AbstractUI::ML and The Construct Basics

Ánoq of the Sun, Hardcore Processing *

May 19, 2023

Abstract

This document gives a quick overview of AbtractUI::ML, which is a crossplatform GUI toolkit for Standard ML capable of handling things like pluggable Look and Feel. The basic architecture is presented and a discription of the motivation for some of it's design is given. The Construct which is a GUI builder for AbstractUI::ML is also briefly presented.

Introduction

With UI standing for user interface, then by definition, AbstractUI::ML is a UI-concept independent UI-API in `Standard ML` currently with crossplatform classical windowing system implementations for `WindowsNT/95/98`, `Linux/X-Windows` and other Unixes. (yes, that's a long definition... :). AbstractUI::ML is distributed under the GNU General Public Library Licence (`LGPL`), which (in short) allows you to build commercial applications with it, as long as you distribute all the sourcecode and copyrightnotices for AbstractUI::ML in unmodified form along with your software.

The Construct is a GUI builder for `AbstractUI::ML` which can import and export `SML'97` code ready to be compiled into an application. Actually it is the first GUI builder ever written in `Standard ML`! The Construct is distributed under the GNU General Public Licence (`GPL`), which (in short) only allows you to write any extensions for The Construct if you distribute all the sourcecode for your extensions under a `GPL` compatiple licence. However you may of course use the user interfaces generated with The Construct for whatever you like.

The project homepages are:

- `http://www.HardcoreProcessing.com/pro/abstractuiml/index.html`
- `http://www.HardcoreProcessing.com/pro/theconstruct/index.html`

This document is available as an online HTML version at:

- `http://www.HardcoreProcessing.com/pro/abstractuiml/presentations/basics.html`

From here there are links to other versions of the document:

- `LaTeX version of AbstractUI::ML and The Construct Basics`
- `PostScript version of AbstractUI::ML and The Construct Basics`

A slight change of name

On the project homepage, the name is still ML::AbstractUI. This is in fact an ML type error, since `ML` is what we already have and AbstractUI is the new element that we're cons'ing onto it. So, the new and correct name is AbstractUI::ML.

Overview

The main components of AbstractUI::ML are the following:

- A directed graph for builing and controlling the user interface.
- This graph consists of typeneutral nodes each of which can be typecasted into a protocol of a given type.
- The types of protocols used in the user interface graph.
- The user interface controls (UI Controls) which are implemented and which are used to form the user interface graph.

User interfaces are built and controlled in the following way:

- A master graph node for the user interface is created.
- All needed UI Controls are created. This is typically one or more windows and all UI Controls for each window.
- These UI Controls are assembled into a user interface graph since they all consist of graph nodes. The master node must be the root of the graph.
- The UI Controls may be modified by typecasting some of their graph nodes into protocols and performing operations by using these protocols.
- A function for creating the actual user interface is called which makes the user interface show up on the screen.
- The UI Controls may still be modified at runtime by using their protocols.

Software component dependencies

In AbstractUI::ML great care has been taken to avoid that applications depend directly on the types and the implementations of UI controls, such as buttons and listboxes. The important dependencies are illustrated in the following UML diagram.

Figure 1: UML diagram of the type dependencies between software components in AbstractUI::ML

Had the applications developed been dependent upon the types or the implementations of UI controls, then any changes or enhancements made on UI controls could potentially break all applications using those UI controls.

Figure 2: UML diagram of how the type dependencies might be between software components in other GUI toolkits.

Protocols

Protocols work the following way:

- The protocols are really records of references to functions.
- Each protocol has a structure with the type of the protocol. This type is called `t` in the structure. So the type of the `PString` protocol is `PString.t`.
- The structure for a protocol also has functions used to execute functions of the protocol. As an example one might write `PString.write someProtocol "New stringvalue!"`.
- It should be noted that in the next release of AbstractUI::ML the protocols just will be records of functions, so that one can conveniently type `#write someProtocol "New stringvalue!"` instead of the above.

