Using a Conceptual Data Language to Describe a Database and its Interface

Kenneth J. Mitchell, Jessie B. Kennedy and Peter J. Barclay

Computer Studies Department, Napier University
Canal Court, 42 Craiglockhart Avenue, Edinburgh EH14 1LT, Scotland, UK
e-mail: <kenny,jessie,pete>@dcs.napier.ac.uk

Abstract. We propose a conceptual approach to defining interfaces to databases which uses the features of a fully object oriented data language to specify interface objects combined with database objects. This achieves a uniform, natural way of describing databases and their interfaces. It is shown how this language can be used in the role of data definition and, when combined with interface classes, in the definition of database interfaces. A prototype developed to test this approach is presented.

1 Introduction

The presentation and manipulation of information are central to interacting with database systems. The efficacy of this interaction is determined by the interface to the database and therefore the design and usability of such interfaces requires investigation to ensure optimal facilitation.

The traditional approach to providing interfaces to databases is through a variety of textual language interfaces, such as SQL [17] and DAPLEX [43]. However, as these interfaces rely heavily on the user to retain knowledge of the structure of the database together with the syntax of commands, they are difficult to use without being accomplished in this style of interaction.

In an attempt to improve the interaction with databases, forms based interfaces were developed. A few notable results have been found on providing graphical interfaces to relational databases, covering specific applications, HIBROWSE [20], Office-By-Example [49][56], FORMANAGER [53] and general applications, Query-By-Example [55], TIMBER [46], GUIDE [52], Santucci and Palmisano's Visualiser [42].

Graphical interfaces that support complex models of information have flourished in recent years, with the application visualisation system (AVS) [48], and application visualisation environment (AVE) [19], for dataflow visualisation. Surveys on

complex graphical interfaces provided by Pickover [37], and Wolfe et al [51] provide insights into their potential.

With the development of object oriented databases systems, the variety and complexity of information modelled has increased. Two dimensional graphical interfaces to object oriented databases have been developed for specific applications, e.g. in EcoSystem [5] and the Banksia geographical information system [54], as well as general applications, such as schema designers and browsers, e.g. Almarode's Schema Designer [1], the O2 browser [18], the CLOSQL interface [32], and Kirby and Dearle's Napier88 browser [28].

Recently, software and hardware technology has improved to the point where the practical use of applications with interactive three dimensional graphics is a viable prospect [41]. With this enabling technology and the promise of improved interaction, 3D interfaces to databases are beginning to emerge, e.g. WINONA [38][39], AMAZE [12][13], PIT [9], Bead [14], GRADE-3D [45], and LyberWorld [24].

Many languages exist for the specification of graphical interfaces, from interface programming languages, OWL [35], OSF/Motif [34], and graphics languages GKS [2], PHIGS [3], to state-transition notations, such as the Storrs-Windsor notation [50], which are surveyed by Green [22]. Higher level abstractions are found in languages for user-interface management systems (UIMS), such as, COUSIN [23] and MIKE [33] based on interaction sequence specification, and HIGGINS [25] and the UIDE [21], which are based on data models. However these languages are designed purely for interface design independent of database interface requirements.

An interface is a rather elusive entity. One interface style may be preferable to another rather different one depending on the particular user's requirements. Considerable effort has been directed towards the customisation of interfaces to databases, eg. EVE [36], CDMS [15], Dbface [26][27], and NIOME [30]. Attention to this concern has been realised in the method of integration between data and interface languages, i.e. by the close coupling, but ultimate separation of data and interface objects. This permits the replacement of one set of interface objects for another to support alternative interfaces to the same data.

Given this accumulation of many diverse interfaces to databases there is an identifiable need for a concise language for database interface description. The language would expect to achieve the same degree of specification which exists for the description of data, in that interface objects may be defined with various properties and behaviour. For this reason the authors believe that an interface to a database should be specified in part of the database's schema definition and must be able to specify any level of interface or data visualisation sophistication.

