A New Interface for HOL - Ideas, Issues and Implementation

Donald Syme

The Computer Laboratory
University of Cambridge
E-Mail: Donald.Syme@cl.cam.ac.uk

Abstract. TkHolWorkbench is a new set of interface tools for HOL implemented using the Tk toolkit. It aims to be robust, extensible, lightweight and user-friendly. The tools are designed to augment the existing HOL interface. The project applies rapid prototyping and the use of an interpreted toolkit to the field of theorem proving interfaces. The topics considered in this paper are: the motivations for a new interface for HOL; the design objectives and usability targets for TkHolWorkbench; a description of the TkHolWorkbench tools as they now stand; and the extensible design architecture used in the implementation.

1 Introduction

This paper describes a new interface for the HOL theorem proving system called TkHolWorkbench. This interface has been under development at the University of Cambridge for the last 6 months, and the author hopes that this interface, or some derivative of it, will eventually become the interface of the HOL2000 project.

The aim of this paper is to give an overview of the functionality of TkHolWorkbench, and to explain some of the motivations and ideas that have contributed to its development. It also seeks to establish a set of *design objectives* and *usability targets* for TkHolWorkbench. A discussion of the extensible design architecture used in the implementation is also included.

TkHolWorkbench is not the first new interface proposed or developed for the HOL system. The most significant efforts have been CHOL by Laurent Thery [11], an unnamed interface by Sara Kalvala [4] and `xhol` by Thomas Schubert [9]. There are, as one would expect, many similarities in motivation and design between these systems and TkHolWorkbench. However, the reader familiar with these systems will see significant differences too.

TkHolWorkbench takes a *tool-oriented* approach to interface design decomposition. The idea is to augment the command line HOL interface with *helper* tools. The tools facilitate various user activities, such as defining a new recursive type or proving a theorem using backward proof. Four tools are described in this paper:

- TkTheoryViewer for browsing theories.
- TkTRS for searching the theory hierarchy for particular theorems.

– TkDefineType for making recursive type definitions.
– TkGoalProof for backward goalstack proof.

Other tools to be included in the upcoming release of TkHolWorkbench are:

– TkNewDef for making new, non-recursive definitions.
– TkRecDef for making primitive recursive definitions.
– TkIndDef for defining inductive relations.

Section 2 of this paper describes some of the motivations for the TkHolWorkbench project. These are made more concrete in a series of design objectives and usability targets in section 3. Section 4 gives an overview of TkHolWorkbench itself and describes the tools mentioned above. Section 5 describes the design architecture of the system, and section 6 looks at how how user extensions can be added to the interface.

2 Informal Motivations

2.1 Real theorem provers deserve real interfaces

The fundamental motivation for building a new interface for HOL is simple: HOL is an excellent theorem proving system, but has a crude, command-line interface. HOL deserves a good interface, or so the reasoning goes, just as it deserved a good implementation in `hol90` [10]. This was summarized succinctly by Laurent Thery [11]: “Real theorem provers deserve real interfaces”.

This seems almost self-evident, but is worth being more specific about the expected benefits of producing a graphical user interface (GUI) for HOL. Good GUIs do not come without cost in time and effort, and, after all, it is always possible that theorem proving will turn out to be one of the application domains where GUIs have little impact. However, the following list of possible benefits indicates why this author believes that HOL users can benefit from a GUI. It draws from several sources [11, 4, 9, 7].

– The current interface to HOL is very weak, particularly in the area of visualisation. *There must be better ways of visualising theory development* than are currently used.
– Theory development by less skilled people will become more common if the goals of formal methods research are realised. HOL *needs to be more accessible* if it is to meet these needs.
– HOL will be *easier to learn* with a good GUI interface. The present text interface provides little help in reducing the steepness of the learning curve.
– Increasingly, *users expect high quality data presentation* to accompany the packages they use. HOL does not currently provide this.
– Also, *users expect a high degree of interoperability between packages.* HOL does not currently provide this.

The following sections will consider in more detail three further motivations for the construction of TkHolWorkbench:

- A GUI will make more functionality readily available to HOL users.
- A good interface can generate new ideas about HOL itself.
- "The time is right", since user interfaces are finally becoming easier to build.

