Using a Persistent System to Construct a Customised Interface to an Ecological Database

Peter J Barclay Colin M Fraser
Jessie B Kennedy

Department of Computer Studies, Napier University

Edinburgh, Scotland EH14 1DJ

Abstract

New applications of information technology have increasingly sophisticated user interface requirements. Such interfaces are sometimes highly application specific and therefore require to be customised to the given application area. Technology is required to support the construction of such highly non-generic interfaces. We describe how by using the persistent programming language Napier88 and the associated windowing system *WIN* it has been possible to construct by modest effort an interface tailored to the requirements of an ecological database.

1 Introduction

The application of information technology to new areas makes increasing demands on user interfaces. In constructing data-intensive applications, there is a need for techniques to access data through visually meaningful representations, rather than the query languages of traditional databases.

User Interface Management Systems (UIMS) [HaHi89] provide strong support for the construction of highly generic interfaces; in [Coop90] Cooper describes how to build generic interfaces on top of a variety of data models in a persistent system.

However, as well as supplying a high degree of genericity, it is also necessary to find technologies better to support specificity, such as applications where particular graphical representations must be used. Here we describe how the persistent programming language Napier88 and the associated windowing system *WIN* have been used rapidly to develop an interface, *EcoSystem*, which provides highly application specific querying facilities over an ecological database. Only recently has database technology been applied to ecological data [Kenn85]; as yet, there has been little attempt to build interfaces such as the one described here.

2 Ecological Data

In this section an overview is presented of the ecological database to which the graphical interface is a front-end.

2.1 The Data

The data used was collected in a survey undertaken in 1986 and 1987 by Paisley College of Technology, in conjunction with the Nature Conservancy Council. Here we give a brief overview of the structure of a subset of the data; more details may be found in [BiCM88].

In this examination we consider only the data relating to land classification. A sample of 1 km $\times$ 1 km quadrats was selected, and each divided into 2500 50 m $\times$ 50 m squares (called *pixels*).

These were surveyed for a variety of biotic features, such as woodland and vegetation types present, and abiotic features such as physiography and boundary structures. This information was collected on coding sheets, using a different coding scheme for each quadrat surveyed. Subsequently, the data was reduced to a standardised representation scheme and subjected to TWINSPAN (Two Way INdicator SPecies ANalysis) [Hill79], to determine various categories for the different features of interest. This procedure is described fully in [BiCM88].

The data was modelled in terms of the object-oriented model described in [BaKe91]; full details of the application model is beyond the scope of this paper, but is described in detail in [BaKe92]. A brief summary follows (see figure 1).

A quadrat is a square aggregation of pixels. Each quadrat has an *xRef* and *yRef*, representing its Ordnance Survey grid co-ordinates, and a *landType*. Each pixel has a *patchType* for each of the categories of features included in the survey, together with an *xIndex* and *yIndex* showing its position within the quadrat. *LandType* and *PatchType* are the domains of various discrete values, representing the different types of land classification established by the survey (eg, Trichophorum-Eriophorum Bog, Saltmarsh, *etc*). *Zones* and *patches* are homogeneous aggregations of quadrats and pixels respectively.

Our aim in modelling the data is not to redo the classificatory analysis already performed upon it, but rather to be able to represent the results of this initial work in an intuitive manner, allowing graphical display of the data, and *ad hoc* querying on it. Any conclusions derived from the data must be supported by the rigorous statistical techniques employed by ecologists [Gauc82], [Piel84]; we hope to offer a facility to browse the data in order to perform explorative "searching for patterns."

