UserInteractionWare Tasks

Lambert Meertens

Version of June 4, 1998

(Preliminary version, just some thoughts jotted down ...
The tasks identified below are not necessarily sequential, but may be ongoing tasks performed concurrently.)

1 I/O trial

Express about the simplest possible interactive program in Slang and generate working code. For example, The Oracle is a program that consists of an infinite loop where each cycle reads one line of text from the input stream and then prints "*Yes*" or "*No*" — depending on whether the input contained an even or an odd number of occurrences of the character "*n*" — to the output stream.

The purpose of this subtask is to clarify what is involved in writing interactive Specware-generated applications (without the burden of the complexity of graphical interfaces and multi-modal input).

2 GUI I/O library

Design a library at an appropriately high level for creating graphical presentations and obtaining input events.

In a first version, composite presentations can be built from other presentations using a boxes-&-glue model as in Latex or tcl/tk, as well as in the form of graphs with (labelled) boxes for the nodes and a few kinds of (also labelled) arrows for the arcs. In later versions we will expand the composition mechanisms and use approximate-constraint solving for easier positioning control.

Depending on the GUI toolbox chosen there may be a need for buffering and incremental repainting, not only for speed of screen updates but also to avoid unnecessary discontinuity (such as screen areas going temporarily grey). In a first version, we can just ignore this and do a full repaint always.

A full library should accommodate about all conceivable GUI bells and whistles.

Provided there is a reasonable initial design, new features may however be added in the course of time as the need for them arises.

The library should have a Lisp API (or any other reasonable Specware target language). This can just consist of invocations of RPC/RMI stubs; the real code could be Java, C or whatever.

The selection of an appropriate GUI toolbox to build upon is a subtask. Factors to be considered are not only the functionality offered and ease of use, but also the long-term viability (in as far as foreseeable), including continued maintenance and support, as well as platform-independence. However, as long as the library is appropriately high-level, switching to a different toolbox later should be a job of moderate size.

On the input side, the GUI I/O library should already map "raw" screen events (e.g. mouse clicks) to events that relate to the higher-level entities presented on the screen.

3 GUI I/O trials

Create a few simple interactive programs in Slang with graphical I/O. For example, a bar-chart editor in which the user can adjust the heights of the bars by dragging, but also by editing a textual presentation of these heights, and a finite-state automaton simulation displaying state changes (e.g. as a token travelling between nodes) in response to user actions.

These trial programs should use ad hoc methods for coupling the (almost trivial) application core functionality with the user interaction.

4 Separation of responsibilities

Modify the I/O trial programs to separate the specification of the application core functionality from the specification of the "user-interaction mediator", a component that can differ from program to program and is still specified in an ad hoc way.

5 Experience building

Create more challenging interactive programs, where possible meeting existing needs within other projects in Kestrel, while keeping the specifications of the application core functionality and the user-interaction mediators strictly separate.

6 Presentation and constraint-maintenance

Design a formalism to make the task of specifying the visual presentation of entities more easy and uniform. Use this in the course of the experience-building process. Expand the constraint-maintenance mechanisms.

7 Towards UserInteractionWare

Examine the collection of user-interaction mediators built up, and extract the commonalities (generalization–abstraction–parametrization). Design an appropriate UIM formalism for expressing user-interaction mediators tersely, separating the coupling aspect per se from more superficial presentation aspects. Use existing interactive programs to test this UIM formalism out.

8 And then ...

Taxonomies of user-interaction models and paradigms. Taxonomies of interactor subtheories. Interactively designer-adjustable UI's. ...