2.2 Making more functionality accessible

It is often claimed that GUIs restrict the amount of functionality accessible to users, in comparison to command line interfaces. However, it is the author's belief that this need not be the case, and that a GUI can make *more* HOL functionality available. We shall consider two examples where this should be the case: the TRS library and the Window Proof library.

The TRS (theorem retrieval system) library was developed several years ago by Richard Boulton [1]. Though well designed and implemented, this library has rarely been used, since it is clumsy to construct the necessary search patterns using a command line interface. TkHolWorkbench includes a simple tool which gives access to this functionality. Adding a GUI interface to this library has made it more accessible, and indeed it is now easy to do dynamic searches through the theory hierarchy.

Alternative proof packages such as the Window Proof library [3] do not seem to get used as often as they deserve. This is probably because of the steepness of the initial learning curve involved. GUI interfaces to such packages should make them easier to learn, and thus more accessible for both new and old HOL users. Window inference seems particularly well suited for use with a GUI. [1]

2.3 Interfaces can generate new ideas

In a research setting, it seems to be the case that a new interface to a tool has the potential to generate new ideas about the research area itself. Already with TkHolWorkbench, users have suggested changes which really require changes to HOL. Typically these relate to higher level actions that users want to perform, yet are not directly supported by HOL. Some examples are the need to support the alteration of definitions, the unloading and reloading of theories, and the management of theory scripts. These represent needs brought to light by the different mental model a new interface provides.

2.4 GUIs are now easier to build

The last motivations we shall mention are pragmatic and personal ones. GUI interface building is simply becoming much more feasible, due to the power brought to developers by tools such as John Ousterhout's Tk/Tcl package [8] and Microsoft's Visual Basic. The use of Tk/Tcl to develop TkHolWorkbench

[1] It is hoped that some kind of window inference tool will be available for use with a future release of TkHolWorkbench, possibly based on work in progress by Thomas Långbacka et al. at Åbo Akademi University[12].

has been instrumental in enabling both rapid prototyping and a robust final implementation.

Lastly, the author has spent much of the last five years developing interfaces for a wide range of systems. A personal desire to apply some of the things I have learnt to my favourite system, HOL, was certainly a significant motivation in this work.

3 Design Objectives and Usability Targets

In this section we follow the methodology of interactive system design described by Newman and Lamming [7] to develop a set of *design objectives* for TkHolWorkbench. These objectives seek to clarify the problem domain with which TkHolWorkbench is concerned. As part of these objectives we establish a set of *usability targets*, which describe the levels of support to be offered by the system.

We will state the design objectives for TkHolWorkbench in terms of the following areas:

- The *human activity* it will support.
- The *prospective users* who will perform the activity.
- The *levels of support* that it will provide, also known as the *usability targets* of the system.

3.1 The Human Activity

In a nutshell, TkHolWorkbench aims to support “theory development”. Contrast this with other definitions we might consider using:

1. TkHolWorkbench will support “backward proof”
2. TkHolWorkbench will support “alternative proof techniques”
3. TkHolWorkbench will support “ML program development”

“Theory development” is a broader concept than (1) and (2) but narrower than (3). There is a deliberate choice being made here. TkHolWorkbench aims to do more than support proof techniques, but does *not* yet aim to be a full ML programming environment, except in the sense that it continues to support the command line HOL interface. Theory development tasks such as backward proof and type/constant definitions are often small and well defined, and seem to be more amenable to improvement using a GUI than arbitrary ML programming tasks. Furthermore, theory development is what HOL is fundamentally concerned with, and so must be the major focus of a user interface.

3.2 The Prospective Users

The prospective users of TkHolWorkbench are, broadly speaking, the current “end-users” of HOL. These presently include people performing hardware and software verification using HOL. Newcomers to the HOL system are implicitly amongst the target user group, because the first tasks a new user performs is invariably some kind of theory development. The user base is not initially expected to include those whose main work is programming new proof tools for HOL.

3.3 Usability Targets

The usability of an interface can be measured in several different ways. One of the most useful bottom line metrics is simple, yet often overlooked by developers: does the interface help users to accomplish their tasks more quickly? Other usability metrics are also important. Those cited by Newman and Lamming [7] are:

- The *incidence of errors* while performing the activity.
- The user's *ability to recover from errors* that occur.
- The magnitude of the user's task in *learning to use the system.*
- The user's *retention of learned skills.*
- The user's *satisfaction* with the system.
- The user's ability to *customize* the system.
- The *mental burden* placed on the user when using the system.

