511-61
136885
p-4

N93-17510

Towards Automation of User Interface Design

Rainer Gastner Gerhard K. Kraetzschmar Ernst Lutz

Research Group Knowledge Acquisition
Bavarian Research Center for Knowledge Based Systems (FORWISS)
Am Weichselgarten 7, 8500 Erlangen, Germany
e-mail: gastner@forwiss.uni-erlangen.de

Abstract

This paper suggests an approach to automatic software design in the domain of graphical user interfaces. There are still some drawbacks in existing UIMSs which basicly offer only quantitative layout specifications via direct manipulation. Our approach suggests a convenient way to get a default graphical user interface which may be customized and redesigned easily in further prototyping cycles.

1 Introduction

The automation of software design becomes more powerful if the target systems generated are limited to a certain domain. The domain addressed in this paper is the class of graphical, highly interactive systems for accessing data of specified data structures by end users. The focus of this paper is further restricted. It concentrates on the automation of the design of a graphical user interface (GUI) for these systems.

Building GUIs with GUI toolkits or user interface management systems (UIMSs) is still a laborious, time-consuming task even if it is supported by direct manipulation facilities [6]. The basic problems we identified are the following:

- The GUI designer has to decide which graphical element is appropriate for a desired interaction, i.e. given a data structure and data type descriptions of the elements to be accessed and a set of GUI elements the designer has to perform a mapping between the data structure and the GUI elements.
- With direct manipulation an initial GUI may be built but if the data structure or the data types are changed the manual adaption of the GUI is arduous. According to the changes of a data structure the extent of the redesign task may cause pretty much effort.
- Due to the lack of adopted GUI design guidelines, for similar data structures in different applications a different GUI may exist which is contradictious to user interface consistency [10].

The approach introduced in this paper to address these problems is the automatic generation of GUIs from a high level specification. This generation is performed by a knowledge-based meta-tool which is used by a GUI designer. Questions which have to be tackled include the following: (1) To what extent can the designer be supported in the specification task? (2) What kind of user interface should the meta-tool have. (3) Which kind of knowledge is domain invariant and which is application specific (and therefore needs to be entered by the designer)? (4) Which set of default design decisions are adequate?

Our approach to answer these questions is based on the following idea: The designer specifies data structures, data types and operations which the user of the target system has to perform with an user-friendly GUI. Corresponding GUI elements realizing these operations are associated automatically and the GUI is generated. The designer in turn refines the GUI by interactivly customizing the meta-tools association and specifying *qualitative* layout constraints. This approach facilitates users who have no knowledge about interface programming to construct a GUI easily. Since the GUI of a meta-tool itself is in the domain our approch is applicable for the design of meta-tool's GUI as well.

In section 2 the addressed domain is introduced in more detail. Section 3 discusses the problems of configuration and generation of the target systems. In section 4 our approach is described to solve these problems. Section 5 compares our approach to related work and section 6 gives some concluding remarks and perspectives on future work.

2 Domain

The domain our meta-tool addresses is the class of GUIs that allow the access of specified data structures whose elements are characterized be specific data types. The access comprises additon, deletion, modification, selection and browsing of data structures and instances.

There exist rather different interpretations of what the notion GUI should mean [6]. In our meta-tool the GUI is built with a set of objects which have a description of a graphical presentation and methods to handle the display presentation and the communication with the underlying window system. Examples are buttons, settings or text fields. No other functionality is added to the GUI. The GUI objects are described within an object-oriented class hierarchy adopting inheritance. This is the common approach how state-of-the-art GUI toolkits and UIMSs are realized [6].

Our meta-tool produces specializations of classes in a class hierarchy provided by the GUI toolkit *LispView* [1] and instantiation methods. LispView provides an interfaces between Sun CommonLisp and OpenWindows. The same structure is generated by the GUI devolopment system *OpenWindows Developer's Guide* [3].

