# Seminar Configurable Systems

Prof. Zeller / Christian Lindig Tcl

## Summary

In *Tcl: An Embeddable Command Language* John K. Ousterhout proposes to make applications end-user programmable by embedding a command language into them. The paper discusses the design of such a language, and its implementation as a C library.

Ousterhout observes that some end-user programmable applications like the Emacs editor are powerful and popular. For their easy and systematic creation he proposes the Tcl (Tool Command Language) toolkit. It provides an interpreter for a shell-like language implemented as a C library. An application becomes programmable with Tcl when important functionality of the application is added as new primitives to a Tcl interpreter, which is then linked with the application. A primitive can take either the form of a new primitive command, or of a new (abstract) datatype.

Embedding an interpreter into an application has implications for its control flow: the application does not control the interpreter, but rather is controlled by it: the building blocks of an application are called from the interpreter and hence ultimately from the program the interpreter is executing. This leads to an application architecture where the core application becomes a library consisting of passive functions that are called from the interpreter.

The design of an embedded language has two important aspects: the language exposed to the user, and the implementation exposed to the developer. On the language level, Tcl is close to a Unix shell: a command evaluates string arguments which it interprets as file names, expressions, numbers, or lists of strings. While being an imperative and procedural language, Tcl is substantially simpler than, say, Pascal: no user-defined types, just one data type (string), no static scoping, or closures. However, Tcl does have exceptions. Ousterhout justifies this simplicity with a likewise simple implementation: null-delimited strings are easiest to handle from within C.

Ousterhout likes to keep things simple but I believe he overdoes it at least for the language: the everything-is-a-string approach leads to an unreadable and slow program full of quotes and escapes, in particular when using lists in Tcl, the only supported higher-level datatype. The Tcl language also misses the separation into a syntactical and a grammatical layer that is common in modern languages. Even Lisp has it, a language that Ousterhout names as an inspiration. In addition, Lisp has several scalar and complex data types, which are badly missed in larger Tcl programs. Contrary to what Ousterhout claims, programs written in embedded languages need not to be small. Emacs, which he also names as an inspiration, is proof for that.

The paper's main contribution is the observation that applications with embedded languages can share the same general core language, which thus can be factored out as a library. But only, if the language is easy to extend with application-specific primitive types and values. I believe the success of Tcl is mostly due to its solid design and implementation as a C library, not as a language. It has sparked a wide array of Tcl-enhanced applications, several books, and has its own conference.