This broadly applicable approach enables the definition of database interfaces from simple command line textual interfaces to 3D interfaces. However, our work is currently focusing on investigating the potential of 3D interfaces to databases [39]. The work presented in this paper describes a prototype 3D interface to an object oriented database in which both the interface and the database is described using NOODL [7], a simple data language intended to allow object oriented modelling of data at a conceptual level.

The following section introduces NOODL by showing an example schema definition. Section 3 continues this example with the introduction of interface classes for the combined definition of a database and its interface. Section 4 presents the prototype interface and how the language description may be mapped to an implementation. Finally, conclusions and further work are discussed.

2 NOODL as a Data Language

The Napier Object Oriented Data Language (NOODL) is a simple language which allows object oriented modelling of data at a conceptual level; it is introduced in [4], described fully in [7] and most recently in [8] in its role as a query language. A NOODL schema contains a list of class definitions, which show the name and ancestors of each class. A class definition also includes a set of properties, operations, constraints and triggers. Some of the details of this language are exemplified in a NOODL schema describing a company database which holds information about the departments in which employees are located (figure 1).

Departments and people have names and each employee has a job title; all of these are represented by text strings. A person's age is represented by a number. Every department has a set of employees and every employee is associated with a department. This is represented by the pair of obverted properties, *employees* and *department*. A department can perform a query through its *list_mature_employees* operation to return the set of all employees associated with that department, who are older than 25. An employee may be transferred to any department, achieved by simply changing the value of their department property through its transfer operation. The fact that this is an obverted property implies this change will also result in removing the employee from the current department's employee set and adding them to the new department's employee set. The *person's age constraint* (PAC) defines the permitted limits for a valid age. This constraint is overridden in the employee class, in order to disallow employees with an age of less than 18. An employee may retire through object migration to person status. This happens automatically once they reach the age of 65, through the use of the *retiral* trigger. Employees may be promoted to managers and managers may be demoted to employees through their *promote* and *demote* operations, respectively. Triggers and constraints are detailed in [6].

```
schema Company

class Department
properties
   name      : Text ;;
   employees : # Employee \ department
operation
   list_mature_employees : # Employee is
     Employee where self.employees.age>25

class Person
properties
   name : Text ;;
   age  : Number
constraint
   PAC is 0<self.age and self.age<120

class Employee
ISA Person
properties
   job_title   : Text ;;
   department  : Department \ employees
operation
   transfer Department d is self.department(d) ;;
   retire    is self.goto(Person) ;;
   promote   is self.goto(Manager)
constraint
   override PAC is 18<self.age
trigger
   retiral is self.age>65 :: self.retire

class Manager
ISA Employee
operation
   demote is self.goto(Employee)

end_schema { Company }
```

Fig. 1. A company schema definition

3 NOODL as a Database Interface Language

In order to provide an interface for the data described in figure 1 it is necessary to describe how the data should be visually represented in the interface. The authors believe that a set of objects in an interface should be described conceptually in a similar manner to the objects contained in a database, therefore NOODL has been adopted as a possible language with which to describe the database interface.

An interface object contains only those features pertaining to the behaviour and visual representation of the database object in the interface. Using NOODL, an interface object can be represented by a set of properties describing its visual

appearance, a set of operations describing its interaction with the user and other objects, a set of constraints describing how an object's behaviour or properties may be constrained in the visualisation and a set of triggers which respond to 'events' in the interface. Therefore, any interface should be able to be described by a NOODL schema thereby allowing the interface objects to be stored in the database along with the database objects. This provides a unified model of the data and interface (figure 2). This model states that each database object with a set of data related features has an associated interface object with a set of interface related features.

Fig. 2. Unifying Model of Data/Interface Objects

In using NOODL to describe data objects the properties and operations of the object are entirely free and dependent only on the semantics of the equivalent real world object being modelled. However, when defining interface objects there exists a 'well defined set' of interface properties and operations which may be used to describe interface objects, the subset used determining the sophistication of the visualisation. The values for these properties define the representation of the objects in the interface. The properties used in the prototype described here include:

- The shape of an object as it appears in the interface. This may be a 2/3D object, a dialog form, or simply refer to a textual description.
- The position of the object in 2/3D space (which may be translated).
- The size of the object (which may be scaled).
- The orientation of the object relative to interface space (which may be rotated).
- The colour of the object.
- The extent of the object which defines the bounding volume of the object in the interface.

