FormGen: A Generator for Adaptive Forms Based on EasyGUI

Alfons Brandl, Gerwin Klein
Institut für Informatik, Technische Universität München

As a common task in modern applications users need to enter, edit or browse large, complex, maybe even recursive data structures. Many users prefer form based user interfaces. The form fillin interface style (Szekely 1998) is in widespread use e.g. for database queries, e-commerce orders etc. and exploits the user's familiarity with the paper equivalent. Many use cases are cumbersome on actual paper forms (e.g. “fill in lines 7-9 only if you answered yes to question 6, proceed to 10 if you answered no”) but can be handled quite elegantly with electronic forms by displaying relevant sections only - depending on the users former input. Those forms are called adaptive (Frank and Szekely 1998) or dynamic (Girgensohn, Zimmermann et al.1995).

Coding adaptive forms by hand or implementing them with a layout based development tool can grow rather expensive for large and more complex data structures. This paper focuses on how the development of adaptive form based user interfaces can be supported by the automatic code generation tool *FormGen*.

Apparently there are three points to consider when developing a form based UI:

- the application domain, i.e. the data structure to be edited,
- the dynamic behavior and
- the layout of each form to be displayed.

In the case of form fillins, layout and dynamics of the user interface can be quite easily deduced from the logical structure of the application domain. Given the form paradigm, the data type determines to a very large degree what to present to the user at a given time. It also determines in which order data needs to be entered and which parts can be entered independently of one another.

Using these relationships FormGen generates a complete set of forms from a formal description of the input data type and in this way enables the development of adaptive forms without any coding in a programming language.

In FormGen, like in many programming languages, data types can be described using tuples (records), variants, lists and basic predefined types. FormGen offers the developer a context free grammar notation for defining abstract data types. The following example illustrates this notation by defining an abstract data structure for search queries in an internet search engine:

Example:

```
Query          ::= CompositeQuery | Text | DateRange
CompositeQuery ::= Query* Operation
Operation      ::= and | or
Text           ::= String:searchText
DateRange      ::= Date:from Date:to
Date           ::= int:day int:month int:year
```

In this example a query is either a composite query containing a list of sub-queries connected with the "`and`" or the "`or`" operation, a piece of text, or a range of dates. Note that this grammar is ambiguous as well as recursive - both cases are handled by FormGen.

As shown in figure 1, FormGen actually consists of two generators: *classgen* and *formgen*. classgen generates a set of Java classes representing the data type described by the specification, formgen generates Java classes that implement the actual form based user interface. The modular architecture of the generator enables the use of classgen not only for the generation of user interfaces, but also in other contexts such as generating attributed abstract syntax tree representations in programming language parsers, intermediate representations in compilers or any other recursive data type expressed by a context free grammar.

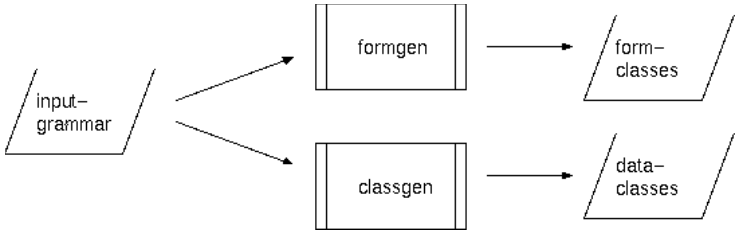

Figure 1: FormGen architecture

Our tool for generating form components is surprisingly simple, accomplishing its task in one pass. This is possible, because the generated classes are based on the most recent version of the Java UI framework *EasyGUI* (Klein 1997). The EasyGUI framework differs from other toolkits such as Java AWT in declaring the layout of an user interface per state rather than per state transition. This feature of EasyGUI allows the presentation aspects to be distinct from the dynamic aspects of the user interface, simplifying the form generator's work enormously.

The underlying concepts of EasyGUI stem from the field of compiler construction and were originally applied in the scope of user interfaces in the *BOSS* system (Schreiber 1994, Schreiber 1997). EasyGUI has been applied in different graduate and undergraduate student courses at the Technische Universität München since 1995. These courses were arranged by the chair for formal languages, compiler and software construction of professor Eickel.

Internally the user edits and navigates an abstract, attributed data tree e.g. adding new nodes or changing the content of existing nodes. FormGen regards the presentation part of the user interface as an attribute of the tree. The content of this attribute is a relatively abstract set (in comparison to e.g. Java AWT) of interaction objects: covering basics like buttons, and composite interaction objects like boxes which order their child objects horizontally or vertically in a TEX-like manner. EasyGUI provides direct support for this kind of layout.

The value of each layout attribute instance is calculated by an arbitrarily complex function. For example this function could display the whole data set entered by the user during the whole session, marking the current editing position in some way, or it could just display the current position in the tree together with the currently available actions the user may want to perform (such as changing the content of the node or adding a new sub node).

