Automatic Generation of User Interfaces using the Set Description Language

Edoardo Ardizzone
DINFO, Università degli Studi di Palermo, Viale delle Scienze, 90128 Palermo, Italy
ardizzon@unipa.it

Vincenzo Cannella
DINFO, Università degli Studi di Palermo, Viale delle Scienze, 90128 Palermo, Italy
peri@csai.unipa.it

Daniele Peri
DINFO, Università degli Studi di Palermo, Viale delle Scienze, 90128 Palermo, Italy
peri@csai.unipa.it

Roberto Pirrone
DINFO, Università degli Studi di Palermo, Viale delle Scienze, 90128 Palermo, Italy
pirrone@unipa.it

ABSTRACT

We present a paradigm to generate automatically graphical user interfaces from a formal description of the data model following the well-known model-view-control paradigm. This paradigm provide complete separation between data model and interface description, setting the programmer free from the low-level aspects of programming interfaces, letting him take care of higher level aspects. The interface along with the data model is described by means of a formal language, the Set Description Language. We also describe the infrastructure based on this paradigm we implemented to generate graphical user interfaces for generic applications. Moreover, it can adapt the user interface of a program to the needs derived from the type of data managed by the user from time to time.

Keywords
Graphical User Interface, Model-View-Control paradigm, description language

1. INTRODUCTION

Usually, the specification of a project software development and the description of a program properties are expressed through proper language, called specification language.
In this case, using a notation defined in a rigorous way in its syntactic and semantic aspect makes it possible to write the specifications precisely.
The main advantage of such a choice is the opportunity of automating the manipulation of specifications. For instance, along with the lines of what a compiler does, a specification can be analyzed syntactically and semantically to obtain a direct execution of the specifications themselves.
It is useful to note that the use of rigorous mathematical foundations does not imply necessarily the adoption of a difficult syntax. It is instead possible sometimes to create a specification language with a simple but equally expressive structure.

A kind of specification is the so-called descriptive specification.
It gives a definition of an application in a very abstract manner, through the definition of the properties that the application must have.
The method of descriptive specification starts from a definition of the state space, by giving a description of the admissible states for the modeled system in a more implicit and general way, through the use of constraints and properties expressed through an algebraic and logical formalism.
In this field, for some time the utilization of languages founded on first order logic has spread.
As proved, this kind of logic is a good basis for a formalism aimed to specify program requirements.
A program specification is given, using the logic, by means of the relationship between the input and the output data of the program.

Fundamentals on the Model-view-control Paradigm

The model-view-control (MVC) paradigm [DixHCI, Kra88] is one of the most widely used assumptions about the software architecture in the design of graphical user interfaces.
Though this paradigm it is possible to link efficiently and successfully a user-interface to the underlined data model.

According to MVC, the input of the user, the description of the links between the components of the real world, and the reply given to the user are explicitly separated and managed by three different objects, each of them specialized in one task.
The view has the task of presenting the data to the user, for instance, as a mixture of text and graphics. When the model changes, the view automatically updates itself in order to reflect the changes in the data model. The controller is instead the part of the interface that lets the user to change the model data.
It receives the input from the user and instructs the model to achieve the actions based on that input. The controller maps the user action with the application reply.
The model encapsulates the data and the functions managing them: in this way, it may capture not only a process or a system state but also its evolution.
The model thus deals with behavior and data of the application domain, replies to the request for information concerning its state, usually through the view, and modifies its state accordingly to the orders received from the controller.

2. THE "CONTEXT" ABSTRACTION

In order to achieve a description of the data model that could be used in an automatic user interface generation infrastructure we defined the "Context" abstraction [Ard02].
A context is defined as a logical structure made up by a set of controllers and views. Each of them deals with a set of variables defined by means of their respective constraints. These constraints can be also parametric. During the creation of the context itself these parameters are bound to their current values.
Every context can contain another context, thus the set of variables can be seen as a vector.
The simplicity of such a structure for the variable set guarantees a better control on the semantics of their definitions. In this way, the risk of having eventual loops in the definitions of variables is avoided.
The contexts being one inside the other form a family tree from father to son, and every son eventually inherits some features from some or all ancestors.
The state space of a context is defined through the definition set of the variables of the context and the state spaces of the contexts inside it.
The definition of every variable in a context can be subject to some or all parameters of the context.
On the contrary, each controlled variable is logically independent from each other. That is caused by the fact that listing the controlled variables does not establish a hierarchy. Their priority degree is equal to the context and so the listing sequence is indifferent.
This choice cannot solve the ambiguity caused by a programmer erroneously defining two reciprocally dependent control variables while defining the context.
In this case, a sort of control loop would be created, which makes no sense, while in a good context design a variable C2 controlling another one C1 is defined in a context that is external with respect to the one containing C1.