This by no means defines the full range of interface properties which may be of use in a conceptual description, e.g. ambient, diffuse and specular coefficients, texture and properties of animated objects, such as, path, spin speed and velocity.

The properties of interface objects may be defined as part of the schema (see Appendix for example definition of position, shape, size, orientation and colour used in the prototype) or may be pre-defined in the language.

Interface object operations are those which either manipulate the interface object in some way or describe interface actions performed by the user to which the object will respond. Interface object manipulation operations are defined in the same way as data object operations, except that they manipulate some aspect of the interface. An interface action is described as a sequence of primitive interface actions. The actions in this example are move and select (see Appendix for definitions of these). To describe that a promotion operation requires the user to select an object (a mouse-click in a WIMP interface) the following might be written,

```
promote is self.select
```

An initial set of primitive actions identified are select, pick_up, drop, move, turn, time_click. With drop, move, turn, and time_click interface actions a value is associated. For example, if an object is moved to a position this may be described by,

```
self.move.position
```

Constraints on interface objects are merely the same as constraints on data objects, but are particular to interface object properties as opposed to data object properties. Examples of these have been shown in the definition of *Colour* and *Position* (Appendix).

In common with interface object operations and constraints, triggers may pertain to the values of interface object properties. In addition, triggers are used to respond to interface actions described in the operations section. For example, if the user performs a promote action on an employee, i.e. selecting it, the corresponding data operation may be triggered with the following,

```
PET is self.promote :: self.data.promote
```

This method of representing interface actions is similar to the approach used in many user interface programming languages [35], i.e. the event is declared for the operating system's event handler as an undefined operation which is defined in an event response operation. The code which states how an event will be recognised is hidden. This allows the designer to specify exactly what sequence of interface actions are required to perform an operation at an abstract level. This exemplifies the integration of data language and interface language. Once the concept of a trigger is

understood the modeller has knowledge to apply this to both responding to integrity violations in the data as well as responding to interface actions.

An Example Company Database Interface

This example (figure 3) defines a 3D interface with direct representations of departments and employees.

The data classes have been omitted for brevity, but are identical to those in figure 1 with the exception that each data class has an obverted property, *interface*, which couples a data object to its interface object. The obverted pair of *interface* and *data* is the first property in each interface class.

When referencing a data object's properties from its interface object via the *data* property, this data property may be omitted, i.e.

```
self.data.department.interface.extent
```

can be expressed in short-hand as,

```
self.department.extent
```

As every interface object has a data object and vice-versa, their associated properties may be referenced directly.

Departments are represented by a department shape positioned on the 'ground' in the interface and is constant for all departments. The colour and size are not specified thereby allowing different departments to take on different appearances in the interface. The *move department trigger* (MDT) states that when a department object is moved then the employees belonging to that department are triggered to move to a new relative position inside the extent of the department's new position.