3 Problem Description

The design of a meta-tool for automating the design of GUIs from specifications of data structures, data types and operations raises some questions which mainly influence the meta-tool design decisions:

- Which kind of knowledge has to be represented to support the generation and which kind of knowledge representation should be used?
- Which part of the knowledge is domain specific but application invariant and which part is application specific?
- What kind of default configuration decisions makes sense? Can specific subdomains be identified for which specific configuration macros may be used?
- What is the most efficient way to enter geometric layout specifications?
- Since an initially generated GUI in most cases does not meet the end user's whishes rapid prototyping facilities for iterative refinement and customization is needed.
- The specification facility must allow only consistent specifications, i.e. the designer's specification has to be syntactically and semantically correct and the generator will produce a GUI inside the domain. How can we support specification consistency?

The following section discusses our approach towards an automation of the GUI design addressing the questions given above.

4 Approach

State-of-the-art UIMSs mainly deal with a user-friendly composition of the GUI. From this point of view only the syntactical aspects in building GUIs are addressed. But naturally GUIs are built for user interactions which have certain semantics. For instance, when the GUI designer using a direct manipulation UIMS selects a button and arranges it in the target interface via mouse dragging he knows the reason why he selects a button and which operation should be performed by clicking on the button. The GUI components are nothing else than graphical presentations of abstract interactions. The mapping from the semantics of these interactions to corresponding GUI elements is the main task of an GUI designer.

Our approach for specifying GUIs starts from a semantic point of view and focuses on this mapping. The GUI designer does not specify a composition of the GUI components itself rather than the interactions the GUI components shall be used for. That means the focus of the specification is not *how* to present interactions on the screen but *what* kind of interactions shall be established. The interactions we consider are the access operations specified in section 2. The mapping from the interaction specification to the GUI components is done by the meta-tool automatically. In a further step the designer may customize the generated GUI either by changing the mapping or specifying additional qualitative layout constraints.

4.1 Configuration Process

In this section the configuration process is discussed. Figure 1 provides an overview of the configuration steps. The actions the designer has to perform are

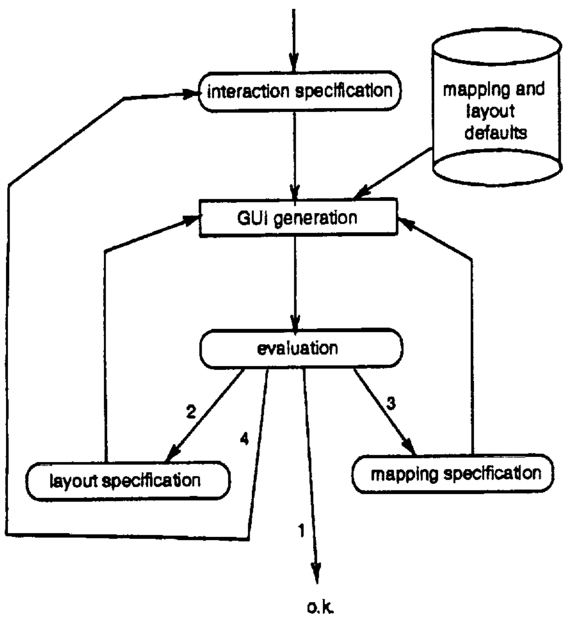

Figure1: iterative GUI configuration process

specification and evaluation represented as round-cornered boxes. The meta-tool activity (the generation of the GUI) is represented as a rectangular box.

The designer starts with the specification of the desired interactions on data structures. Then an initial GUI is generated by the meta-tool using default mapping and layout configurations stored in a knowledge base (see section 4.2). The initial generation has to be evaluated by the designer. Then one of the following four choices may be made:

1. The designer agrees with the generated GUI and the configuration process is finished.
2. The designer specifies qualitative geometric layout constraints to rearrange the GUI components on the screen.
3. The designer alters the mapping between the specified interactions and the corresponding GUI component.
4. The designer manipulates the interaction specification, e.g. a new element is added to a data structure.

In case of a new or re-specification a new generation cycle starts. The order given above implies the extent of the GUI redesign in a cycle after evaluation. Choice 2 affects only the geometric position of GUI elements, choice 3 affects the presentation of an interaction, and choice 4 affects the interaction itself. Explorative rapid prototyping by iterating the configuration cycles is supported conveniently, since the designer starts with a specification of abstract interactions omitting GUI aspects in the initial phase. In following cycles he can customize presentation aspects very quickly or redesign the interactions.