2.2 The Database

The term database is used loosely here to describe *Isis* (Islay Survey Information System) [Barc92], an application in Napier88 which manages the ecological data under the appropriate model. The data is represented as a collection of persistent Napier88 structures. Privileged utility programs may access these structures directly; access for all other applications is through an object oriented message interface, which forces a view of the data consistent with the application model. *Isis* contains no notions of any graphical representation

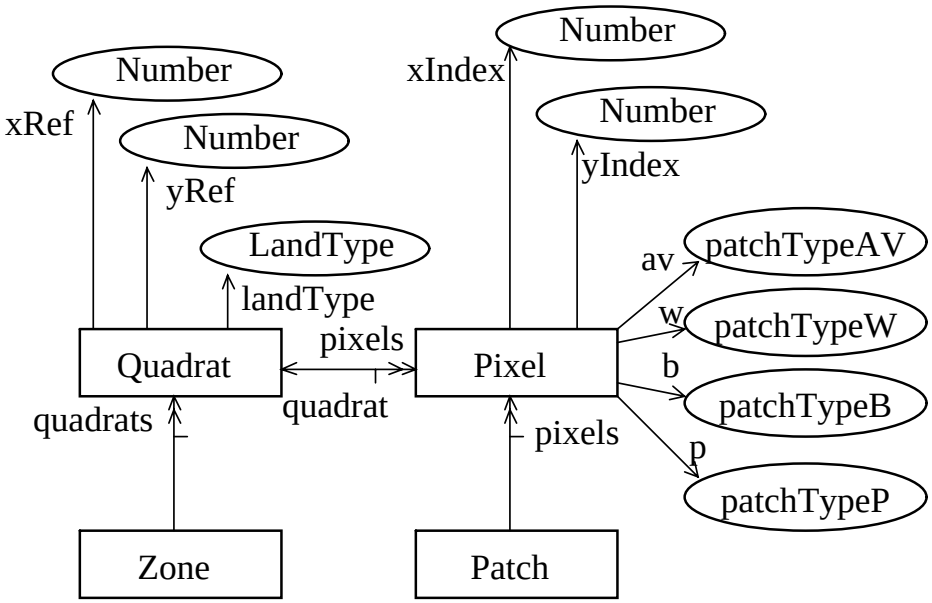

Figure 1: Application Model

of the data. An intermediate system termed a *query engine* translates from the internal query representation used in *EcoSystem* to that used in the data management application, decoupling the two applications (figure 2).

3 Napier88 and *WIN*

In this section the language Napier88 and the windowing system *WIN* are briefly reviewed.

3.1 Napier88

Napier88 [MBCD89], [DCBM89] is a high-level block-structured language with orthogonal persistence. Napier88's type system is based loosely on one proposed by Cardelli and Wegner [CaWe85], and provides an infinite union type, a type for extensible bindings, variants, recursively defined types, parametric polymorphism, existential types [MiP188] and first-class procedures.

The operation of the language's persistence mechanism is transparent to the language-user; any object created by the execution of a program may be specified as persistent, in which case the object will continue to exist after the termination of the program. The persistent object may then be reused in a subsequent execution of the same program, or by some other program. Napier88's persistence is type-safe, so that any program execution which uses a persistent object may use it only as an object of the type with which it was created.

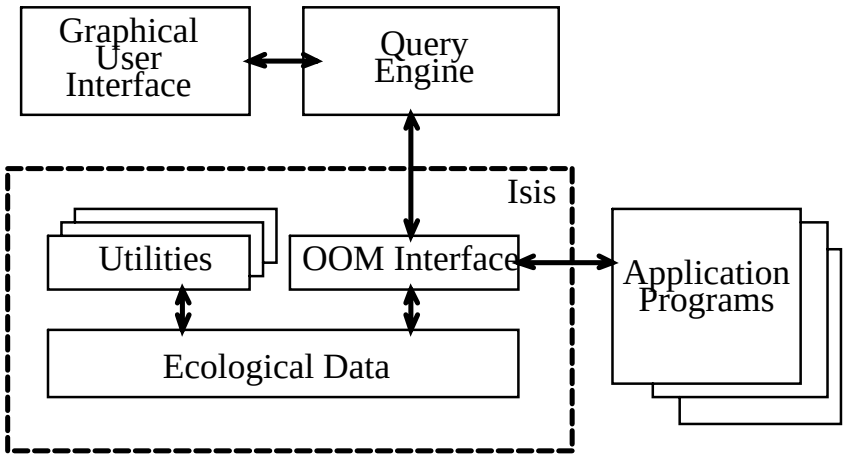

Figure 2: System Architecture

Napier88 supports two graphics types, *image* (a raster type), and *pic* (a point-and-arc type). These types are fully integrated into the language; they may be passed as parameters, returned from functions, and they may persist. The graphics types facilitate graphical programming, but are rather low-level to serve as primitives for interface construction.