Persons do not have a representation in the interface as we are only interested in employees. Employees are represented by blue human shapes with a vertical orientation. Therefore all employees look the same in this interface only their position may vary. The *in department constraint* (IDC) specifies the position of the employee inside the spatial extent of a department. The *promote employee trigger* (PET) states that on execution of the promote operation in the interface object the promote operation of the data object is triggered. Similarly the *transfer employee trigger* (TET) triggers the equivalent data operation, passing a parameter defining the new department to which the employee has been transferred. Manager objects are specialised kinds of employees shown by the ISA reserved word. This demonstrates that interface objects may be inherited and that their properties, operations, etc. may be overridden. For example, the manager's inherited colour property is overridden from blue to white.

```
schema Company

{ Data Classes }
...
{ Interface Classes }

class Department_Interface
properties
   data      : Department \ interface ;;
   shape     : Shape is “department” ;;        { constant }
   position  : Position is “on ground” ;;      { constant }
   size      : Size ;;
   colour    : Colour
trigger
   MDT is not self.extent.holds(self.employees.position)::
     self.employees.position.put_inside(self.extent)

class Person_Interface
property
   data : Person \ interface

class Employee_Interface
ISA Person_Interface
properties
   data        : Employee \ interface ;;
   shape       : Shape is “human” ;;          { constant }
   position    : Position ;;
   orientation : Orientation is “vertical” ;; { constant }
   colour      : Colour is “blue”             { constant }
operations
   promote is self.select ;;
   transfer is not self.move.position.is_inside
     (self.department.extent)
constraint
   IDC is
     self.position.is_inside(Department_Interface.extent)
trigger
   PET is self.promote :: self.data.promote ;;
   TET is self.transfer :: self.data.transfer
     (Department where its.extent.holds(self.position))

class Manager_Interface
ISA Employee_Interface
properties
   data : Manager \ interface ;;
   override colour : Colour is “white”   { constant }
operation
   demote is self.select
trigger
   DMT is self.demote :: self.data.demote

end_schema { Company }
```

Fig. 3. A combined company database interface definition

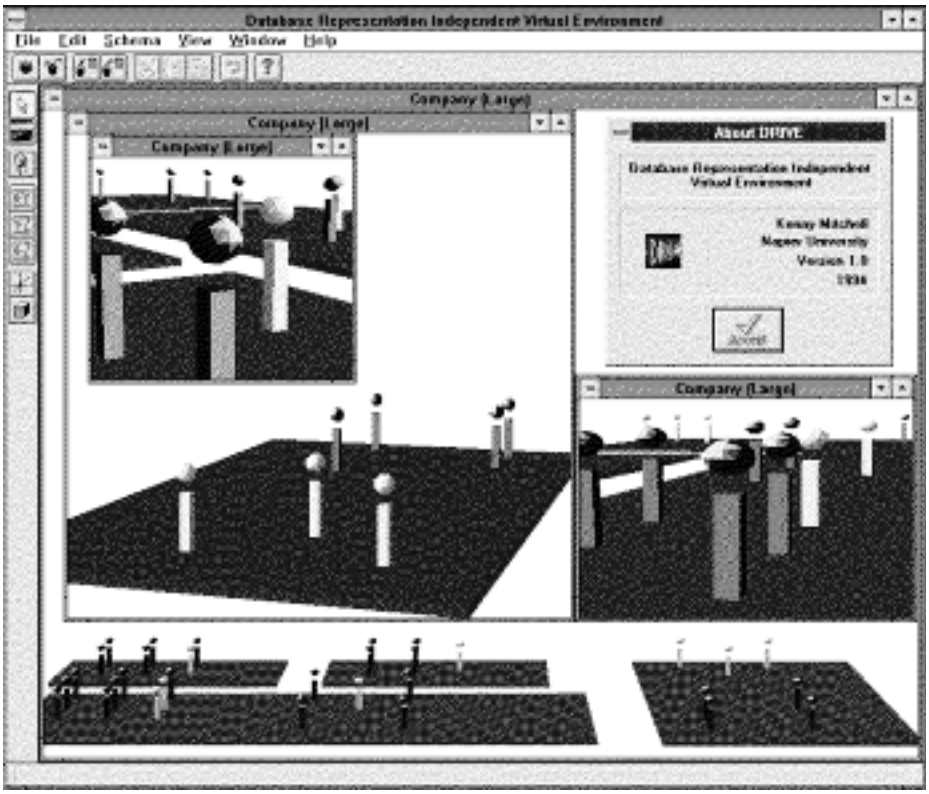

Fig. 4. Prototype company database interface

4 Prototype Interface

A prototype has been developed to test and evaluate the use of NOODL as a database interface language. It has been implemented on a PC-platform under Microsoft Windows in C++ [47], using the Borland's ObjectWindows Library [11] [35] and the RenderWare [40] graphics dynamic link library for the 3D representation of interface objects. A NOODL data model layer (similar to ObjectStore's Meta Object Protocol [29]) implemented with the POET [10] persistent C++ extension provides a means of dynamically creating persistent database interface schemata together with their data.

