TAS and IsaWin: Generic Interfaces for Transformational Program Development and Theorem Proving

Kolyang, C. Lüth, T. Meyer, B. Wolff

Bremen Institute for Safe Systems (BISS), FB 3
Universität Bremen, Postfach 330440, 28334 Bremen
{kol,cxl,tm,bu}@informatik.uni-bremen.de

1 Introduction

We present a new approach to the implementation of graphical user interfaces (GUIs) for formal program development systems like transformation systems or interactive theorem provers. Its distinguishing feature is a generic, open system design which allows the development of a family of tools for different formal methods on a sound logical basis with a uniform appearance.

The context of this work is the UniForM project [KPO+95], the aim of which is to develop a framework integrating different formal methods in a logically consistent way. Consistency is achieved by encoding formal methods such as CSP and Z in the theorem prover Isabelle [Pau94], which is used to perform the program development as well as to prove the correctness of the transformation rules. One of the main UniForM objectives is to enable non-expert users to actually perform at least part of the development themselves. Hence there is a crucial need for an encapsulation technique of these Isabelle encodings providing a generic way of building graphical user interfaces.

2 System Architecture

2.1 Isabelle and sml_tk

The system is entirely implemented in Standard ML (see Figure 1). The main reason for this is Isabelle's system architecture, and ML's powerful modularization concepts.

Since Isabelle essentially consists of a collection of ML types for objects such as theorems, proofs and rule sets, and ML functions to manipulate these objects, organised into a collection of ML structures and functors, one can conservatively extend Isabelle by writing ML functions, using the abstract datatypes provided by Isabelle.

To implement the graphical user interface, we are using the interface description and command language Tcl/Tk, encapsulated into Standard ML by the sml_tk package [LWW96](also developed at the University of Bremen). This package provides abstract ML datatypes for the Tcl/Tk objects, thus allowing the programmer to use the interface building library Tk without having to program the control structures of the interface in the untyped, interpretative language Tcl.

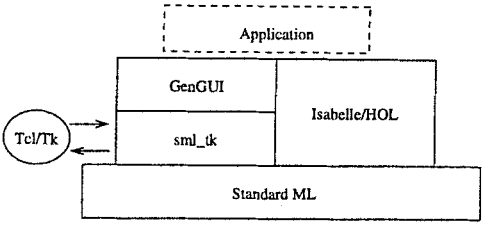

Fig. 1. System Architecture

2.2 The Generic Graphical User Interface GenGUI

GenGUI builds on the interface description facilities provided by sml_tk to provide a generic graphical user interface. Its main components are a module allowing the user to manipulate *items* (graphical objects) on a *canvas* (a window area to draw on) by *grabbing* them with a cursor, moving them across the screen or *dropping* them onto other objects (thereby possibly triggering an operation), and a module giving a semantics to these items with respect to a given application, which can be abstractly characterised as follows:

- It has *objects*, each of which has a *type*. The type determines which operations are applicable to this object, and is indicated by the object's *icon*.
- For each object type, the application provides a dictionary of unary operations, such as a function to display the object;
- For all pairs of object types, there is a binary operation, the result of which is an object produced by the operation.

Hence, applications are described by an ML signature `APPL_SIG`, and the Generic GUI is implemented as a functor

```
functor GenGUI(structure appl: APPL_SIG ) = ...
```

which provides a graphical user interface for each application.

2.3 Generic Visual Appearance

The main window of any GenGUI instance consists of two areas: the *assembling area* in the upper part, and the *construction area* in the lower part. The assembling area contains the icons representing the available objects. They can be dragged, moved and dropped onto each other, affecting the binary operations described by the application. Each object offers a pop-up menu of the available unary operations.

Each application is geared towards one particular type of objects, called the *construction objects*. These objects (and only those) can be manipulated in the construction area. Here, the state of the object under construction is displayed. Moreover, it can be altered, in contrast to the drag&drop operations in the assembling area which leave the involved objects untouched.

The application determines the visual appearance of the icons, the size of the window, and the details of the construction area. It can also add elements such as menus or buttons to the window.

We will now present two example applications: the Transformation Application System TAS, and the Isabelle graphical user interface IsaWin.

3 The Transformation Application System TAS

TAS is a system for transformational program development in Isabelle (for theory details and background see [KSW96]). It is designed to keep everything about proofs in Isabelle away from the user. The proof obligations resulting from applying a transformation rule are proven using another Isabelle tool (like IsaWin below), such that the user does not have to worry about the details of how the transformational process is implemented within Isabelle, leaving him with the main design decisions of transformational program development: which rule to apply, and how to instantiate its parameters.