3.2 *WIN*

WIN (Windows In Napier) [CDKM89], [CuDK90] is a windowing system written entirely in Napier88; it is based on the experience of earlier user interface tools implemented in PS-algol [CoMA88].

Each window has associated procedures to set its title, display images in its pane, resize it, *etc.* Further, each window has an application which runs within it; this is a procedure which takes a (keyboard or mouse) event as its parameter, and implements the functionality of the window.

Windows are displayed, opened and closed, and moved by a window manager. *WIN* has extended the traditional window manager paradigm by making windows persistent, so that they may be reused by different applications; further, windows are first-class objects which may be passed between different window managers. A window manager may run inside another window, permitting an arbitrary hierarchical nesting of windows.

Event-distribution among windows is controlled by a notifier hierarchy, which is described in detail in [CuDK90]; this mechanism is largely transparent to *WIN* users. (However, the user can take control over event distribution if desired; for example, *EcoSystem* disables the left mouse button by placing a trap for any left mouse events at the top of the event distribution hierarchy).

WIN also provides a range of specialised windows, such as light-buttons, sliders, menus and scrolling menus. The application associated with these specialised windows implement the functionality of the particular interface tool.

WIN is constructed on top of the integral graphics primitives of Napier88,

building up to the level of components sufficient for the construction of graphical interfaces.

4 A Customised Interface to Ecological Data

This section describes the graphical interface *EcoSystem*; a more detailed description, together with an overview of the methodology used to map the interface design onto its implementation, may be found in [Fras91].

4.1 The Design of the Interface

In this section the design of the interface is reviewed; its implementation is discussed later.

There are several abstract models of the human computer interaction; some are surveyed in [Cout89]. Here, however, the problem is to facilitate construction of an apposite, specific interface by persons who are not interface specialists.

The design of the interface reflects the task analysis. Discussion with the intended users of the interface identified the kinds of tasks which were to be carried out[1]; these were broken down into appropriate constituent subtasks where appropriate. A graphical representation for each task was chosen, and a mock-up of the interface component for this task constructed for user-feedback. Individual task windows such as the query editor or session window (query executor) were designed in this way.

The interface displays were built up from components such as those supplied by *WIN*. Hence, task analysis proceeded top-down by functional decomposition, and interface-design bottom-up from available components. Where subtasks were sufficiently structured, they were supported by appropriate interface subdisplays; in this way, the design process continued recursively.

Light buttons in each display call up task windows appropriate to each task which may be performed there; the backdrop map associates each task with its geographic locality (figure 3).

4.2 The Design of the Software

Since no guidelines were available on how to map an interface design onto *WIN*, finding an approach to this mapping constituted a fair amount of the development effort. The approach used is described in some detail in [Fras91], and could form the rudiments of a graphical user interface design methodology for use with *WIN*.

The structure of the software reflects naturally the structure of the interface which it implements. Environments (Napier88 collections of name-object bindings) were created to store interface objects and procedures separately. Further environments were created for tasks supported by the interface, and where necessary, hierarchically nested environments for subtasks. The implementation of one particular task was therefore generally hidden within one environment.

[1] These tasks are covered in a succeeding section on the functionality of the interface.

Figure 3: Screen Layout

Figure 4: Inter-Module Communication

The objects necessary to produce the display for a given task were constructed first. These objects comprise of the task window, together with its window manager and notifier, menus, light buttons, *etc*, which are created from the generator procedures supplied by *WIN*; and other items for graphical display such as text and images. These items were then assembled in the correct arrangement to form the display, and stored in the task's environment.