For these usability metrics it is easy to develop a set of usability targets for TkHolWorkbench. In particular:

- TkHolWorkbench must make theory development faster. For instance, theorems should be quicker to formulate and prove. Activities performed with the graphical tools should not take longer than with the command line interface.
- Errors must not be more common than with the command line interface, and must it be easier to recover from errors when they are made.
- TkHolWorkbench must be comparatively easy to learn. It is unrealistic to expect people who know the HOL logic to need more than about 60 minutes learning time, spread over the first days of use.
- Users must be able to customize TkHolWorkbench to their needs.
- TkHolWorkbench should reduce the mental burden of using HOL.

These targets provide a mechanism for measuring the success of an implementation, and also contribute to the formation of design guidelines that help the developer during the design and implementation process.

This paper does not evaluate TkHolWorkbench against these targets, but rather states them as goals to be achieved in current and forthcoming development. The field of Human Computer Interaction provides methodologies for performing usability evaluations such as Cognitive Walkthroughs and Keystroke-Level Analysis. Such methods will be used to evaluate TkHolWorkbench rigorously in the future.

4 An Overview of TkHolWorkbench

In this section, we move on to describing TkHolWorkbench itself. We shall consider some of the design principles which have featured in its development so far, and then some particular features of TkHolWorkbench.

From the user's viewpoint, TkHolWorkbench can be seen as set of *independent yet cooperating tools*. This means:

- The interface is a series of optional modules, exhibiting a low degree of interdependence.
- Tools do have to be used if the user does not find them helpful.
- Tools support standard X-Windows interaction techniques.

More will be said in section 5 on the internal mechanisms used to support this. TkHolWorkbench is also designed using the principle of *non-intrusiveness*. This means:

- The interface supports existing modes of interaction. The user is not forced to adopt the new modes offered.
- No end product of the interface relies on the interface itself. For example, proof scripts do not rely on tactics which are peculiar to the interface. Scripts can be rerun without the interface present.
- Running TkHolWorkbench does not have any side effects except those exhibited by running a normal HOL session.

Lastly, TkHolWorkbench is implemented using the principle that *lightweight* systems are more likely to meet their usability targets. This is with regard to installation time, tool startup time and runtime memory usage. The idea here is to balance resource usage against the functionality provided.

4.1 The Command Line

An important feature of TkHolWorkbench is that *the standard HOL interface is still available to the user*. In terms of implementation, a TkHolWorkbench program has a Unix stdin/stdout which behaves identically to a normal HOL process. A screen picture illustrating this is shown in figure 1. This feature has important ramifications:

- Users can start TkHolWorkbench as a substitute for their regular HOL session. The only difference is the appearance of a window in addition to the HOL prompt.
- Existing users are not forced to change their work patterns.
- TkHolWorkbench can be used from within editors like Emacs in the same fashion as the normal HOL interface.
- The interface developer is relieved of a great burden, as the interface no longer needs to provide access to *all* of the HOL system. The developer can concentrate on developing tools aimed at specific needs.