The object types of TAS are transformational program developments as construction objects, transformation rules with their parameters not instantiated, transformation rules with their parameters instantiated, and parameter instantiations. (With typical transformation rules, parameter instantiations are lengthy enough to merit a dedicated object type to avoid having to retype them, allow copying them etc.) The operations include applying a transformation rule by dropping it onto a transformational program development, and instantiating the parameters of a transformation rule by dropping the instantiation on the rule.

Figure 2 shows a screenshot of TAS. In the upper part of the screen, the assembling area shows a collection of icons. The construction area in the lower part of the screen shows the current transformational development.

4 IsaWin— a Graphical User Interface for Isabelle

IsaWin can be used as an interface to Isabelle in its own right, as well as to prove the proof obligations arising from transformational developments using TAS, or even the correctness of the transformations of TAS.

Its object types are theorems, proofs, two types of rule sets, and theories (collections of type declarations, theorems and rule sets). The construction objects are proofs; once a proof is finished, it can be turned into a theorem. The operations include backward resolution by dropping a theorem onto a proof, forward resolution by dropping a theorem onto a theorem, or rewriting by dropping a rule onto a proof.

Figure 2 shows a screenshot of IsaWin. The assembling area doesn't look that different from TAS, but the construction area is far more elaborate, offering the user control over the various Isabelle tactics. The assembling area will not hold all theorems known within Isabelle when starting up, because there are too many. Rather, the user is provided with a theorem and theory browser with which he can select the relevant theorems and place them on the assembling area.

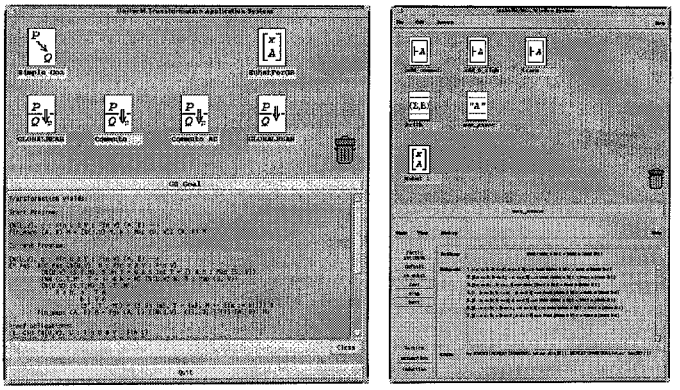

Fig. 2. Screenshots of TAS (on the left) and IsaWin (on the right).

5 Related and Future Work

Pioneer transformation systems include PROSPECTRA [HK93] and KIDS [Smi91], but we believe that they are too monolithic and difficult to change. Our approach offers a greater flexibility, thus allowing extendability and reusability.

Other GUIs for specific theorem provers like TkHOL and XIsabelle are implemented in Tcl, which lacks the powerful modularization concepts of ML, and consequently do not have the generic, open system architecture allowed by our approach.

The main emphasis during development has been put on a clear and generic system architecture rather than bells and whistles. Having achieved the former, we are going to concentrate on the latter, and are going to implement extensions such as better error handling, pretty printing (using mathematical notations) and focusing (applying a transformation rule or an Isabelle tactic to a subterm of the current goal, leading to the concept of a generic focus) in the near future.

References

[HK93] B. Hoffmann and B. Krieg-Brückner. *Program Development by Specification and Transformation.* LNCS 690. Springer Verlag, 1993.

[KPO+95] B. Krieg-Brückner, J. Peleska, E.-R. Olderog, D. Balzer, and A. Baer. UniForM Workbench — Universelle Entwicklungsumgebung für formale Methoden. Technischer Bericht 8/95, FB 3, Universität Bremen, 1995.

[KSW96] Kolyang, T. Santen, and B. Wolff. Correct and user-friendly implementations of transformation systems. In M. C. Gaudel and J. Woodcock, editors, *Formal Methods Europe '96*, LNCS 1051, pages 629– 648. Springer Verlag, 1996.

[LWW96] C. Lüth, S. Westmeier, and B. Wolff. sml_tk: Functional programming for graphical user interfaces. Technical Report 7/96, FB 3, Univ. Bremen, 1996.

[Pau94] L. C. Paulson. *Isabelle - A Generic Theorem Prover.* Number 828 in LNCS. Springer Verlag, 1994.

[Smi91] D. R. Smith. KIDS — a semi-automatic program development system. *IEEE Transactions on Software Engineering*, 16(9):1024– 1043, 1991.