The prototype system allows the user to specify NOODL schemata for database interfaces interactively and automatically generates the resulting 3D representation from instances created by the user. Currently, this implementation supports only the structural aspects of NOODL. Therefore, for both data and interface schemata only the properties and inheritance semantics can be realised in the prototype without behavioural aspects, such as operations, constraints and triggers. There are two modes of operation in the system, *designer mode* which allows the user to

manipulate the database interface schema and instances and *interface mode* which simulates the specified database interface.

Figure 4 shows a display from the prototype for the schema described. The display shows several windows each depicting a different viewpoint of the interface. The main window shows all the departments in the database with their associated employees, the medium sized window shows a close up of one department while the smallest windows show close-ups of some employees. Any number of windows of different database interfaces are permitted and the user is free to navigate anywhere in the 3D space. The representation for the employees and departments in figure 4 are purposely simple; however a more sophisticated visualisation of the interface to the Company database is possible.

Figure 5 displays an interface where the employees and departments can be represented by more sophisticated shapes. Selecting an object in designer mode shows its instance details. The dialog box in the top right of the display shows an Employee data instance and provides access to its properties and associated interface instance through the middle-right dialog box. The dialog box on the bottom right of the display shows a preview facility in which the shapes available for assigning to objects may be viewed.

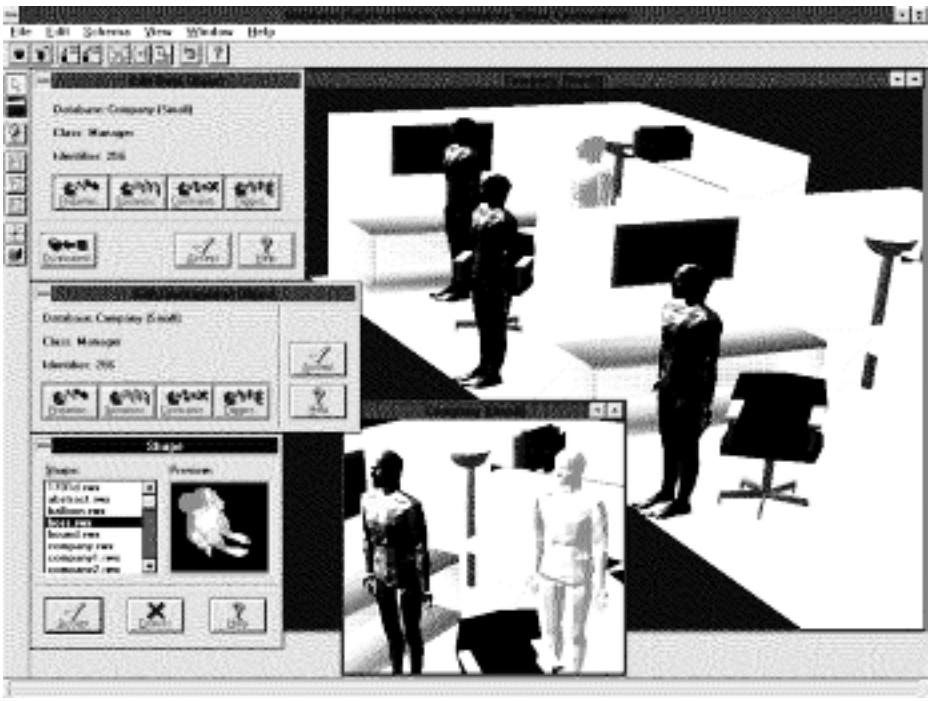

Fig. 5. Alternative company database interface

The performance of this software for interacting with large scale complex collections of data is determined by the efficiency of both hardware and software architectures supporting graphical and database system components. With a high-specification system sophisticated interfaces (such as figure 5) may be used whilst maintaining a practical level of interactivity.

5 Conclusion

We have presented a unified model for the conceptual design of databases and their interfaces. In considering a language suitable for the description of interfaces to databases, we chose to investigate the use of NOODL. The authors believe that the following benefits have been gained from such an approach:-

- An integrated data and interface language provides a much tighter coupling between the data and interface components of a database system. This facilitates the construction of direct manipulation interfaces [44] as shown in the prototype presented.