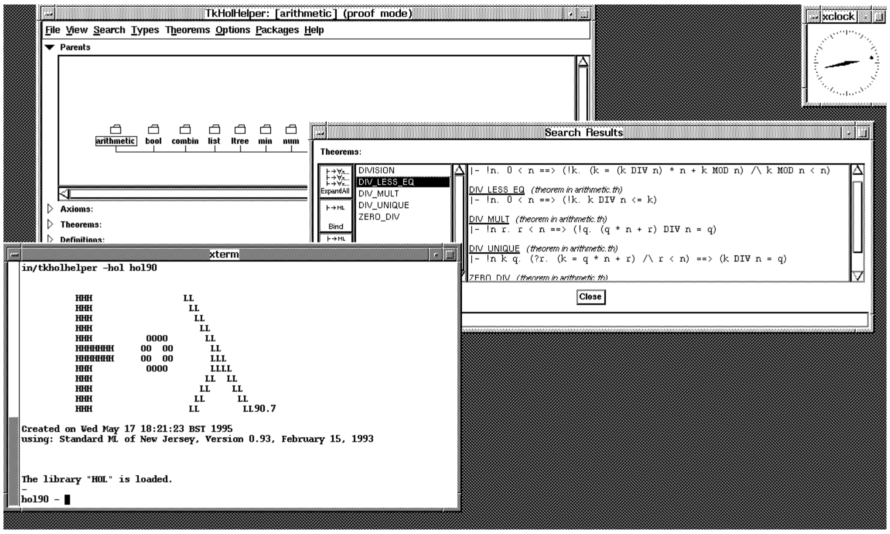

Fig. 1. TkHolWorkbench supports the existing HOL interface

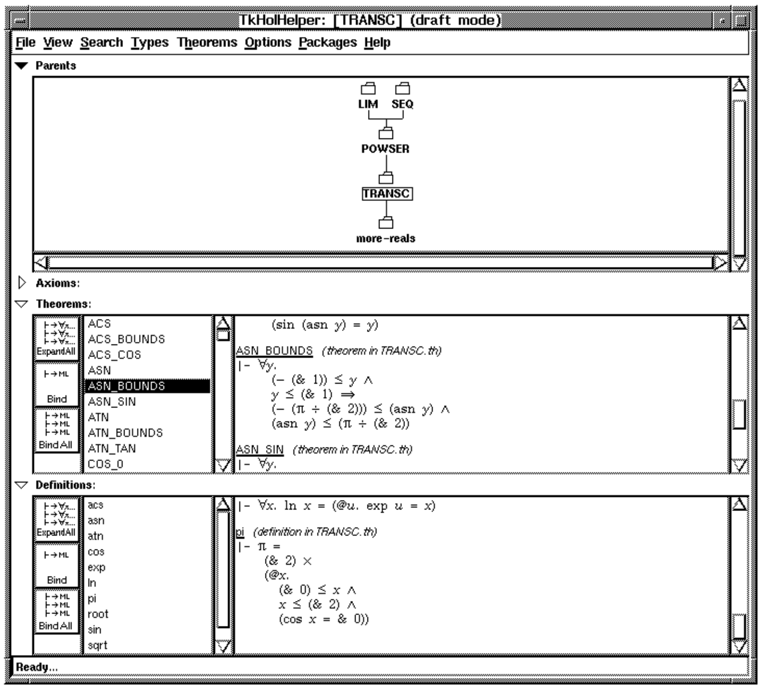

Fig. 2. Browsing the `reals` library with TkTheoryViewer

4.2 TkTheoryViewer

We now introduce some of the tools in the current release of TkHolWorkbench. TkTheoryViewer and TkTRS are both *visualisation* tools. This means they are primarily concerned with representing data to the user, rather than allowing state transitions. The primary role of TkTheoryViewer is to visualise two things:

- The theory graph of the theories loaded into the HOL session.
- The theorems, definitions and axioms of a selected theory.

More than one TkTheoryViewer may be created, if the user wants to view multiple theories simultaneously.

TkTheoryViewer provides an ideal mechanism to browse unfamiliar theories. This is a common task for both new and old HOL users. An example of this is shown in figure 2, where TkTheoryViewer is being used to browse through the `reals` library. The tool also makes an excellent proof assistant tool, as it is an easy process to look through a theory for a theorem which may be needed for a proof.

The theory graph of the theories loaded into the HOL session is visualised as a tree. This behaves in a manner similar to a file browser directory hierarchy:

- Nodes of the tree may be expanded and collapsed.
- A set of theories may be selected for operating on. In particular, the selected set of theories may be used as the basis of a TkTRS search.

Given that the theory structure in HOL is a graph it may seem odd that a tree is used to represent the information. Initially this was done for ease of implementation, but as it happens it seems to be an adequate and efficient representation, even for large theory structures.

The theory graph is also somewhat modified, allowing the tree to be "flattened" at certain points. It is particularly useful to flatten the tree at collector theories such as `HOL`, since the details of the theory structure above this point is not normally important. This feature can be inhibited if the user desires.

The theory graph automatically updates its contents when a new theory or library is loaded into the system. This happens even if the load command was issued from the command line interface. The mechanism used to implement this functionality is described in section 5.5.