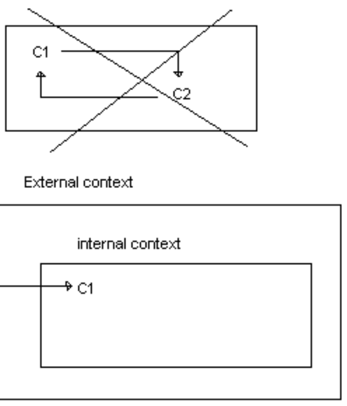

Figure 1

The vector organization of the variables of a context state space makes the recognition of such errors easier, being also a good guide to the programmer. Additionally, the occurrence check, which is, in general, an onerous computational task, is not needed.
These considerations can also be made for the view variables.
On the contrary, there is a hierarchy between the control and the view variables of the context with the latter depending on the first.
Sometimes, for instance, the same variable belongs both to the control variable list and to the view variable list.
In this case setting a value for the control variable is equal to setting the same value for the view variable. Therefore, the values inserted by the user are directly shown.
Differently, it may happen that the dependence of the view on the control variable is mediated by one or more conditions between the two variables.
Therefore, a view variable can also be dependent on more than one control variable.

Set Description Language and Contexts

The definition of a context needs a formal language ad hoc. To this purpose we designed the Set Description Language (SDL) [Ard01].
We implemented the SDL in Prolog [Plg97,Amz] also keeping the logic structure of this language.
In SDL complex contexts' structures are declared by using a vocabulary of only four keywords: context, controls, views, contains.

This feature is undoubtedly an advantage to the programmer using this language for the first time.
We now show an example of code written in a slightly modified version of SDL:

```
context ContextName(ParametersList) :=
                              controls := ControlsList,
                              views := ViewsList,
                              contains context
                              ContainedContext(Lp_con
                              t_context):

                              (Conditions_on_variables).
```

where,

- ContextName is the name of the context that is to be generated;
- ParametersList is the list of parameters of the context; such parameters are passed to the context at the moment of the its generation;
- ControlsList is the list of the control elements of the context;
- ViewsList is the list of the view elements of the context;
- ContainedContext is the name of the context container inside the context ContextName;
- Lp_cont_context is the list of the parameters of the container context ContainedContext;
- Conditions_on_variables is the list of constraints and conditions to which the variables listed in ControlsList and ViewsList are subject.

A control element is a list of two elements: the first is the variable controlled by the control, the second is a string used by the user to insert the comment. A view element has a structure similar to that of a control.
Not all the terms used in the previous code are necessary to define a context: in fact, a context can either contain only controls or only views, and it can even contain no other context. However, a context must contain almost a control or a view, and in order to define the variable managed by this unique element in the context a formal definition must be given by means of conditions to which it is subject.
In conclusion, every context has a set of parameters possibly empty. The SDL language makes thus a clear division between the interface description and the data structure: the former is expressed through a context, the latter inside the definition of the context is defined using SDL sets or finite domains.

GUI front-end

Graphical interfaces are often built by programmers resorting to object-oriented API that supplies a complete set of pre-assembled components for interfaces.
In the development of our infrastructure, we chose to use the GTK+ 2.0 API [Gtk].
Written entirely in C, GTK has been implemented with in mind the ideas of classes and callback functions.
Using GTK functions, it is possible to define wholly the graphical layout and the functionalities of a graphical interface.
The programmer must create and place the widgets, i.e. the various components of the graphical interface, among which menus, buttons, images, and even windows.

Model specification in SDL

In this work we present a simple medical image viewer as an example of the proposed system. This application gives the user the opportunity of inserting a free string (the image file name), to choose it among a limited set of strings (image labels) and/or to specify numeric values.
It is possible to draw an ideal parallel between the concept of context and a graphical interface.
In fact, a context, expressing relationships between variables, corresponds to a graphical user interface, that allows the user to interact with the elements of the context itself.
Following this idea we implemented a SDL engine that analyses the specification of a context expressed in SDL language and generates a corresponding graphical user interface.
The SDL engine uses two specific kinds of widgets, among those offered by GTK: menus and text-entries. It also makes an analysis of the type of the data dealt in the context. It chooses to use a menu to allow the user to select in a group of possible candidates, when the variable dealt by the widget is alphanumeric; through a text-entry, on the contrary, the SDL engine allows the user to insert directly the numeric value that he wants to give as input. In case the variable to be managed is numeric it chooses a text-entry widget.
The value introduced by the user are often subject to constraints and conditions limiting the choice of the user by forcing him to insert no values violating the conditions. Therefore, another function of the SDL engine is to enrich widgets with the ability of rejecting erroneous values.
In this framework menus and text-entries perform the role of controllers, according to what has been initially defined in the chosen paradigm.
The SDL engine, for this specific task, uses the GTK widget image, that corresponds to the view of our paradigm. Under every element of the graphical interface there is a label, another kind of widget of GTK. The label has not a functional role; no callback function has to be linked to it. It is used as a sort of

help which would explain to the final user the purpose of the element of the interface to which it is linked.