- Cooper [16], states that the provision of interfaces requires considerable '*intricate low-level programming*' *e.g.* event handling, drawing graphical primitives, managing interface component identifiers, *etc.*. We have presented a language which allows the conceptual design of interfaces in abstract terms.

- NOODL provides a fully object oriented data model. Features such as, polymorphism, extensibility, complex objects and behaviour which have been used to great effect in existing interface programming languages [35], have been shown to be equally effective in a conceptual interface language.

- The provision of a unified model and language results in the necessity to learn only one language for both database and interface description.

- The impedance mismatch existing between data languages and interface languages observed by Paton [36] has been eliminated through the use of a common language for both.

- Given any particular graphical interface to a database, a database system must be able to regenerate the visual layout of the interface. As NOODL is a language for describing database contents, this implies that if it is used to describe an interface, then this interface information may be stored in the database. Storage of the interface's description in the database facilitates this regeneration.

6 Further Work

Further work to be undertaken includes the development of both NOODL as a language for the specification of database interface descriptions and further

experimentation of the interface styles for databases using the prototype software already developed. In addition the prototype will be enhanced to support the behaviour of operations, constraints and triggers on the data. The architecture which fully supports the genericity and configurability of this prototype requires further development for these enhancements.

The presentation of NOODL as an interface description language shows how we suggest the language be used. It does not show exactly how certain conceptual interface features map to their implementation. There are several ways in which this may be done and although we have mapped our example to an implementation we intend to investigate alternative mechanisms before committing to a particular specification.

As NOODL is a general purpose conceptual modelling language we feel that there is enormous potential for the variety of features that may be described in an interface using this language. We plan to investigate the full power of NOODL is this way and evaluate NOODL as an interface description language in comparison to existing purpose-built user interface languages.

In providing an interface to a database it is possible to view the database in many different ways. Possible interfaces to databases include ones based on a schema visualisation, a schema visualisation combined with some instance representation or a representation of all the instances in the databases. In addition to this, different users of the same database may have differing preferences for how they wish to visualise and interact with the data. Therefore, besides developing a more sophisticated or realistic example interface using the prototype, it is planned to experiment with interfaces based on these different views of a database and the ability to switch between them.

It is important that in providing an interface to a database for end-users that the layout of the objects in the interface be ‘meaningful’. We plan to investigate the possibility of defining layouts of interfaces and the automatic generation of the layouts from its definition. This investigation includes the generation of visualisations for the results and construction of database queries.

Addressing these issues requires an inter-disciplinary approach. The results of research on both databases and human-computer interaction require integration in a framework to promote organised mutual exploitation. In satisfying this need a conceptual framework for user-interfaces to databases has been developed [31].

References

1. J. Almarode (1991) Issues in the Design and Implementation of a Schema Designer for an OODBMS, *ECOOP’91*.

2. A.N.S.I. (1985) *American National Standard for Factors Engineering of Visual Display Terminal Workstations - Graphics Kernal System (GKS).* American National Standards Institute.

3. A.N.S.I. (1988) *American National Standard for Information Processing Systems - Programmer's Hierarchical Interactive Grahpics System (PHIGS).* American National Standards Institute.

4. P.J. Barclay & J. Kennedy (1991) Regaining the conceptual level in object oriented data modelling. *BNCOD,* 9, 269-305.

5. P.J. Barclay & J.B. Kennedy (1992) Using a Persistent System to Construct a Customised Interface to an Ecological Database, *1st International Workshop on Interfaces to Database Systems,* 1:14.

6. P.J. Barclay & J.B. Kennedy (1992) Semantic Integrity for Persistent Objects, *Information and Software Technology,* 34:8, 533-541.

7. P.J. Barclay (1993) *Object oriented modelling of complex data with automatic generation of a persistent representation.* Phd Thesis. Edinburgh: Napier University.