TkTheoryViewer displays the contents of the currently selected theory in the windows below the theory graph:

- Theorems may be displayed in *Rich Text*. This uses different fonts and colors for various components of terms.
- Each of the windows for axioms, definitions and theorems are collapsible, and when no objects of a particular kind occur in the theory, the window is automatically collapsed. This minimizes the amount of screen real-estate used by the viewer.

The theory contents update automatically when theorems are saved or new definitions introduced.

There are several other actions available from TkTheoryViewer, as it is the normal main window of a TkHolWorkbench session. In future releases it will be possible to locate these actions on any top level window, as most of them are not directly related to the visualisation of theories. These actions include:

– Access to theory operations: loading a theory; adding a new parent; creating a new theory; and switching between proof mode and draft mode.
– Loading a HOL library. The list of all libraries available to be loaded is shown to the user.
– Loading an ML file.
– Saving the image of the HOL session.
– Access to other packages such as TkTRS and TkGoalProof via configurable menus.
– Access to a database of configuration options for TkHolWorkbench.
– Access to the values of global variables and flags in the HOL session.
– Help menus which use a WWW browser to access local or remote help files.

4.3 TkTRS

The TkTRS tool provides a simple interface to Richard Boulton's Theorem Retrieval System (`trs`) library [1]. The `trs` library provides the following functionality to the user:

– The ability to search loaded theories for theorems and definitions.
– Search patterns may be based on names or theorem structure.
– Arbitrary lists of theorems may be searched as well.

The TkTRS tool provides fairly primitive access to a subset of this functionality. Figure 3 shows an example of using TkTRS to find all the theorems concerning division in all the loaded descendants of the `arithmetic` library.

In the top windows of the search window, the set of source theories to search is shown in the right hand list, and all other theories are shown on the left. Theories may be transferred to and from these lists.

The search pattern is specified in the lower boxes. It may be any of the following:

– Find theorems which contain a particular term structure.
– Find theorems whose name matches a particular wildcard pattern.
– Find theorems which match an arbitrary `trs` pattern.

The search results are presented in a separate window, with similar properties to the windows used to display theorems in the TkTheoryViewer.

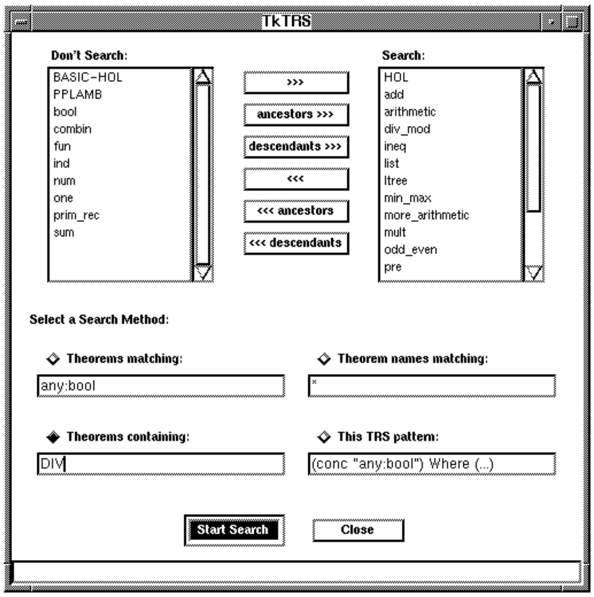

Fig. 3. Searching descendants of `arithmetic` for theorems about division

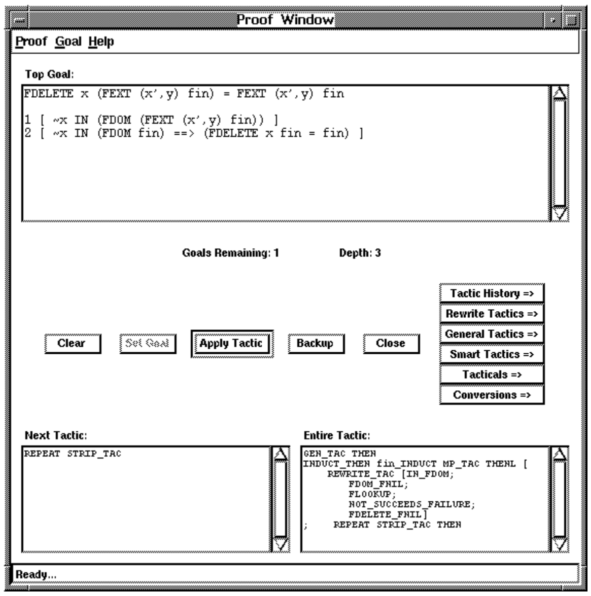

Fig. 4. Using TkGoalProof

4.4 TkGoalProof

The next two tools are examples of constructive tools. They support the user activities of backward goal proof and concrete type definition.

