions of the state reflecting the versions of definitional changes.

Revisionist History: It should be possible to support full or partial *revisionist* histories. A partial revision occurs when the state at the time of a definitional change remains consistent with the old definition, but later modifications to the state

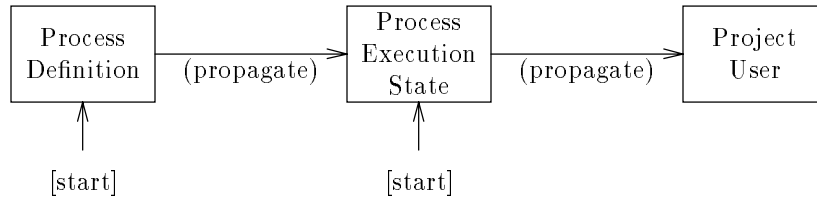


Figure 1: PSEA Process Change Model.

are made in accord with the new definition. In a full revision, the whole state is made consistent with the new definition. This obviously requires modifying existing state structures.

Supporting Process Change by Rapid Re-Execution

I want to propose a mechanism for supporting changes to the process state that meets the above list of capabilities. This mechanism makes several assumptions about the context in which it operates. First, assume the existence of a process definition and the existence of an executing instance state reflecting that process definition. Second, assume that a change occurs to that definition and that the change is to be propagated to the instance to produce a new instance that is consistent with the new definition; that is, full revision is desired. Finally, assume that the process state is represented as a tree of tasks¹.

The mechanism is based on explicit state versioning plus what I call “rapid process re-execution.” The basic idea is conceptually simple. When a definitional change is to be propagated to the process state, a new “version” of the state is instantiated where the new version has no initial state. The new version is populated by attempting to apply the new process definition against the old process state². The basic step involves choosing a task from the new version and attempting to expand it by using the expansion of the corresponding task from the old version. If the expansion can be done in the new definition, then it is duplicated from the old state into the new version. The process repeats for newly added tasks. If the expansion fails because it is inconsistent with the new definition, then the task is marked as needing to be truly re-executed. Note that failure to expand a task

¹This may be a strong assumption, but I believe that the approach outlined here should work with other representations as well.

²I use the term “apply” in a very loose sense. The Process-Wall [Hei92, Hei91], for example would operate by notifying the clients that specify the process definition.

can affect subsequent tasks that depend on the output of that first task.

In effect, then, the change mechanism pretends to re-execute the process and where the re-execution indicates no changes from the old state, it by-passes true execution and incorporate the old state into the new state. The end result is a new process state with gaps indicating pieces of the process that need to be revised/re-executed using the new process definition. At this point, the gaps can be brought to the attention of appropriate project personnel who can decide what to do. In the case that the change has some defined repair, it can be applied to revise the state automatically. In other cases, manual examination is necessary.

In this approach, the version serves not only as historical record but also serves to encapsulate the hypothetical state mentioned above. If the required repairs are deemed too extensive, the new version can be wiped out. Note that while the new version is being re-executed, the old version can continue to be executed in parallel. The parallel execution will be incorporated into the new version when the re-execution “overtakes” the old version. This re-execution mechanism can handle both full and partial revision. Partial revision is trivial in that essentially the whole existing state is incorporated into the new state and new changes will appear only in that new state. Full revision is handled by potentially rebuilding (via re-execution) all of the pieces of the state affected by the revised definition.

It is possible that the re-execution process can be optimized in certain cases. It may be that the changes between definitions can be analyzed in sufficient detail so that the state changes can be computed directly and applied in one batch to the state. This would avoid the sequential element of re-execution³. The advantage of full re-execution is that it may handle cases that the optimization cannot.

Obviously the idea of re-execution has some prob-

³This optimization, but not the full re-execution mechanism, would appear to be similar to the change mechanism developed for Marvel [BKH92].

lems and leaves many important problems unsolved. Hidden in this mechanism is the assumption that process changes do not propagate widely in the state. In large part, this must be determined experimentally. However, iteration in the process definition is one place where this assumption probably does not hold. For example, suppose the state records a sequence of design efforts followed by design reviews. If the design review is modified, then its output may invalidate all subsequent design reviews. This may mean that the re-execution will stop at the first review, leaving much of the process to be truly re-executed.

In order to handle this case, it is reasonable to make re-execution operate as a co-routine with respect to repair. That is, if the first review is determined after repair to have no important consequence on the design, then re-execution can continue on to the next cycle of design and review. This can continue until re-execution finds a review that does have an effect and then it must stop.

Another problem concerns task re-ordering. Suppose that the change to the definition only re-orders some subtasks of a task. Naive re-execution would claim that the subtree must truly be re-executed. Smart re-execution would attempt to match the new subtasks against other tasks in the old state and would recognize the re-ordering. This approach generalizes to the full task tree. Before marking any task as needing true re-execution, the whole prior state would be searched for task instances that might match the new task. This would allow for very general re-ordering of tasks at the cost of matching task instances, which can be difficult.

I am sure that there are other problems not yet recognized. Nevertheless, this is a promising and fairly general approach to handling changes to process state as a result of changes to the process definition.

Acknowledgement: This material is based upon work sponsored by DARPA Grant Number MDA972-91-J-1012 and DARPA/ONR Grant Number N00014-92-J-1862. 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

- [BKH92] Israel Z. Ben-Shaul, Gail E. Kaiser, and George T. Heineman. An Architecture for Multi-User Software Development Environments. In *Proc. Fifth ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, Washington, D.C., 9-11 December 1992.
- [Hei91] Dennis Heimburger. A process server. In *Proceedings of the 7th International Software Process Workshop*, Yountville, CA, 15-18 October 1991. (to appear).
- [Hei92] Dennis Heimburger. The ProcessWall: A Process State Server Approach to Process Programming. In *Proc. Fifth ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, Washington, D.C., 9-11 December 1992.