The procedures necessary to represent the activity of the task were then coded, compiled, and bound to the menu panes and light buttons which evoke their functionality, in the task window where these activities are to be performed. The functionality of a procedure may, of course, be to display a subtask window, thereby making further subactivities available. Where a task involves interacting with the underlying database *Isis*, the activity-procedures call the data-access procedures supplied by the application interface of *Isis*.

A top-level module controls the initialisation and finalisation of the system as a whole on each execution, and monitors the keyboard and mouse.

Objects which exhibit a sufficient degree of reusability (such as calibration information) were placed in a global pool in the store, so that they could be bound into various different modules (figure 4). All binding between system components has been arranged to be dynamic, to support incremental evolution of the system better.

The overall structure of the system is shown in figure 5, which shows the visibilities among system components.

4.3 The Functionality of the Interface

In this section we review the functionality of the graphical interface. Basically, two classes of function are supported by *EcoSystem*; these might be categorised as querying functions, and supporting functions. The supporting functions are

Figure 5: Module Visibilities

described first, since generally they are simpler.

4.3.1 Supporting Functions

The first class of supporting functions concern the map of the survey site and its geometry. Two functions which need be used only when the system is installed are *load map* and *calibrate*. The *load map* function converts a raster image of the survey site into a Napier88 image and makes this image persistent so that it may be used in subsequent program executions. The *calibrate* function calculates the scale and offset of the resulting image, so that the position of any point may be calculated with respect to the Ordnance Survey grid.

The second, related set of supporting functions are of more general utility. The function *grid* allows a representation of the ordnance survey grid to be toggled on and off on the backdrop map. The function *co-ord* will furnish the grid coordinates of any point on the map selected by the mouse, and the function *distance* will furnish the (actual) separation in kilometers of any two points which are selected on the map (figure 6).

The help system is the remaining supporting function. Help is provided in two ways. Firstly, prompts and messages are displayed appropriate to the choices available to the user at any time, guiding her use of the system. Secondly, more detailed reference information on each task is available in a scrollable text window which may be invoked at any time (figure 7).

Figure 6: Geographic Supporting Functions

Figure 7: Help Function

4.3.2 Querying Functions

The construction and execution of queries are separated. There are three reasons for this separation. Firstly, the construction and execution of queries were identified by task analysis as two separate activities. It was considered that the user may wish to construct a query or set of queries to examine some particular hypothesis, and that during this task, identifying the queries necessary would be an atomic task rather than examining the result of each individual query as it was constructed. Secondly, the queries are persistent queries, so that once a query has been formulated, it may be run as often as required, and specifically on different executions of *EcoSystem*. The actual interrogation of the database, however, takes place at query execution time, so the result of the query will always reflect any alterations in the underlying data. It is envisaged that further tasks may be required between query executions. The third, and perhaps most important, reason for this separation is so that a query or set of queries that have been run over one survey 1 km $\times$ 1 km quadrat may then be run over any other quadrats desired.

(Having said that construction and execution of queries are two separate functions, it is to be understood that at any time the user may switch between these two modes).

4.4 Query Formulation

The initial stages in constructing a query are shown in figure 8. Here, the query editor displays a library of icons, which may be used to represent the query result; the desired icon is selected with the mouse. The query editor window shows the various categories for which data is available; this is effectively a dictionary showing the metadata of the underlying database. The required category is selected with the mouse.

In figure 9 the query window shows the query as formulated so far. The associated icon is shown in the top left pane of the window, the features sought in the main pane, with logical connectors between. The query editor window shows for which features data is available within the category already selected. Features may be incorporated into the query under construction by selecting them with the mouse.

Thus the query editor effectively allows the user to browse the underlying database, establishing for what features data is available, and to construct queries which show the geographical distribution of (Boolean functions of) these features. The queries in existence, and their definitions, may be viewed at any time as shown in figure 10.