TkGoalProof is essentially an interface to a HOL goal stack. The operations conducted in the window will be familiar to anyone who has performed a backward proof in HOL:

1. The user enters the goal in the top window, and selects "Set Goal".
2. The user enters tactics into the "Next Tactic" window and selects "Apply Tactic". The goal is redisplayed after each application of the tactic.
3. As the user applies tactics, a complete tactic is constructed automatically in the "Entire Tactic" window.
4. The user may backup through the application of tactics. The entire tactic is adjusted accordingly.
5. After the theorem is proved, the user is prompted for a name by which to save the theorem. A script is then displayed by which the theorem may be reproven in a later HOL session.

As with all TkHolWorkbench tools, more than one TkGoalProof window may be created. Each window accesses different goal stacks, and hence extra windows can be used to prove lemmas needed for the current proof.

It is the author's experience that TkGoalProof is sufficiently useful to replace command line backward proof. However, proofs are still carried out in conjunction with forward proofs, which are performed at the command line. In this scenario the command line becomes a simple evaluator for ML phrases. These observations are based on about 50 proofs performed over a period of 3 days, while developing a theory of finite maps. The goals themselves were not large, although the proofs were sometimes quite complicated.

4.5 TkDefineType

The TkDefineType tool provides an interface to the functionality of Tom Melham's recursive types package [6]. An example of the results of using the tool to define a binary tree type `btree` is shown in figure 5.

The type grammar is entered in the top window, and a set of operations to perform with regard to the grammar is selected in the lower window. After the operation has been completed, the theorems and definitions which have been produced are displayed in a window, along with a script which will reproduce the operation in a later HOL session.

It is hoped that similar tools will soon be available to allow users to access other type definition packages. The extensibility of TkHolWorkbench means that these packages could come with their own interface tools.

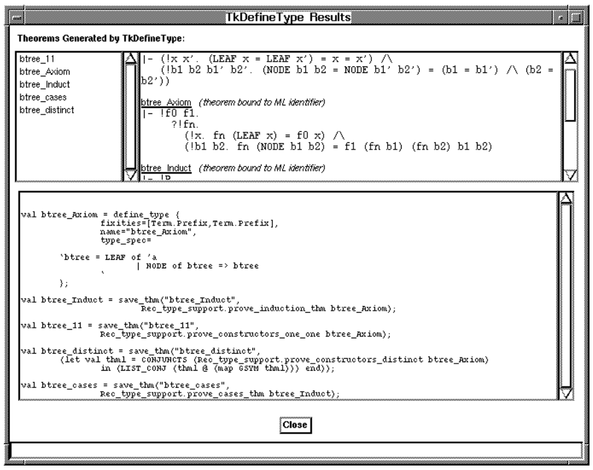

Fig. 5. The results window after defining `btree` using TkDefineType

5 Implementation Techniques

This section discusses some of the implementation techniques and the design architecture used in the current release of TkHolWorkbench. The two most important features of the architecture are its *modularity* and its *extensibility.* An understanding of the techniques used will benefit anyone wanting to extend TkHolWorkbench with a new tool.

5.1 Expect, Tk and HOL

TkHolWorkbench is implemented using Tk/Tcl [8]. The architecture of TkHolWorkbench is not dependent on the implementation language. However, it is facilitated by it, and hence the reader unfamiliar with Tk and Tcl may find the following facts helpful.

Tcl (Tool Command Language) is a small, interpreted language developed by John Ousterhout at Berkeley [8]. It takes a "nihilistic" approach to language design, allowing arbitrary additional command constructs and using string values only. It is sufficiently efficient and flexible for many purposes, though much of its value lies in the many extensions which have been built on top of it.

Tk is a C extension to Tcl which implements a widget set for the X window system. The standard Tk interpreter is the program `wish`.