Since end users are supposed to design the GUIs the meta-tool must provide user-friendly graphical interfaces itself. To support specification consistency, the specification is menu-driven as far as possible. Menus with appropriate selections may be offered which is further discussed in section 4.2. An interesting issue is that the GUI of the meta-tool to enter the specification is itself in the domain of the meta-tool. Since the meta-tool allows to use the specification languages directly without the corresponding GUI[1], the GUI for the meta-tool can be generated by the meta-tool itself.

4.2 Configuration Knowledge and Representation

This section deals with the knowledge needed to automate the GUI configuration. We distinguish two classes of knowledge. Knowledge is needed to support an efficient user-friendly specification and to generate a GUI with an minimal specification. This kind of knowledge is application-invariant and refered as *domain-specific* (in the GUI domain). On the other hand *application-specific* knowledge must be enterd by the designer to build an GUI for a set of certain interactions. The following two subsection discuss these two knowledge classes.

4.2.1 Domain-specific knowledge

The following listed knowledge categories are stored in the meta-tool's knowledge base in order to support specification and generation. Note that this is mainly knowledge *about* the possible application-specific knowledge (e.g. possiple types of layout constraints) and therefore meta-knowledge.

- **model of target architecture;** the structure of the code generated by our meta-tool is given by the code structure the Developer's Guide for LispView interfaces [3] generates.
- **a library of interaction types and data types;** interaction types include read and write access to data and selection of data. Currently the library of data types includes enumeration, character, real, integer, string, symbol, and object-class.

[1] Otherwise there would be meta-tool tower never ending.

- **a library of GUI elements**; this library is given by the used GUI toolkit LispView [1].
- **mapping of interaction specifications to GUI elements**; the mapping is stored as a matrix in which for certain conditions made in a data type specification a set of possible GUI components is associated. The GUI component selected by default is marked (see also section 4.4.
- **library of layout constraints**; currently we have realized 36 layout constraint types which are hierarchically organized and offered in menus. Furthermore, there exits a layout constraint construction facility for the meta-tool designer to implement additional constraint types based on a combination of types from a basic set.
- **standard configurations**; see section 4.4.

The domain-specific knowledge is stored in ASCII-files in special representation languages. The files may either be edited directly by a text editor or be generated from graphical specifications. An interpreter reads these files and maps the external representation to internal objects.

4.2.2 Application-specific knowledge

As shown in figure 1 there are three specification possibilities providing input for the generator.

- **interactions**; the specification comprises the type of operation and the data type to be accessed. The data type is specified separately. Thus more than one interaction may access data of the same data type in different ways. The example below shows the declarative specification generated from the graphical specification environment. A manipulation interaction is specified on data of an integer slot. The value range is restricted between 100 and 500, the slot is single-valued, and the value must be unique and entered.

```
(def-interaction
      :id 'engine-number-manipulation
      :operation 'manipulation
      :data-type 'engine-number-type)
(def-integer
      :id 'engine-number-type
      :equalorgreater 100
      :lessorequal 500
      :mincard 1
      :maxcard 1
      :unique T)
```

The declarative specification languages may also be used directly by the designer. Both interactions and data types are offered in menus to the designer. The menus are configured dynamically according to certain specification constraints; e.g. the following constraint may not be violated in the example above: `(lessorequal mincard maxcard)`.

- **association of interactions and GUI components**; if the designer does not agree with the meta-tool's association he may select another association or more than one associations for a given interaction from a menu. The menu items consist of all GUI components which are acceptable presentations for the interaction asserting a consistent specification. If the designer associates more than one GUI component to an interaction, the interaction is presented in different fashions in the GUI. For instance, the interaction in the example above may be presented as numeric field or a slider. The meta-tool would select the numeric field by default.
- **layout constraints**; the qualitative layout constraints may be specified using a declarative specification language or a GUI generating sentences of this language. The following example demonstrates the power and user-friendlyness of our layout mechanism:

Let B_1, B_2, …, B_5 be boxes which shall be arranged as follows: B_4 and B_5 shall be at the bottom of the layout frame; B_1 shall be in the upper left corner of the layout frame; and B_3 shall be over B_2 and B_5. This is expressed as follows:

(bottom-margin B_4 B_5)
(upper-left-corner B_1)
(over B_3 (B_2 B_5))

Entering the first layout constraint via the specification GUI B_4 and then B_5 would by selected with the mouse on the screen and then the constraint `bottom-margin` would be selected from the menu.

