DylanWorks User Interface Substrate

Scott McKay & Roman Budzianowski

1. Introduction

The purpose of this paper is to describe the goals and overall strategy for the DylanWorks user interface substrate. It is not the purpose of the paper to describe, in detail, the contents of the libraries that implement the user interface substrate.

At the highest level, the user interface substrate will be divided into three distinct layers, all of which are usable by any end-user:

- A binding to the native user interface system (at least its graphics, color, window management, and event management functionality). This will be implemented via the FFI (possibly using Creole), and will not be portable between different Dylan platforms.
- A portable, higher-level interface to the native bindings. This layer deals mainly with the coarse-level of an application, and implements roughly the same functionality as the lower layer: graphics, color, windows, and events. It will also contain a facility for managing *application frames*, which consist of a set of *panes* arranged in one or more *layouts*. Frames can be embedded within other frames, as in OpenDoc and Ole. This layer corresponds closely to the Silica layer of CLIM, or to Fresco.
- A portable, high-level toolkit for constructing the contents of an application, based on its semantics. The functionality includes streams, output recording, formatted output, incremental redisplay, presentation types and presentations, input editing, context-sensitive input and help, and a high-level command processing model. This high-level functionality will also model such things as embedded applications.

Note that it is not the intention of our user interface substrate to hook up to every bit of functionality in the native toolkit(s). For example, our user interface substrate will not support every bit of functionality found in the Microsoft Foundation Classes (MFC); we intend that the full functionality of MFS will be accessible via Creole.

2. Requirements

- Native look and feel.
- Integration with OpenDoc and/or Ole.
- Portability, at least at the two top levels.
- Good performance at the high level layer, and high performance at the lowest layer (it should be virtually indistinguishable from the performance of the native libraries).

3. The Layers

3.1 Native Toolkit Binding (low) Level

This level will implement a clean FFI (and possibly Creole) based binding to the required native toolkit functionality. Initially, this will be a binding to the Win32 user interface API.

Note that we will bind to a subset of all the possible functionality we could interface to. Users requiring additional functionality will simply use Creole or the FFI.

3.2 Gadget and Event (middle) Level

The level will implement a portable interface to the lower level. It is intended that users can program using this level to achieve the effects of programming in a traditional, compositional toolkit, except that the resulting code will be portable. Programming at this level will produce look-and-feel independence — the resulting interface will use the look-and-feel of the underlying toolkit.

The main decision to be made here is, what model to use. Candidates include CAPI (the least likely candidate), Silica, or Fresco. The Silica model has the benefit that the higher layer is known to work well on it. On the other hand, Silica is not particularly well-known, or very strong on embedded applications. Fresco is more modern, but there is the problem that we have to either interface to Fresco after writing a Fresco back-end, or we have to implement it entirely ourselves.

Note that we do not propose to directly use the Win32 API as the middle layer, because it is not sufficiently abstract to describe other UI models, nor is it at a high-enough level to provide leverage to programmers who wish to do things beyond the most primitive gadgets-and-events programming.

Functionality at this layer includes (using Silica terminology):

- A model of simple regions (rectilinear “boxes” and points).
- A separable module for more sophisticated geometric regions (general polygons and polylines, ellipses, and so on).
- Affine transformations.
- Basic support for color, line styles, and text styles.
- Basic support for raster images.
- Basic support for color maps, and so forth.
- Sheets (called “glyphs” in Fresco), including geometry management and parent/child management.
- Event management.
- Repaint management.
- Mediums.
- Ports and grafts.

- Mirrored sheets.
- Gadgets.
- Panes (that is, sheets that obey layout).
- Basic drag-and-drop.
- Frames, including basic sujpport for embedding.

3.3 Semantically-based (high) level

The high level layer is modelled on CLIM, but we plan to learn from CLIM's mistakes. In particular, the inking model (here and in the middle layer) can be simplified, output records can be unified with Silica sheets and Fresco glyphs, and we can divide the functionality at this level into separable modules.

Functionality at this layer includes (using CLIM terminology):

- Input and output streams.
- Output recording.
- Incremental redislay.
- "Viewers" and notification of viewers when data changes.
- Table formatting.
- Graph formatting.
- Presentation types, input contexts, and presentations; views.
- Input editing and completion.
- Menus (including integration with native look-and-feel).
- Dialogs (including integration with native look-and-feel).
- Semantically-based drag-and-drop.
- Commands and command processing.
- Frames (again), including more support for embedding.
- Scripting.

4. Other Functionality

4.1 Graphics Programming

By graphics programming we mean 2-d or 3-d drawing of geometric shapes.

Graphics programming doesn't have to deal with the look-and-feel issues and as such is simpler to tackle. We could have our own portable graphics model for simple programming (and for our own

use). As the API we have a choice of several popular models (some subset of Postscript is attractive). We can also interface with third party libraries like OpenGL.

4.2 Multi-media

This includes support for pictures, audio, video and animation. We may also support unusual UI peripherals as speech I/O, Polhemus-like 3 and 6 D input devices, handwriting recognition [especially the new “graphiti” style character recognition] and stereo graphics. None of these are release one requirements.