xmb - dancing in the tar pits
Using Tcl/Tk to build a CASE environment

Henry R. Tumblin
CertCo L.L.C., New York, NY
Charles E. McElwain
Open Market, Inc., Cambridge MA

xmb - dancing in the tar pits

Using Tcl/Tk to build a CASE environment

Henry R. Tumblin
CertCo L.L.C.
55 Broad Street
New York, NY 10004
tumblin@certco.com

Charles E. McElwain
Open Market, Inc.
245 First Street
Cambridge, MA 02142
mcelwain@openmarket.com

Managing software across multiple build platforms and environments can be thought of as dancing around a tar pit; if you slip and fall then you may not be able to get out for a long time! There's far too much knowledge about the local build environment required to enable software developers and release engineers to build, test, and release products. We intend to show that by using **Tcl/Tk** as a CASE integration engine, the amount of time and training required to bring a new developer, test or release engineer online is greatly reduced. Product build environments become standardized and the cost of adding new platforms is reduced.

The challenge we faced was one of release engineers having to build on multiple platforms, developers needing to build and test on a subset of these platforms, all working from a variety of individual environments and experience levels.

The problems that had to be solved were:

- distributing builds across the multiple platforms
- centralizing knowledge of build configurations in a tool (for repeatability)
- giving feedback on system resource usage and sharing
- providing better feedback and control (via a UI) and minimize training
- evolving a CASE tool towards greater maintainability, extensibility and reusability.
- adding multiple projects, and co-ordinating and analyzing inter-project dependencies

We looked at solving these problems through a three layer tool containing: a point-and-click GUI, a CASE kernel, and the remote tasks. The tool was based on a "build flow" which ran through:

Select Project -> Select Platform(s) -> Select Options -> Distribute -> Analyze

The problem was then how to turn this into a programmatic flow using **Tcl/Tk**.

We built a small kernel representing these tasks, whose behavior is controlled by and changeable via config files which are themselves **Tcl**. Scripts to execute on the remote hosts were generated, then launched on the remote host. From within these scripts, host-specific config files were sourced to enforce the build environment and policies, as well as supplemental information, including compilers, compiler flags, paths, etc.

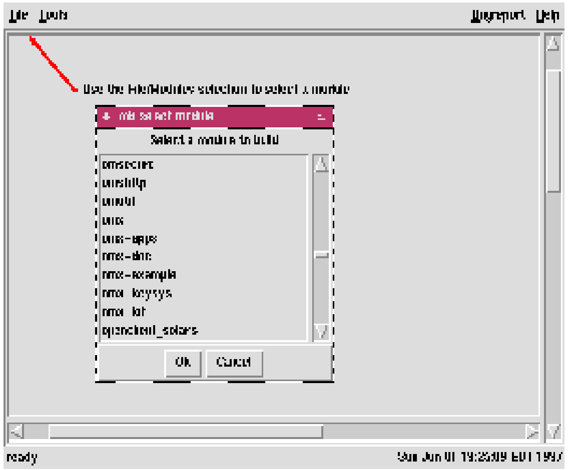
These scripts captured statistical information on each distributed build in the form of a **Tcl** statement representing a database entry. The reduction code for this is a small kernel which sources all database entries as text files and builds several structures as crosses of the data, and provides an API to access these views.

With the CASE kernel implemented, we then built a point-and-click GUI using **Tk**. We could now easily supplement the earlier **Tcl** kernel with point-and-click interfaces to the above stages of the build process. Control, feedback and configuration was now also visual. Training on the CASE environment became easier, intuitive and standard across multiple platforms and projects.

For maintainability of the CASE tool, we had to minimize global state. We did this by a hierarchical structuring of state data, class-oriented naming of procedures which contain self-initialization (constructor) code, along with standard module headers for extracting documentation.

In order to more easily support and analyze multiple projects, both within single invocations and across multiple invocations of the CASE tool, we used the same techniques as in handling the need to report resource usage via a **Tcl**-based database. A daemon was then implemented to co-ordinate and serialize writing to the database, utilizing TCP primitives in **Tcl** (thus avoiding the need for custom socket code).

Figure 1: Selecting a project or module to work on. This list is generated from the underlying source control system, CVS in this case.



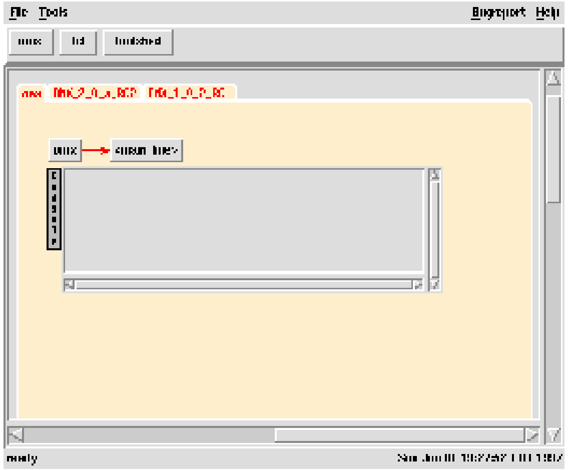


Figure 2: Selecting different projects, **omx, Tcl**,and **Toolshed** and multiple tags, *main_line, OMX_2_0_a_RC2, OMX_1_0_2_RC1*, within one of them.

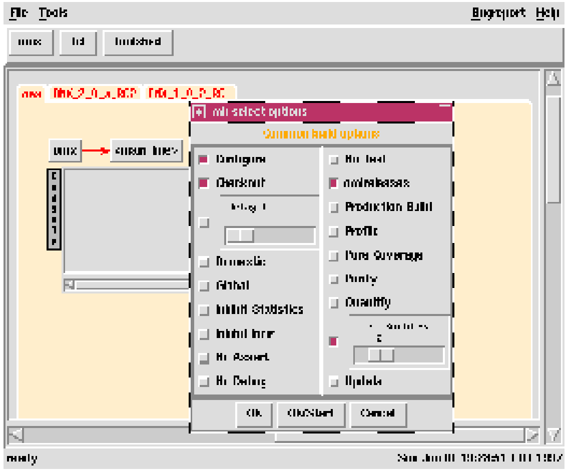


Figure 3: Selecting build options for the current project **omx** and tag *main_line*.

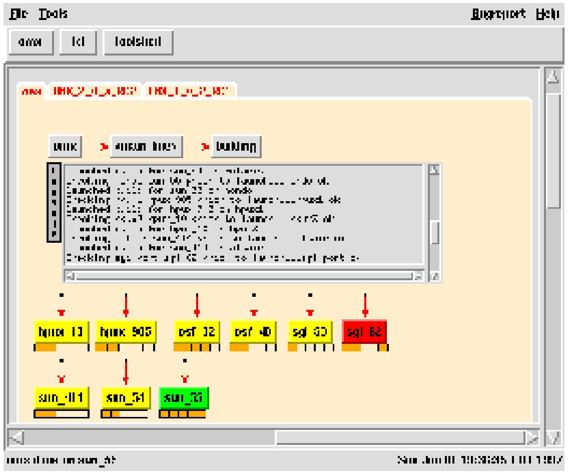


Figure 4: Watching a build in progress across multiple platforms. One has succeeded, *sun_55*, one has failed, *sgi_62*, and the others are still in progress.