Some of the protocols which are implemented:

- PString, PInt, PBool etc. These protocols allow modification of and access to a single dataelement of some type. These protocols also make it possible to add eventhandlers for when the value of the dataelement changes.
- PStrings, PInts, PBools etc. These protocols allow modification of and access to arrays of dataelements of some type. It is also possible to add eventhandlers for when such an array changes.
- PIntDrawable is a protocol for drawing on a UI Control using integer coordinates.

Typecasting a typeneutral node into a protocol can be done in 2 ways:

- The quick way is to use the `DGP*` structures. For instance to typecast a node to a PString protocol one would write something like `val protocol = DGPString.p node`.
- An equivalent way is to write `val protocol = DGNodes.protocol PString.P.c node`.

About the type-neutral-hack

Some of the reasons why the graph nodes are typeneutral and has to be typecasted into protocols are:

- It would be virtually impossible to parametrize an entire user interface graph with ML's types.
- The type of each user interface that a user might implement would be different which would not allow the same uniform treatment of the graph as is the case.

Typeneutral nodes are implemented (very roughly) in the following way:

- Let the type of some protocol be `t`.
- A global reference `R` of type `t option ref` is set up for this type of protocol.
- A typeneutral instance of the protocol is created from a protocol `P` (having type `t` of course) as a function `s` of type `unit -> unit`. This function just assigns `SOME(P)` to the global reference `R` for the protocol type.
- A typecasting function `c` is implemented for the protocol type. `c` takes a function `f` of type `unit -> unit` as argument, which is the typeneutral instance, such as the function `s` from before. `c` is implemented so that it first assigns `NONE` to the reference `R`. Then it calls the function `f` and checks if the reference `R` now has a value of `SOME(protocol)`. If it has, then `protocol` is returned from the typecasting function `c`. Otherwise an exception is raised because that the type of the given protocol was aparently incorrect.

About the implemented UI Controls

The components in AbstractUI::ML consists of typeneutral graph nodes. The subgraph of an implemented UI Control typically looks something like the following:

Figure 3: The subgraph of a typical UI Control

- The node named `Window` is the main node of the UI Control.
- The other nodes connected to it are the parameter nodes for the UI Control. These are the nodes that are usually typecasted into the appropriate protocols so that the UI Control can be modified through these.
- The `Children` node can be connected to other UI Controls in this Window Control. The Window Control should itself be connected to a graph from a similar `Children` node of the master node for the user interface. This is how the entire user interface graph is connected.
- Notice that the connections of the graph are directed. The directions of the connections are important.

The Construct - a GUI builder for AbstractUI::ML

Given the introduction above to AbstractUI::ML, then the GUI builder for it should be really simple to use and understand. The easiest way to learn about it is to install it and try it. When starting The Construct you will see the following windows:

UI Palette

In this window there's a button for each UI Control available in AbstractUI::ML (or rather - which were available when The Construct was compiled).

- Push the button labeled Window to create a new window.
- UI Controls can be added to this window by first pressing the appropriate button followed by a click on the desired window that you have created earlier.

Protocol Accessor

This is where you can modify the UI Controls that you create.

- First click on the desired UI Control that you wish to edit. This will display a list of all the protocol nodes for the selected UI Control.
- You can modify each protocol by typing in the textfields to the right, except when they contain the text `N/A:` or `(only for bindings)`.
- It is also in the Protocol Accessor window that you can make desired variable bindings from the created UI Controls to your application. For instance to make a binding for the TextLines protocol of a ListBox you select the listbox to activate the Protocol Accessor and click the toggle button at the left of the window on the line with the TextLines label. Then enter the desired variable name.

Operator

This window lets you load and save the user interfaces that you have built with The Construct.

- You must first type the filename to load or save and then press either the load or the save button.
- The user interfaces are saved as `SML'97` code which is ready to be compiled into an application. Of course the application will have to be compiled with AbstractUI::ML in order to work.

The generated code may not be very suitable for editing and may not be very readable for humans either. Also The Construct will certainly not be able to read any ML file you throw at it. It is intended only to be able to read the files that it has generated itself. However you should not need to edit the code, since the code generated by The Construct is intended to be completely separate UI code which is independent of application code.