TkHolWorkbench is implemented by the well known technique of driving the HOL session as a slave process of the Tk/Tcl process. The two process are run in tight synchronisation, and cross-language calls from Tcl to ML can be made. Thus, for most practical purposes, the two processes may be considered one. The implementation of this mechanism uses "Expect" [5], a process control extension to Tcl. Expect is specifically designed for the control of command line processes such as HOL.

5.2 Separating out the User Interface

The most basic notion of design separation used in TkHolWorkbench is that of dividing the program in two halves - the "computation engine" and the "user interface". Though often touted as a "good idea", this ideal is often ignored in practice. TkHolWorkbench tries to adhere to it strictly.

The basic relation between the components is client-server. The computation engine acts as a "data server", and the user interface is a client of the services it provides. In the case of TkHolWorkbench the computation engine roughly corresponds to the HOL system itself, and the user interface to all those components written in the Tcl language.

When such a division is made, the question naturally arises as to exactly what goes where. An obvious test of separation is whether it is feasible for the computation engine to be run in an environment where the windowing system is not present. Another good rule is that all non-interface related functionality used by interface components should lie in the HOL system, preferably in HOL libraries. These libraries should be available for all to use, including other interfaces.

A common problem is that the computation engine ends up being "incomplete" without the interface being present. The approach used in TkHolWorkbench is to treat the user interface as little more than a fancy method of performing transitions in the underlying system. Naturally, the interaction techniques used may themselves be complex, but the end result of an action needs to be series of well defined transition in the underlying computation engine. The end result of a TkHolWorkbench action must be ML code which can be executed without the interface being present.

5.3 Modules in the User Interface

Modularity can be difficult to achieve in user interfaces. The basic tension is often between modularity and interoperability.

In the GUI setting, interface components will usually be widgets. Some of the basic principles of designing widgets are:

- Each widget must be given a well defined, unambiguous purpose.
- Widgets which visualise data that changes should not store their own data. Instead they should "reflect" data that comes from an external data source. Often this data will be stored in the computation engine.
- Widgets should be configurable. The Tk toolkit is a good set of examples of configurable widgets.
- Widgets should interoperate by standard techniques where possible. These include clipboards, selections, drag-and-drop and modality. This will reduce the need for dependencies between widgets.

By applying these rules carefully, the implementation of TkHolWorkbench has turned out to be highly modular. This has resulted in a set of reusable, configurable widgets from which to build other tools.

5.4 Modules and Dynamic Loading

The architecture of TkHolWorkbench allows modules containing new tools and widget types to be *dynamically loaded*. This is achieved primarily by the autoloading facilities available in Tcl. The benefits of dynamic loading are enormous. It makes rapid development far easier. Also, it removes a considerable obstacle to achieving extensibility - the need to recompile or reconfigure the system to include a new module. Extensibility is covered further in section 6.

5.5 Anonymous Notification of State Change

One of the principles of widget construction listed in section 5.3 was that widgets should rarely store their own data but should instead "reflect" data from an external source. This, however, begs the following question: how do widgets know when the data they are representing has changed? An example of this problem is the case of the theory tree display in the TkTheoryViewer tool. An interface should update this display when the theory tree changes. Note that a change in the data may be induced by some external source, for instance from an interaction at the command line prompt.

The solution to this problem lies in expanding the range of services provided by the computation engine, i.e. HOL, to include *notification*. In particular:

- The computation engine must include facilities to notify clients, i.e. interface components, when changes occur in the state of the engine. Where possible, *all* items of state which may be dynamically changed should be traceable in this fashion.
- This notification must be anonymous, in the sense that the computation engine must allow runtime, rather than compile-time, binding of clients. This is necessary if the integrity of the computation engine is to be maintained.
- More than one client must be able to register its interest in each data item.

Thus, to solve the problem of maintaining an up-to-date representation of the theory graph, TkHolWorkbench uses modified versions of several side-effecting functions in the HOL system. These functions support the registration of notification clients. Ideally this kind of functionality should be built into the core of HOL.

6 Extending TkHolWorkbench

6.1 Packages and User Extensions to TkHolWorkbench

The last section described how TkHolWorkbench can be thought of as a set of essentially independent modules (or objects). In TkHolWorkbench jargon, the code implementing a set of object classes is called a "package". Packages usually contain several files of Tcl code. Packages are very similar to HOL libraries, and indeed it is conceivable that HOL libraries can act as packages. Often a package will implement a new tool, just as a HOL library will contain a theory.

