Type-Oriented Construction of Web User Interfaces

– Extended Abstract –

Michael Hanus
Institut für Informatik, CAU Kiel, D-24098 Kiel
mh@informatik.uni-kiel.de

Abstract

We propose a new technique for the high-level construction of type-safe web-oriented user interfaces. Our approach is useful to equip applications processing structured data with interfaces to manipulate these data in an efficient and maintainable way. The interfaces are web-based, i.e., the data can be manipulated with standard web browsers without any specific requirements on the client side. In order to support type-safe user interfaces, i.e., interfaces where users can only input type-correct data (types can be standard types of a programming language as well as any computable predicate on the data), we propose a set of type-oriented building blocks from which interfaces for more complex types can be easily constructed. This technique leads to a very concise and maintainable implementation of web-based user interfaces.

1 Introduction

The construction of user interfaces for applications manipulating structured data is usually a complex and often tedious task. In many cases the effort to implement a user interface is equal or even bigger than the implementation of the application itself. Thus, there is a demand to support the efficient construction of maintainable user interfaces. In this paper we propose a new technique for the case of web user interfaces (WUIs) where the client uses a standard web browser for communicating with the application.

Our approach is useful in situations where a web-based editor should be constructed for data of an application program, i.e., the user should be provided with an HTML form to manipulate some data of the application (see Fig. 1). For this purpose we assume that the application program supplies the WUI with the current data of the application and an operation to store the modified data. It is obvious that this is not a restriction since application programs usually have such a functionality. Using our concept, nothing more is required to construct WUIs in a high-level way by a few lines of program code. Our programming model can be characterized by the following features:

- The construction of a WUI is *type-oriented*, i.e., the definition of a WUI follows the structure of the data types of the application.
- There is a set of *basic WUIs* to manipulate data of basic types, e.g., integers, truth values, strings, finite sets. This set can be easily extended since there is a clear methodology to implement such basic WUIs.
- There is a set of *WUI combinators* to construct WUIs for complex data types from simpler types similarly to type constructors in programming languages. For instance, there are combinators for tuples, lists, union types etc.
- It is ensured that an update of the data is only performed with *type-correct inputs*. If the user tries to input illegal data (e.g., incorrect integer constants), the WUI does not accept the data and ask the user to correct the input. Thus, the application program need not check the data and perform appropriate actions (e.g., providing error forms to correct the input etc).
- Type-correct inputs (in the sense of types used in programming languages) are often not sufficient in real applications. For instance, strings containing email addresses must have a particular form, a date like “February 29, 2006” is illegal, or two input fields containing a password and the repeated password must be always identical. For this purpose, WUIs can be *restricted with any computable predicate* so that input data is only accepted if it satisfies the specified predicate. Furthermore, WUIs can be customized to provide *application-specific error messages* in case of illegal inputs.
- WUIs can be adapted to other data types in order to provide a simple method to define *WUIs for user-defined data types*. For instance, there exist WUI combinators for tuples that can be easily adapted to a user-defined record type by mapping tuples to records. Although this method is often sufficient to construct WUIs for user-defined types, there is also a methodology to extend the standard set of WUI combinators with new application specific combinators.

In principle, our ideas can be implemented in various programming languages. However, in order to support a compact, high-level, and type-safe implemen-

Figure 1: A WUI for a list of persons

tation, some requirements to the underlying programming language are necessary. Therefore, we provide a concrete implementation of our concept in the declarative multi-paradigm language Curry [1, 3]. The integration of functions as first-order objects, logic variables, and strong typing, as available in Curry, is exploited in our implementation.

2 Constructing Web User Interfaces

Our only requirement to the application program is that it supplies the WUI with the current state of the data and an operation to store the data modified by the user. Thus, the main operation to construct a WUI has the type signature

```
mainWUI :: WSpec a -> a ->
              (a -> IO HtmlForm) -> IO HtmlForm
```

so that an expression (`mainWUI` *wspec d store*) evaluates to a web page containing an editor that shows the current data d and executes ($store\ d'$) when the user submits the modified data d'. The operation *store* (also sometimes called *update form*) usually stores the modified data in a file or database, returns a web page that informs the user about the successful (or failed) modification, and proceeds with a further interaction.

The parameter *wspec*, also called *WUI specification*, specifies the kind of WUI elements to be used in the interface. Our implementation provides a number of predefined WUI elements for particular types. For instance, to edit simple strings, there is a predefined entity

```
wString :: WSpec String
```

defining a WUI element that shows the string in a simple text input field.

In order to edit integer values, there is an entity

```
wInt:: WSpec Int
```

defining a WUI element that shows an integer in a text input field. Note that WUI elements are type safe, i.e., if the user inputs a non-integer in such an input field, the implementation of the WUI emits an error message and asks the user to correct the input.

To select elements from a finite set values through a selection box, there is a WUI element

```
wSelect :: (a->String) -> [a] -> WSpec a
```

The first argument is a function to show an element as a string (shown in the selection box) and the second argument contains the list of elements to be selected.

Similarly to the use of type constructors for the construction of complex data types from simpler types, WUIs for complex types can be constructed from WUIs for simpler types by *WUI combinators*. A WUI combinator is a mapping from simpler WUIs to WUIs for structured types. For instance, there is a family of WUI combinators for tuple types:

```
wPair   :: WSpec a -> WSpec b -> WSpec (a,b)
wTriple :: WSpec a -> WSpec b -> WSpec c
           -> WSpec (a,b,c)
...
```

Thus, the expression

```
wDate = wTriple (wSelect show [1..31])
                (wSelect show [1..12])
                wInt
```

evaluates to an HTML form to edit a triple of integers representing dates.

In order to manipulate lists of data objects, there is a WUI combinator for list types:

```
wList :: WSpec a -> WSpec [a]
```

Thus, `(wList (wTriple wString wString wDate))` specifies the WUI shown in Fig. 1. From this simple specification, the handling of input values, error correction forms etc. is automatically generated.

In order to adapt standard WUI specifications to application-specific requirements, our library provides operations to customize WUIs, e.g., restrict input fields with specific conditions and appropriate error messages, define new renderings for WUI elements, adapt WUIs from one data type to another etc. More details can be found in the full paper [2].

References

[1] M. Hanus. A Unified Computation Model for Functional and Logic Programming. In *Proc. of the 24th ACM Symposium on Principles of Programming Languages (Paris)*, pp. 80–93, 1997.

[2] M. Hanus. Type-Oriented Construction of Web User Interfaces. In *Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'06)*. ACM Press (to appear), 2006.

[3] M. Hanus (ed.). Curry: An Integrated Functional Logic Language (Vers. 0.8.2). Available at `www.informatik.uni-kiel.de/~curry`, 2006.