Reconfigurable User Interfaces for Databases

David England and Richard Cooper

Department of Computing Science
University of Glasgow
Glasgow G12 8QQ, UK

de@uk.ac.glasgow.dcs
rich@uk.ac.glasgow.dcs

Abstract

The fixed nature of user interfaces to databases often means that they cannot easily be made to fit the requirements of a particular group of users or tasks. This paper looks at an approach to the interactive reconfiguration of such interfaces. Thus the creation of novel interfaces to the same database can be greatly speeded up and simplified. The paper shows how these concepts can be employed in the representation of schema design and queries to a semantic data model.

1 Introduction

User interfaces to databases, even those using current graphical software technology, are often fixed by the designer and may not always meet the needs of the end-user. If changes are required they can be expensive to implement. Most interactive systems offer a limit degree of tailoring for organisational or individual requirements but this tailoring does not affect the fundamental usability of the system. Users will reject a system with a poor or inappropriate user interface as surely as they would a system with poor underlying functionality. This paper will discuss an approach to the dynamic reconfiguration of database interfaces. The central aim is to enable the computer-literate end-user (be they schema designer or query user) not only to easily tailor the user interface but also to radically alter the interface.

The task of adding a new interface to a database system takes place in the context of three sets of facilities. Firstly, there is the database system itself, with its internal model of data. Secondly, there is the model of data that the interface is intended to export to the user. Finally, there are the facilities that the development system provides for user interaction. Increasingly, these form a sophisticated set of facilities, perhaps being provided as a toolbox or being associated with a window manager.

When creating an interface, the implementer is, essentially, providing the following:

1) a mapping from the user data model to the internal model;
2) an attachment of the facilities associated with the data model with the facilities available for user interaction.

It is the latter task that this paper is concerned with, taking the start point that there are two sets of already existing code - one which implements the functionality of the user data model and one which implements a toolbox of concrete user interactions. These two sets must be connected to produce the user interface to the data model. Currently this connection is created by gaining access to the two sets of code by some code inclusion mechanism and then writing the program which wraps all of this up in a high level language. This interface program contains three parts:
1) a selection of the operations which will be available;

2) a parameterisation of the required user interface primitives (setting positional or shape information for instance);

3) an overall architecture to bind everything together.

Clearly, the latter two take most of the programming effort - parameterising a user interface is known to be a time consuming process, while building the superstructure is creating the main essence of the interface program. Yet this last task is essentially repetitive from one interface to another, while the parameterisation should be supported in such a way as to make it easy to test the effects of the parameterisation - a feature not provided in most high level language systems.

We therefore propose a system which provides sympathetic tools for glueing together the code which is about data management with the code for user interaction. Such a system allows the user to create connections between data modelling constructs and user interface facilities. This would include icons to represent different categories of data model construct, pictorial organisation styles to represent database structure and user dialogues to represent operations on the database.

This initial proposal for such a system, termed Configurable Data Model [Coop90] was outlined in terms of a simple class-based prototype implemented in the persistent programming language, PS-algol. In this paper, we turn instead to using a recently developed tool from the world of user interface design to achieve the glueing together. This tool, Iconographer [DrWa91], has been designed for visualising data files in a variety of ways, but will now be used to create the data model interfaces themselves.

2 Review

The approach discussed in this paper is a synthesis of the work in two areas: user interface design tools; and semantic models of data.

2.1 User Interface Tools

One goal ofuser interface research over the last ten years has been the production of dialogue models of human-computer interaction. The aim of dialogue models has been to provide an abstract layer between the specific functionality of an application and the physical characteristics of users environment. Thus a dialogue model based on transition networks [Jaco86] describes the sequence of user input actions and links this sequence to application components and user interface feedback. Other models have been based on event-action rules and context-free grammars [Gree86]. This work has culminated in the production of a number of User Interface Management Systems (or UIMS). Most UIMS, though they provide a degree of abstraction in user interface production, are rather inflexible in their architecture's, thus forcing user interface designers to make certain compromises at the design stage. One particular source of inflexibility, in the context of database interfaces, is the insistence on insulation between the user interface and the underlying structure of the application. This questionable strategy is at its least appropriate when applied to a general purpose application such as a DBMS, in which the data model and user interface would more usefully be intimately related [KiNo89].

More flexible than a UIMS, but less abstract, are user interface toolkits such as OSF/Motif [OSF90], Macintosh Toolbox [AppI85] and XView [Hell90]. These provide a common set of components (rather than an actual kit of tools) with which to build a user interface. Their level of abstraction is that of one or more function calls between each component and the application. Most interfaces built by UIMS or toolkits can be tailored