Feedback Handling in Dynamic Task Nets

Carl-Arndt Krapp and Bernhard Westfechtel
Lehrstuhl für Informatik III, RWTH Aachen, D-52056 Aachen
[krapp|bernhard]@i3.informatik.rwth-aachen.de

Abstract

While a software process is being executed, many errors and problems occur which require to reconsider previously executed process steps. In order to handle feedback in a process management system, several requirements need to be addressed: adaptability, human intervention, impact analysis, change propagation, restoration of the work context, and traceability. Feedback management in DYNAMITE meets these requirements. DYNAMITE is based on dynamic task nets and specifically supports feedback through feedback relations, task versions, and customized semantics of data flows. A methodology for feedback handling is also represented.

1 Introduction

It has been recognized by many researchers that software processes are highly dynamic. However, current models and tools address process dynamics only to a limited extent. In particular, this applies to management of *feedback*. While a software process is being executed, many errors and problems occur which require to reconsider previously executed process steps. For example, a module test may reveal errors in the implementation, or a designer may detect inconsistencies in the requirements definition.

While feedback is mentioned in virtually every textbook on software engineering, it is by no means obvious or trivial how to support it in a process management system. To this end, an underlying formal model is needed which meets the following requirements:

Adaptability. In routine processes, feedback may be planned. On the other hand, in less well-structured processes, feedback may occur unexpectedly and randomly, calling for flexible and reactive (instead of proactive) support.

Human intervention. In many cases, handling of feedback requires human judgment and can at best be automated partially. For example, in case of severe errors, large parts of a process may have to be suspended while minor errors only affect a small set of process steps. Usually, the process engine cannot tell these cases apart.

Analyses. Sophisticated analyses have to be offered to assess the impacts of feedback, the current status of processing feedback, etc.. These analyses concern the past (terminated steps to be iterated), the present (active steps to be suspended), and the future (planned steps to be postponed).

Change propagation. Flexible commands must be provided to propagate changes induced by feedback timely (to avoid wasted work on obsolete inputs), precisely (to exactly determine actually affected steps), and incrementally (to process changes rather than to restart from scratch).

Restoration of the work context. In many cases, iteration of a process step is performed by refining its previous results. To support incremental improvements, the old work context (inputs, outputs, available tools, etc.) has to be restored.

Traceability. Handling of feedback has to be recorded and packaged in an experience base. By analyzing historical data concerning feedback, problem spots in the software process may be identified.

2 Feedback Handling: Mechanisms and Methodology

In DYNAMITE [1, 3], a software process is represented by a *dynamic task net* (Figure 1). Development activities are modeled by *tasks* (boxes) which are related by *control* and *data flow relations*. A dynamic task net resembles a PERT chart and, additionally, offers execution semantics. A task can be subdivided into several subtasks (not shown in the figure). Therefore several levels of abstractions can be provided.

The example in Figure 1 shows a cutout of a task net for constructing a multi-pass compiler. The cutout only covers a subset of its data structures (`GraphCode` for the intermediate code and `BinFile` for the compiled code) and a subset of its passes (`SemAna` for semantic analysis and `CodeGen` for code generation). After an initial coarse design, module interfaces are defined (`Interface`), module bodies are implemented (`Implement`) and finally tested (`Test`). Tasks for defining interfaces and testing modules are performed in bottom-up order; implementation tasks may proceed in parallel. Even tasks connected by control flow relations (e.g., `ImplementGraphCode` and `TestGraphCode`) may be executed in parallel (simultaneous engineering).

