Lightweight Tool Coordination

Path* - a minimal framework for tool coordination

Reid D. McKenzie
Computer Science
The University of Texas at Austin
Austin, TX
reid@cs.utexas.edu

Dewayne E. Perry
Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX
perry@mail.utexas.edu

***Abstract*—*We present a lightweight tool for coordinating tool usage in a structured and unobtrusive manner allowing for the formal description and implementation of development cycles with minimal human intervention. This tool, here and after referred to as ``Path*'' seeks to provide a minimal yet powerful framework for tool coordination by scripting actions to be triggered on events such as disk writes to a project directory and version control system commits. These events execute user-defined scripts for the purpose of automating tasks such as partial rebuilds and style checking in an IDE and platform independent framework.*

***Index Terms*—Tool Coordination, Process Guidance, Process Automation, Process Scripting. Plug-in.**

I. Introduction

In the late 1980s and early 1990s, one of the leading process research ideas was that of process centered environments (PCEs). The desire to use defined processes as means for systematic and consistent software developments was the main driver underlying this idea. "Among the many benefits touted for process-centered environments are the ability to automate various aspects of a process and the ability to monitor the progress of a process in order to guide, enforce, or measure that process." [5] Among the various approaches to this idea were Kaiser's Marvel System [3] and Minsky's Law-Governed Systems [4]. Both approaches used descriptive rules as the basis for governing the development and evolution of software systems. Minsky went further and used his declarative laws to cover the structure and characteristics of the product as well.

What has become the de facto approach instead has been the integrated development environment (IDE) framework that is individualized and/or customized with the extensive use of plug-ins, such as Eclipse [1].

The question we want to consider is how to provide a useful process mechanism that can be used in this context (for example, a plugin), provide guidance, automation, and enforcement, and that is both light-weight and useful enough to avoid the adoption problems of Marvel and LGS.

An approach, but less expressive approach, is suggested by path expressions [2]. Path expressions are regular expressions that define the allowed sequence of operations on shared protected data. We believe that path expressions can be used to serve a useful purpose in providing lightweight definitions of desired or allowed sequences of, or policies about, developer actions relative to tools and tool commands being used.

Here, we propose and argue for a lightweight tool, Path*", in seeks to offer unintelligent yet useful assistance to developers by automating menial tasks such as rebuilding and style checking, while simultaneously providing for the definition and enforcement of development workflows using a path-expression derived formalism for defining legal editing and work patterns.

II. Utility of Automated Hooks

Many IDEs such as Eclipse and Netbeans provide automated partial recompilation of code bases at edit time so that syntax and type errors may be indicated to the programmer as quickly as possible, even before she has progressed beyond the point of error in editing. However such automated rebuilding and checking support is extremely environment dependent and not extensible to other languages. Path* would provide a simple and extremely extensible structure under which all file writes in the monitored directories would trigger an "on-edit" script which could use simple policies based on the path of the changed file and its file type to determine the appropriate compiler and chain-load such operations as the developer may deem appropriate.

By implementing this extended hooks structure in terms of shell scripts rather than some more domain-specific scripting language we explicitly and deliberately leave open the possibility of Path* hooks being used to chain-load other user-defined tools such as file and repository permission policies, manager authorization requests/checks and so forth.

This would be a cleaner and more flexible alternative to the rebuild scripts used in some development projects which simply continuously invoke the appropriate compiler until the script is terminated wasting both CPU time and programmer time in devising such once-off spinning scripts.

III. Utility of Regular History Verification and consequence of hooks

Considering the development process to be a sequence of operations such as edits and version control system (VCS) commits, it makes sense to consider the entire development process as a relatively stable and predictable sequence of operations such as edits, compiler invocations, lint or other verification tools, test suite runs terminated with a VCS commit.