4.3 Layout computation

Each GUI element has a rectangular bounding-box which provides the size for the layout generator. The 36 layout constraints are one-dimensional geometric relationships between these boxes. N-ary relationships are resolved into binary ones which are connected with a conjunction. The corners of the boxes are represented by variables and the constraints are always inequations of the following form:

$$\alpha_i \leq x_j - x_i \leq \beta_i$$

These unequations can be solved using a longest-path algorithm suggested in [11]. If there are inconsistencies in the specified constraint set our algorithm retracts contradictious constraints. The boxes are arranged fulfilling the specified constraints and are positioned in the upper left corner of the layout frame. In a second cycle overlapping boxes (this may occur if the constraint set is not restrictive enough) are solve by adding additional contraints with disjunctions: A box B_1 and a box B_2 do not overlap if (beside $B_1 B_2$) $\vee$ (beside $B_2 B_1$) $\vee$ (over $B_1 B_2$) $\vee$ (over $B_2 B_1$) holds. Since there may be a huge number of layout configurations solving the constraint set without overlapping the layout algorithm gets a certain time for processing (e.g. three seconds). The algorithm generates a set of solutions and then selects the best solution when the time is over. The selection criteria adopted currently is either to minimize the area of the layout frame if the size is not prespecified or to arrange the boxes with equal distances between them in a fixed layout frame.

4.4 Standard Configurations

In order to give support in the specification of GUI component associations to interactions and to select default associations we (partly) represent knowledge found in the *OPEN LOOK application style guidelines* [2]. This knowledge is stored in a matrix in this way that for each GUI component it is marked under which conditions it is appropriate and if it should be selected by default. Furthermore, OPEN LOOK provides a unique look-and-feel for all the target GUIs and the GUI sepecification environment of our meta-tool.

It is possible to preconfigure special editor types which include a number of fixed interactions. For instance, a login editor consists always of two interactions, one for entering the user's name and one for entering the password. These two interactions are preconfigured as a symbol and string manipulation interaction. Furthermore, a layout frame with a fixed size is configured, layout constraints are specified that both GUI components (the meta-tool will associate two text fields) should be centered and the text field for the user's name should be located over the field for the password entry. The configuration is stored as subclass of a preconfigured editor class. Other specialized editors may be partly preconfigured and layouted like object editors or browsers. Preconfigured GUI classes can be dynamically added by the designer.

Adopting this configurartion library and the represented OPEN LOOK style guidelines we facilitate the generation of GUIs which have a common structure and supports GUI consistency [10].

4.5 Implementation

Our meta-tool is implemented in Sun CommonLisp, CLOS and LispView [1]. Object-oriented programming is adopted basically. The target code is generated using templates which are expanded according to the designer's specification or standard configurations. By replacing the templates it is possible to generate other GUI target code as well.

5 Related Work

In the last decade human-computer interaction and the user interfaces have become an important research field. UIMSs try to improve GUI development and support mechanisms for GUI and dialogue specification, representation and management [6] [9]. In [7] several generations of UIMSs are identified. It is predicted that future UIMSs will be knowledge-based and generate a user interface automatically using the specification of the underlying application. Our approach is a step in this direction. Currently the interactions have still to be coded by a GUI designer, but there should be a way to generate the interaction specification from application programs automatically as well.

A number of development methodologies have been suggested for user interfaces. Most of them claim explorative prototyping as our approach (see figure 1), e.g. the star life cycle suggested in [7].

User interfaces may be specified language-based with special user interface description languages, graphical-based with direct manipulation facilities or with automatic generation from interaction descriptions [9]. Since our meta-tool generates code which can be manipulated by the Developer's Guide [3] our approach combines these three possibilities which may be alternatively used.

