Seminar Configurable Systems

Prof. Zeller / Christian Lindig Make

Summary

Make – A Program for Maintaining Computer Programs by Stuart F. Feldman describes a file-oriented tool named Make that automates the build process for a modular software system. In such a system a change to one of its modules typically does not require all modules to be rebuild, i.e., to be compiled and linked, but only modules that directly or indirectly depend on the changed module. Make knows about these dependencies from a user-provided specification, called a Makefile, and will execute just the commands necessary to update a given module, which is called a *target* in Make's terminology.

The specification language is simple and independent of the implementation language of the system being managed. The basic concept is a rule that relates a module, the modules it depends on, and an action that updates the module provided the dependencies were brought up to date previously. Conceptually, the rules form a directed acyclic graph with modules as nodes and action-labeled edges. Make traverses this graph depth first to find modules that changed after a target was built. It detects changes by comparing the time stamps of modules as they are provided by the Unix file system: a module is out of date, if one of the modules it depends on has a more recent time stamp. An action that updates a module is a sequence of commands that are passed to a Unix shell for execution. Thus, the command language and its interpreter are external to Make; only a minimal language with rules and variables is built into Make. Rules can be abstracted with generic rules based on file-name patterns.

The paper uses a small running example to demonstrate dependencies in a project for which it develops a Makefile. Therefore, the presentation is practically oriented and does neither offer theoretical details, nor an evaluation besides a small experience report.

Make appears to work well for small projects with all files in a single directory, but the paper does not discuss how well it scales to larger projects. In such a project Make's global namespace for variables and rules may become a problem, too.

Given that we are still using Make, 25 years after its inventions, Make must have done many things right. In my opinion these are language independence and simplicity.