In order to model feedback situations, where the flow of control is redirected to earlier stages of the

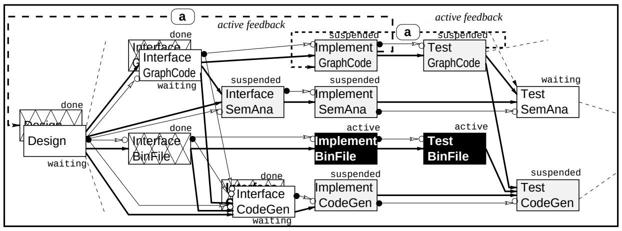

Figure 1: Dynamic task net

overall process, a special *feedback relation* is used. In our example, the test task for the `GraphCode` module observes that performance requirements are not met, resulting in feedback to the corresponding implementation task. Since the implementation task could not solve the problem, another feedback relation is introduced back to the design task. The consequences of feedback introduction like suspension or restart of dependent tasks are under manager's control. Here, we assume that the interface of `GraphCode` needs to be modified. Accordingly, all dependent tasks are suspended (not including those tasks which are only concerned with the module `BinFile`).

During feedback handling, traceability of the overall process can be lost very easily. Due to the traceability requirement we have introduced the concept of *task versions*. If an already terminated task has to be restarted again, a new task version is established which can be provided with the same work context (same inputs, same outputs, same actor, etc.) as the original task (shown behind the new version) or which may be embedded into the task net differently.

By means of the presented concepts, a manager can handle arbitrary feedback according to his needs without loosing traceability. Analyses on dynamic task nets are offered which support him in finding potentially affected tasks.

We now consider how feedback is handled methodologically. The following steps can be identified:

Target determination. After occurrence of an error, the first step is to determine the target task of the feedback. This can be either done by the actor of the source task himself or by the process manager who usually has a better overview of the overall process.

Source behavior. After feedback introduction the behavior of the source task has to be determined. Several alternatives are possible: The source task may stay active, or it may be suspended or aborted.

Consequences to source successors. Successors of the source task may be affected, as well. Analysis on the net supports the manager in deciding upon appropriate reactions. All tasks reachable by control flow relations can be highlighted. The manager may now decide to suspend a subset of these tasks in order to avoid wasted work. Tasks which are likely not affected may continue their work.

Target behavior. If the target task has already terminated, a new task version is introduced. In case it is still active, which can be possible if parallel execution of tasks is allowed, appropriate reactions have to be performed. Either the responsible actor is only informed about the feedback, or (s)he is forced to process the feedback immediately.

Consequences to target successors. Target successors are potentially affected. Again, analysis on the task net helps to identify these tasks. Similar to source successors, the manager can perform appropriate reactions individually for each task.

Feedback propagation. After all consequences have been estimated and appropriate reactions have been performed, feedback can be propagated trying to accelerate the wave of changes through the net. Note that a feedback relation is called *active* until it is terminated (see below). In Figure 1, this was visualized by a flag attached to the feedback relation.

Feedback termination. Active feedback relations influence the behavior of some tasks. Thus it is essential to determine the point when a feedback can be considered as closed. Again, different policies can be used depending on the kind and severity of the feedback: (1) Feedback can terminate if the flow of control reaches the feedback's source task again. (2) Feedback can terminate if the feedback's target task terminated successfully. (3) The manager determines explicitly when feedback can be terminated.

3 Conclusion

In this paper, we have presented the main ideas of our approach to feedback handling at an informal level. A detailed presentation and comprehensive comparison to related work is given in [3] and [4]. We have formalized dynamic task nets by means of programmed graph rewriting. This formal specification serves as the basis for rapid prototyping of a process management system [2].

References

[1] P. Heimann, G. Joeris, C.-A. Krapp, and B. Westfechtel. DYNAMITE: Dynamic task nets for software process management. In *Proc. ICSE 18*, pages 331–341, Berlin, Mar. 1996.

[2] P. Heimann, C.-A. Krapp, and B. Westfechtel. An environment for managing software development processes. In *Proc. SEE '97*, pages 101–109, Cottbus, Germany, Apr. 1997.

[3] P. Heimann, C.-A. Krapp, B. Westfechtel, and G. Joeris. Graph-based software process management. *International Journal of Software Engineering and Knowledge Engineering*, 7(4), Dec. 1997. To appear.

[4] C.-A. Krapp and B. Westfechtel. Feedback handling in dynamic task nets. Technical Report AIB 97-9, RWTH Aachen, Germany, 1997. `ftp://ftp.informatik.rwth-aachen.de/pub/reports/index.html`