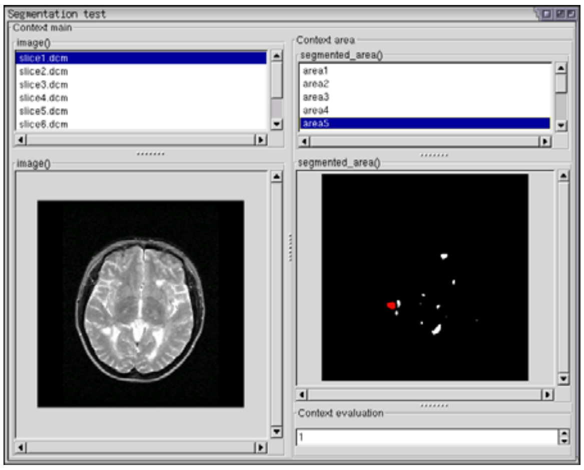

Figure 2

Structure of the layout

The arrangement of widgets inside the interface is established according to a defined criterion. Controls and views are linked between them; they are, in fact, grouped in a context. The arrangement of widgets inside the interface highlights the connection between the widgets of the same context, placing them next to each other. If a context is contained in another one then its graphical implementation is inside the one corresponding to the containing context. Every context implementation has a graphic layout similar to a newspaper page. Widgets are arranged in columns and placed one next to the other. The first widgets inserted are controls, the second are views. It is evident, in this way, the direct dependency of views on controls.
In order to decide the arrangement of the widgets inside the columns, the application associates a weight to every widget roughly proportional to its height. A menu and a text-entry have a weight equal to 1 and an image has a weight equal to 4. The weight of every column is the sum of the weights of the widgets that it contains. The stacking of the widgets must produce stacks with heights almost equal. The goal is to obtain an almost rectangular shaped context. The search of this configuration is not very heavy, thanks to the fact that generally the number of widgets inside a context implementation is relatively low. While determining the layout of a context all the contained ones are treated as single widgets. This makes possible to perform a complete search to find out the best configuration. At the beginning all the possible configurations of the widgets of the context are generated.
The configurations whose columns have a weight smaller than 3 are discarded.
Then the standard deviation of weights of the columns for each configuration is computed, and the configuration with the lowest standard deviation is chosen. An example layout for an application in the medical diagnosis support domain is shown in Fig.2.

3. CONCLUSIONS AND FUTURE WORK

The presented paradigm offers several benefits to the designers and programmers of interfaces. The SDL engine provides undeniable utilization simplicity. The programmer has only to describe the contexts, using a very simple language consisting of only four words without delving into the large number of the underlying API instructions.
The second benefit offered by the SDL engine is that the arrangement of the widgets inside the interface, i.e. the graphical layout, is totally delegated to the computer. The SDL engine takes also care of this task while only the task of defining the semantic of the variables dealt by the interface, providing a definition of the variables and of the links between them, has been left to the programmer.
As future work, we plan to extend the presented paradigm including user models. This would hopefully foster the development of more customizable and effective interaction modalities. Other research directions include the improvement of usability and design criteria for the choice of widgets and the definition of the graphical interface layout.

REFERENCES

[Ard02] Ardizzone, E., Peri, D., and Pirrone, R.: User Interfaces for SDL Applications, *KES2002 Knowledge-Based Information Engineering System & Allied Technologies*, Podere di Ombriano, 2002.
[Ard01] Ardizzone, E., Peri, D., Pirrone, R., Palma, A., Peri, G.: A Knowledge based Approch to Intelligent Data Analysis of Medical Images, IDAMAP 2001, London, September 4th.
[Plg97] Console, L., Lamma, E., Mello, P. and Milano, M.: Programmazione Logica e Prolog, UTET UNIVERSITA' (1997).
[Amz] Building Expert System in Prolog, http://www.amzi.com.
[DixHCI] Dix, A. J., Finlay, J. E., Abowd, G. D. and Beale, R.: Human-Computer Interaction, Prentice Hall.
[Gtk] The Gimp Toolkit, *http://www.gtk.org.*
[Kra88] Krasner G. E. and Pope S. T.: A cookbook for using the model view controller user interface paradigm in Smalltalk-80, Journal of Object-Oriented Programming, No. 1(3), pp. 26-49, 1988.