In the implementation of TkHolWorkbench, each component is simply one such package. The program `tkhol` is just an invocation of a small Tcl program which runs several packages together.

Extension is a simple process. Additional packages may be specified from the command line:

```
> tkholhelper -package ~/mytool -win MyTool
```

The `-package` argument specifies a directory which contains the package, and the `-win` argument specifies that a toplevel window of the given class should be created at startup. In Tcl, this means a procedure "MyTool" which creates a toplevel instance of the tool should be defined somewhere in the package. All the functionality of MyTool will have been implemented somewhere in the package `mytool`.

It is also possible to load packages dynamically from a menu within the system, or from the HOL command line. In future versions it is hoped that the interface will be able to detect and automatically load packages provided in HOL libraries.

The author does *all* development of new tools as package extensions. Like HOL libraries, packages can come with documentation and online help and so can provide facilities upon which other tools can be built.

6.2 Adding a Virtual Theory Controller

To test this functionality, a small extension was added to TkHolWorkbench by the author and Paul Curzon working together. The idea was to write a small interface to some "virtual theory" functionality being developed by Curzon [2]. TkHolWorkbench comes with an example "skeleton" package, which was used as a starting point. Within two hours, an interface had been developed from existing widgets which controlled the current "virtual theory" being accessed. The ease with which this package was created and integrated illustrates the potential that an extensible architecture holds.

7 Summary

This paper has presented the motivations, objectives and current status of TkHolWorkbench. A rigorous evaluation of TkHolWorkbench has not yet been performed. However it is the author's belief that it is well on the way to meeting the design objectives and usability targets outlined in section 3. It is hoped that given time it will become an accepted part of a standard HOL environment under X Windows.[2] Already it is a useful tool, and future development should only

[2] Note that Tk/Tcl is also supported on MS Windows platforms, which should soon be able to support HOL.

improve the situation. Importantly, the architecture has been both practical and extensible, and facilitates rather than hinders development.

It would seem that the user activity within HOL is surprisingly complicated, incorporating, as it does, arbitrary programmability. TkHolWorkbench does not yet aim to support all HOL user activities. This may become a significant issue in future development. The HOL community is increasingly paying more attention to the larger issues of theory development, and interface issues are very relevant to this area.

Future work on TkHolWorkbench will concentrate largely on using the extensibility of the product to develop interface support for language embeddings in HOL. A further test of the TkHolWorkbench architecture will come with some cooperative work on a Window Inference package being planned with Thomas Långbacka et al. at Åbo Akademi University [12].

References

1. Richard Boulton. *The HOL trs library - A Theorem Retrieval System.* From the HOL88 system distribution.
2. Paul Curzon. Virtual theories. Submitted to the 8th International HOL Workshop on Higher Order Logic and Its Applications.
3. Jim Grundy. *The HOL window library.* From the HOL88 system distribution.
4. Sara Kalvala. Developing an interface for HOL. *Proceedings of the 1991 HOL Workshop*, 1991.
5. Don Libes. *Exploring Expect.* O'Reilly & Associates, January 1995.
6. Tom Melham. Automating recursive type definitions in higher-order logic. In G. Birtwhistle and P.A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automated Theorem Proving*, pages 341–386. Springer-Verlag, 1989.
7. William Newman and Mik Lamming. *Interactive Systems Design.* Addison-Wesley, December 1995.
8. John Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, April 1994.
9. Tom Schubert and John Biggs. A tree-based, graphical interface for large proof development. *Supplementary Proceedings of the 1994 HOL Workshop*, 1994.
10. Konrad Slind. An implementation of higher order logic. *Master's Thesis*, January 1991. Research Report No. 91/419/03, Department of Computer Science, University of Calgary.
11. Laurent Thery. Real theorem provers deserve real interfaces. In *Software Engineering Notes*, volume 17. ACM Press, 1992.
12. Rimvydas Rukšėnas Thomas Långbacka and Joakim von Wright. TkWinHOL - a tool for doing window inference in HOL. Submitted to the 8th International HOL Workshop on Higher Order Logic and Its Applications.

This article was processed using the LaTeX macro package with LLNCS style