4.5 Query Execution

The function called *session* allows queries to be executed over the desired 1 km $\times$ 1 km quadrat. There are two query positions, in each of which a query may be selected to run, or be removed. Once run, the result of the query may be shown on or cleared from the display. The display shows the 50 m $\times$ 50 m pixels into which each quadrat is divided by the survey methodology; in these pixels the icon associated with the query may be displayed. A cross-hatch icon

Figure 8: Query Construction

Figure 9: Query Construction

is used where a pixel is positive for both queries simultaneously. By way of example, figure 11 shows where streams pass through semi-natural woodlands.

5 Evaluation of the Interface

Isis supplies an interface to ecological data, supporting an object oriented model; this interface may be used by (batch) application programs. *EcoSystem* accesses the data through the same interface, providing a user with the ability to query the data graphically under this model.

The object oriented model treats sampled areas as objects, with their observed features as properties. Thus queries can reference properties of areas discovered in different stages of the survey, or indeed by different surveys. This "multi-dimensional" view of the data was not available before the implementation of *EcoSystem*. The ability to have derived views of the data is also supported.

The kinds of queries allowed, and the method of display of results, emerged from discussions with the system's intended users. Clearly, changes in these requirements will necessitate alterations to *EcoSystem* itself. However, the facility with which the system was constructed lends encouragement that such changes could be met by modest effort. The system has provided a rapidly-constructed answer to highly specific user requirements. We are confident that the system will be able to evolve easily to meet changes in these requirements, although confirmation of this can only be provided by experience through the lifetime of the system.

6 Evaluation of *WIN* and Napier88

WIN supplies sufficient primitive components to support a project such as *EcoSystem*. There was no need to code any of the building blocks from which the interface was constructed. The user need have no knowledge of how these components work to design a system which uses them, and the details of the event distribution mechanism are also largely transparent to the user. Thus the user can concentrate on designing the functionality of the system, rather than being distracted by implementation details[2].

In terms of the taxonomy presented in [Talb91], *WIN* is a toolkit — the lowest in a hierarchy of five levels of support for user-interface construction. Nonetheless, owing to the seamless integration of *WIN* with the application language, Napier88, a high level of support is offered.

The ability to build libraries such as *WIN* depends heavily on Napier88's expressive type system, which simplifies the description of objects required, and allows the user to construct her own types as required by a particular application. First-class procedures allows the construction of objects which represent activities.

The availability of both first class procedures and first class graphical objects, which may be dynamically bound to each other, made possible the approach where tasks and displays were designed top-down and bottom-up respectively, separately but in mutual co-ordination.

[2]However, an experienced user can modify *WIN* at a low level if desired.

Figure 10: Query Browsing

Figure 11: Query Execution

The provision of orthogonal persistence considerably lightens the system-developer's load, since she need expend no effort in storing and retrieving system components (including images), nor in preserving the system's state. This results in a considerable reduction of code size.

EcoSystem is implemented by around 7000 lines of Napier88. However, the degree of modularity is high. The number of separate modules (compilation units) is 42, so that each one has on average only 170 lines of code. All but one of these units create the various objects defined by the system design, and place them in the persistent store; (apart from debugging), each module need be executed only once. The remaining module is the top-level executable program; running this module fires up the entire system, and controls its shutdown.

This demonstrates that persistence encourages a highly flexible and incremental style of software development; a system may be constructed by small chunks. This considerably simplifies not only construction, but also tuning and evolution of the system.

The main problems met in the construction of the system were in the lack of further tools which would have been useful; for example, neither an image-editor nor a browser for the persistent store were available during the development of *EcoSystem* (although a browser has been developed — see [KiDe90]).

7 Conclusion

Application specific graphical interfaces can considerably facilitate the access of data in a way meaningful to its users. Although such interfaces are novel for an ecological database, it has been possible to construct such a system by modest effort.

A main contributor to the ease with which the system was developed is the seamless integration of the database, the interface toolkit, and the application interface. The database is written in Napier88, and long-term data storage is managed by the persistence mechanism of the language. The interface toolkit is built on Napier88's integral graphics primitives. The application interface, written in Napier88, utilises both the stored data and the interface components as required, simply and without discrimination. This experience suggests that Napier88 provides an efficient technology for the construction of application specific graphical interfaces.