FormGen provides a standard layout calculating function that handles the presentation of tuples, variants, lists and built-in basic types. This is sufficient for most cases, but in real world applications domain specific customizations will be necessary.

Since FormGen provides the layout calculating function in a very flexible and pluggable way, customizations can be dealt with quite easily by writing customized layout functions. It is possible for the designer to customize the layout e.g. for all tuple types, for any special type (e.g. a special layout for all “and”-queries) or even for a concrete instance of a tree node (e.g. the root query could be given another layout than the sub-queries).

The screenshots of some generated forms in figure 2 show the correlation between the layout of the form and the corresponding abstract tree.

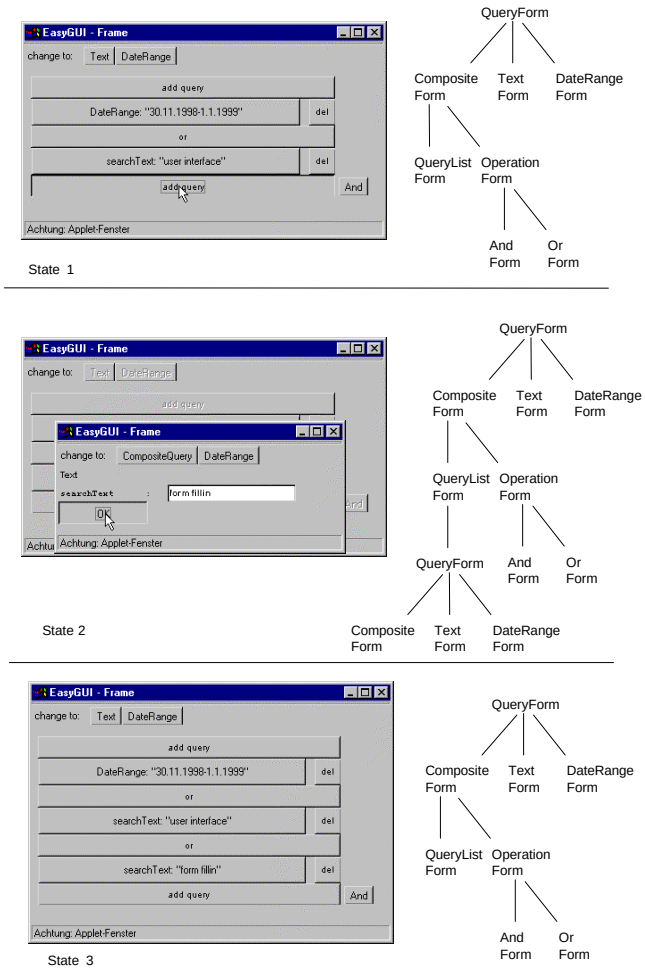

Figure 2: Screenshots of generated forms

Even if the application domain and therefore the input grammar has changed, the customized layout functions can be reused because of the applied object oriented generation concept.

In the current version the developer implements the customized layout functions in Java. Our future work includes a wysiwyg interface builder which will allow to customize the form layout without having to write any code in a programming language.

Why does this interface builder approach differ from conventional layout based tools?

Generally the development of appropriate forms is split up into the design of the layout of the form and the programming of logical aspects e.g. by implementing the data type. With conventional tools user interface designers are able to edit the layout of the form visually whereas the logical part is done by programming. Unfortunately both tasks are not coupled with each other and the tool cannot assist the designer in important aspects e.g. in deciding whether all data is visually presented.

On the other hand our model based visual interface builder will be equipped with all relevant information about the logical structure of the application domain and would already provide the developer with a good suggestion for the layout. Most importantly the layout only would have to be adjusted and wouldn't need to be created from scratch. This should reduce development time drastically and can eliminate a whole set of common errors that cannot be checked by conventional layout based interface builders.

References

M. Frank and P. Szekely (1998). Adaptive Forms: An interaction paradigm for entering structured data. In: *Proceedings of the International Conference on Intelligent User Interfaces* (San Francisco, USA, January 6-8), 153-160.

A. Girgensohn, B. Zimmermann, A. Lee, B. Burns, and M. E. Atwood (1995). Dynamic forms: An enhanced interaction abstraction based on forms. In: *Proceedings of Interact'95, Fifth IFIP Conference on Human-Computer Interaction* (London, England), 362-367.

G. Klein (1997). *Die objekt-orientierte Bedienoberflächenbibliothek EasyGUI*. Fortgeschrittenenpraktikum, Technische Universität München.

S. Schreiber (1994). Specification and Generation of User Interfaces with the BOSS-System. In *Human computer Interaction, Selected Papers EWHCI'94 Conference*, Springer LNCS 876.

S.Schreiber (1997). *Spezifikationstechniken und Generierungstechniken für graphische Benutzungsoberflächen*. Dissertation, Technische Universität München.