Seminar Configurable Systems

Prof. Zeller / Christian Lindig — Cons

Summary

Cons is a tool for building software from a hierarchy of source files. Cons was implemented by a team of people where the principal architect appears to be Steven Knight. Compared with Make, Cons offers explicit support for many situations that Make only supports by idioms.

- Hierarchical projects: source files, intermediate files, and finally programs are distributed over a hierarchy of directories.
- Automatic dependency analysis and fine-grained dependencies based on cryptographic hash sums. In particular, a derived file depends on the action that constructs it.
- Modular specification files with explicit import and export of variables between them (along the hierarchy) and thus local scopes for variables.
- Support for variant builds: a single set of sources can generate a variety of products that may exist in parallel.
- A build environment bundles all variables that effect a build. The environment may be specialized for parts of a build process but in principle is intended to hold all parts that may vary from build to build.

Cons is implemented as a stand-alone tool based on Perl; specification files use Perl's syntax and in fact are high-level Perl scripts that the tool evaluates. As such, Cons is a mixture between a library that extends Perl with capabilities for software construction, and a domain-specific language for construction where the underlying implementation language shines through.

A Cons specification file, by convention named *Conscript* or *Construct*, contains mostly variable definitions and declarative commands. The commands relate source files and important intermediate files (like libraries) or results; they do not mention intermediate files like object files. The knowledge how to produce and use them is built into Cons and the actions that produce them are parameterized by variables in the build environment.

Cons derives dependencies between files from two sources: high-level commands (like `Program` or `Library`) and by scanning source files for included files. Each source file can have an associated scanner routine (implemented in Perl) that produces a list of *source* files, which are recursively scanned.

Because it is manual, the paper contains a lot of details that are relevant for the user, but does not presents its concepts in the best possible way. Except for Make, no comparisons are drawn with other tools that may employ similar concepts.

While Cons tries to be language independent, it is biased towards C and C++. This is evident from the built-in knowledge for building object files, the default construction environment, and the approach to dependency scanning. Cons provides hooks for languages different from C but they don't appear as tried. In particular, it is not obvious how to add knowledge for such languages into the tool without changing its implementation.

The author praises his implementation language Perl for being simple, well-defined, and with familiar semantics. This seems questionable, since even he struggles with Perl when implementing a new dependency scanner. It is well known that Perl's semantics are complicated and heavily depend on the actual Perl version. However, it could be argued that a powerful and general scripting language benefits a software construction tool. Compared with Make, I expect a user of Cons to resort less often to external tools to express a certain fact in a specification file.

Overall, Cons tries to provide syntactical support for important concepts in software building but falls short in their implementation. Consequently, Cons is no longer under active development and was superseded by a new implementation based on Python.