8 Further Work

It remains to extend *EcoSystem* to deal with the whole body of survey data, rather than a subset; and to extend it to support more complex, computation-based querying as described in [BaKe92].

9 Acknowledgements

We wish to thank the following people: Prof. David Curtis of Paisley College of Technology for his collaboration in this work, and for making available to us some of his data; Graham Kirby of St Andrews University for his unfailing advice and assistance in using *WIN*; Richard Cooper and Paul Philbrow of

Glasgow University, for the benefit of their experience in persistent programming; and Prof. Malcolm Atkinson of Glasgow University, for his guidance.

References

[BaKe91] Barclay PJ, Kennedy JB. Regaining the conceptual level in object oriented data modelling. In: proc BNCOD-9, Wolverhampton. Butterworths, Jun 1991

[BaKe92] Barclay PJ, Kennedy JB. Modelling ecological data. In: proc 6th international working conference on scientific and statistical database management, Ascona, Switzerland, Jun 1992

[Barc92] Barclay PJ. Object oriented modelling of complex data with automatic generation of a persistent representation. PhD thesis, Napier University, Edinburgh, 1992 (forthcoming)

[BiCM88] Bignal EM, Curtis DJ, Matthews JL. Islay: land types, bird habitats, and nature conservation. Technical report, Paisley College of Technology, 1988

[CaWe85] Cardelli L, Wegner P. On understanding types, data abstraction, and polymorphism. Computing Surveys Dec 1985; 17(4)

[CDKM89] Cutts QI, Dearle A, Kirby GNC, Marlin CD. WIN: a persistent window management system. Technical report, University of St Andrews, 1989

[CoMA88] Cooper RL, MacFarlane DK, Ahmed S. User interface tools in PS-algol. Technical report, University of Glasgow, Mar 1988

[Coop90] Cooper RL. Configurable data modelling systems. In: proc 9th International Conference on the Entity-Relationship Approach, Lausanne, Switzerland, Oct 1990, pp 35 – 52

[Cout89] Coutaz J. Architecture models for interactive software. In: Cook S (ed) ECOOP89: proc 3rd European Conference on Object Oriented Programming. Cambridge University Press, 1989, pp 383 – 399

[CuDK90] Cutts Q, Dearle A, Kirby G. WIN programmers' manual. Technical report, University of St Andrews, 1990

[DCBM89] Dearle A, Connor R, Brown F, Morrison R. Napier88 - a database programming language? In: proc DBPL 2, Gleneden Beach, Oregon, Jun 1989

[Fras91] Fraser CM. Persistent systems for graphical interface construction. Technical report, Napier University, Edinburgh, May 1991

[Gauc82] Gauch HG. Multivariate analysis in community ecology. Cambridge University Press, 1982

[HaHi89] Harton HR, Hix D. Human-computer interface development: concepts and systems. ACM Computing Surveys Mar 1989; 21(1):5 – 92

[Hill79] Hill MO. TWINSPAN - a FORTRAN program for arranging multivariate data in an ordered two-way table by classification of the individuals and attributes. Technical report, Section of Ecology and Systematics, Cornell University, New York, Jul 1979

[Kenn85] Kennedy JB. A study of ecological database management and associated data analysis. Master's thesis, Paisley College of Technology, 1985

[KiDe90] Kirby G, Dearle A. An adaptive graphical browser for Napier88. Technical report, University of St Andrews, 1990

[MBCD89] Morrison R, Brown F, Connor R, Dearle A. The Napier88 reference manual. Technical report, Universities of Glasgow and St Andrews, Jul 1989

[MiPl88] Mitchell JC, Plotkin GD. Abstract types have existential type. ACM TOPLAS Jul 1988; 10(3):470 – 502

[Piel84] Pielou EC. The interpretation of ecological data: a primer on classification and ordination. John Wiley and Sons, 1984

[Talb91] Talbot S. Software design aspects in HCI. Technical report, BA SEMA, Glasgow, 1991