Similar approches for automatic generation of GUIs are used in the GADGETS system [8] and the PRED system [13], but they lack qualitative layout specifications. Automatic presentation systems for information like SAGE [12] also use meta-information to select an adequate presentation style. A similar approach of default configurations of editors is applied in the meta-tool DOTS [4].

6 Concluding Remarks and Future Work

We suggested an approach towards automation of user interface design which starts from a semantic point of view. The initial specification only deals with *what* the GUI is to be built for and not *how*. Further prototyping cycles allow to customize the generated GUI *qualitatively*. Since the generated GUI code is interpretable by the direct manipulation tool Developer's Guide [3], also quantitative layouting is available and may be adopted alternativly. Since the meta-tool's GUI is in the meta-tool's domain itself a reflexive application of the meta-tool is possible.

In the project KME (Knowledge Maintenance Environment)[2] we designed a meta-tool called KME workbench [5] for generating maintenance components for knowledge bases of expert systems. A maintenance component for updating objects of an object oriented representation needs a GUI of the domain described in this paper. Thus the GUI design meta-tool is part of the KME workbench. We experienced in this project that qualitative layout specifications are very convenient and allow rapid explorative prototyping. The GUI specification environment also allows end users (e.g. knowledge engineers with only few programming experience) to build adequate GUIs easily.

We acquired GUI design knowledge from the *OPEN LOOK GUI application style guidelines* [2] which is represented in a matrix representation and allows the meta-tool to provide default configurations. Furthermore, the explicit representation can easily be changed and augmented.

Currently we work on the extension of default configurations and GUI facilities. Special editor types are identified in more specific application domains and represented. We will evaluate how the GUI specification can be acquired automatically from the underlying application. In the knowledge maintenance context we will try to generated a default dialogue control supported by a transaction management.

[2]KME was started as joint project between FORWISS and the company BMW, Munich.

References

[1] *Lisp View Programming Manual.* Sun Microsystems, Inc., 1989.

[2] *OPEN LOOK Graphical User Interface Application Style Guidelines.* Sun Microsystems, Inc., Addison-Wesley, Reading, Massachusetts, 1990.

[3] *OpenWindows Developer's Guide 1.1, User's Manual.* Sun Microsystems, Inc., 1990.

[4] Henrik Eriksson. *Meta-Tool Support for Knowledge Acquisition.* PhD thesis, Linkoeping University, Sweden, 1991.

[5] Rainer Gastner, Gerhard K. Kraetzschmar, and Ernst Lutz. Kme-workbench: a meta-tool for designing maintenance components for knowledge based systems. paper submitted to ECAI92, January 1992.

[6] H. Rex Hartson and Deborah Hix. Human-computer interface development: concepts and systems for its management. *ACM Computing Surveys*, 21(1):5-92, March 1989.

[7] H.R. Hartson and D. Hix. Toward empirically derived methodologies and tools for human-computer interface development. *Int. Journal of Man-Machine Studies*, 31(4):477-494, October 1989.

[8] Johannes L. Marais. The gadgets user interface management system. *Structured Programming*, 12(2):75-89, 1991.

[9] Brad A. Myers. User-interface-tools: introduction and survey. *IEEE Software*, 15-23, January 1989.

[10] Jakob Nielsen, editor. *Coordinating User Interfaces for Consistency.* Academic Press, London, 1989.

[11] Thomas Ottmann and Peter Widmayer. *Algorithmen und Datenstrukturen.* BI Wissenschaftsverlag, Mannheim, 1990.

[12] Steven F. Roth and Joe Mattis. Automating the presentation of information. In *Seventh IEEE Conference on Artificial Intelligence Applications*, pages 90-97, IEEE, IEEE Computer Society Press, Washington, February, 24-28 1991.

[13] S. Xie and P. H. Winne. Kamit: a knowledge acquisition and maintenance interface tool. In M. H. Hamza, editor, *Expert Systems Theory and Applications*, pages 115-118, IASTED - Acta Press, Anaheim, 1988.