A typical development cycle using C, for example, could be characterized as a sequential path:

```
1.  checkout & lock in VCS
2.  (edit | lint)+
3.  state:no-lint-problems
4.  compile
5.  state:no-build-problems | goto 2
6.  VCS checkin & unlock
```

which cleanly represents a basic process model ensuring that the core build on the main repository is never broken. Practically speaking, requirements about the state of the project repository are expressed as process requirements such as these and implemented as version control system hooks allowing for preconditions such as requiring that the preconditions of build and lint sanity prior to checkin. The Git, CVS and Mercurial version systems all provide verification hooks known by the “pre-” prefix. These hooks are simple Unix shell scripts which can abort the VCS operation should they return a nonzero (shell false) value.

While the implementation of hooks is obvious, they are at least under the Git version system restricted in that some operations such as merges of multiple development threads. Such a hook would be useful in allowing the definition of rules restricting which threads of development may be affected and under what conditions. For example a development environment using a “development/testing” codebase to which programmers have free access providing a condition that all commits are in accordance with a commit precondition policy like the above and a “master” codebase which is regularly as the deployment base in a critical application. In this position one can define a “release” as a sequence of edits which is moved as an update from the development codebase to the master codebase presumably after extensive testing and staging. Therefore it would be valuable to codify this implicit rule that development code must be audited in a pre-release state and cannot be merged directly into the live codebase. In order to do so however we must extend the standard hooks provided by Git and other repository systems to include a “pre-merge” hook in order to implement this policy. Also for “post-merge” events such as automatically evoking a bump script after merges to master or other such side-effect policies.

It is obvious that hooks are an effective and efficient tool for implementing preconditions on when a developer may perform actions and what tests should be automatically invoked to verify that the appropriate preconditions have been met. However, the hooks system can only be chained by the version control system. As we noted earlier, there is great value to be derived from automated invocation of tools such as lint and make on a “when file is flushed” basis: a basis which version systems are unable to support.

IV. Design & Structure of Path*

As existing VCSs provide a subset of the event hooks which we consider most valuable, we feel that it would be foolish to ignore them as part of the implementation of a Path* work environment in no small part because Path* is intended to augment VCS workflow. Consequently we propose to implement the Path* system as a meta-VCS command suite which provides a wrapper over VCSs such as CVS and Git explicitly using their hook features rather than attempting to supplant existing infrastructure.

This leads us to a definition of Path* as a tool which first executes any predicate/guard hooks not defined by the wrapped VCS, executes the appropriate VCS command passing arguments through and then executes any and all side-effect hooks not defined by the VCS upon a zero exit code (success) from the VCS command. This lends itself to an extremely flexible structure as a simple wrapper that can be ported to any VCS at all regardless of what hooks facility of lack thereof it may possess. Furthermore by providing a standard for hook script argument format we free hooks from dependence on the VCS for answering questions such as “what is the current branch” or “what files are staged for commit”?

V. Conclusions

Here we present existing version control and hook systems in a historical context of process and workflow model research, arguing that they are good and helpful tools. However, we further present our own system, Path* which seeks to provide a meta-vcs wrapper around these tools to provide a simple and lightweight implementation of additional features atop existing infrastructure. Our approach increases the potential utility of the system to developers and developer employers alike by adding support for the implementation of additional simple and useful tools without burdening users with concerns about their specific platform. We then further argue that not only is our meta-platform useful, but that as it offers a greater diversity of trigger events than any existing platform that it simplifies the implementation and enforcement of finely grained yet largely unobtrusive policies regarding developer workflow.

VI. Referenced

[1] Anonymous. "Eclipse - The Eclipse Foundation open source community website", 30 January 2013, http://www.eclipse.org/

[2] Roy H. Campbell and A. Nico Habermann. "The specification of process synchronization by path expressions", Proceedings of an International Symposium on Operating Systems, April 23-25, 1974 (Lecture Notes in Computer Science, No. 16).

[3] Gail E. Kaiser, Peter Feiler, and Steven Popovich. "Intelligent Assistance for Software Development and Maintenance", IEEE Software, May 1988.

[4] Naftaly Minsky, "Law Governed Systems", IEE Software Engineering Journal, 6:5 (September 1991).

[5] Alexander L Wulf and David S Rosenblum, "Process Centered Environments (Only) Support Environment-Centered Processes", Proceedings of the 8th International Software Process Workshop (ISPW8), Schloss Dagstuhl, Germany, March 2-5, 1998.