8. P.J. Barclay & J.B. Kennedy (1994) A conceptual language for querying object-oriented data, *Proceedings of BNCOD,* 12:13, 187-204.

9. S. Benford & J. Mariani (1994) Populated Information Terrains, *2nd International Workshop on Interfaces to Databases,* 2:9, 159-169.

10. B.K.S. Software (1994) *POET (Version 2.1) - Programmer's & Reference Guide*. B.K.S. Software.

11. Borland (1995) *Borland C++ compiler (version 4.0) - user manual*. Borland International Inc.

12. J. Boyle, J.E. Fothergill & P.M.D. Gray (1993) Design of a 3D user interface to a database, *Database Issues for Data Visualisation Workshop.*

13. J. Boyle, J.E. Fothergill & P.M.D. Gray (1994) Amaze: a three dimensional graphical user interface for an object oriented database, *2nd International Workshop on Interfaces to Databases,* 2:7,117-131.

14. M. Chalmers (1994) Design Perspectives in Visualising Complex Information, *FADIVA Workshop,* 1:4.

15. R. Cooper (1991) Configurable data modelling systems, *Entity-Relationship Conference,* 9,35-52.

16. R. Cooper (1994) Configuring Database Query Languages, *2nd International Workshop on Interfaces to Databases,* 2:1, 1-17.

17. C.J. Date (1987) *A Guide to the SQL Standard*, Addison-Wesley.

18. O. Deux, *et al* (1991) The O2 System, *Communications of the ACM,* 34:10, 34-48.

19. D.S. Dyer (1990) A Dataflow Tookit for Visualisation, *IEEE Computer Graphics and Applications*, 10:4, 60-69.

20. G.P. Ellis, J.E. Finlay, A.S.Pollitt (1994) HIBROWSE, *2nd International Workshop on Interfaces to Databases,* 2:3,45-58.

21. J. Foley, W. Kim, S. Kovacevic, & K. Murray (1989) Defining Interfaces at a High Level of Abstraction, *IEEE Software*, 6:1, 25-32.

22. M. Green (1987) A Survey of Three Dialog Models, *ACM Transactions on Computer Graphics,* 5:3, 244-275.

23. P. Hayes & P. Szekely (1983) Graceful Interaction Through the COUSIN Command Interface, *International Journal of Man-Machine Studies,* 19:3, 285-305.

24. M. Hemmje (1994) LyberWorld - A 3D Graphical User Interface for Fulltext Retrieval, *FADIVA Workshop,* 1:5.

25. S. Hudson & R. King (1988) Semantic Feedback in the Higgens UIMS, *IEEE Transactions on Software Engineering*, 14:8, 1188-1206.

26. R. King & M. Novak (1993) Designing Database Interfaces with Dbface, *ACM Transactions on Information Systems*, 11, 105-132.

27. R. King & M. Novak (1989) FaceKit: A Database Interface Design Toolkit, *Proceedings of VLDB,* 15.

28. G.N.C. Kirby, & A. Dearle (1990) An Adaptive Graphical Browser for Napier88, *Technical Report, University of St.Andrews.*

29. M.O.P. (1994) *ObjectStore - Language Interface Users Guide (Release 3.0),* Object Design Ltd, 249-326.

30. K.J. Mitchell (1994) *Schema visualisation*. MSc Thesis. Edinburgh: Napier University.

31. K.J. Mitchell, J.B. Kennedy, & P.J. Barclay (1995) A Framework for Interfaces to Databases, *Technical Report, Napier University (submitted to VLDB'95).*

32. S. Monk (1994) A Graphical User Interface for Schema Evolution in an Object Oriented Database, *2nd International Workshop on Interfaces to Databases,* 2:9, 171-184.

33. D. Olsen (1989) MIKE: The Menu Interaction Kontrol Environment, *ACM Transactions on Graphics*, 5:4, 318-344.

34. O.S.F. (1989) *OSF/MOTIF - Manual*, Open Software Foundation.

35. O.W.L. (1994) *ObjectWindows (Version 2.0) for C++ - Programmer's Guide.* Borland International Inc.

36. N. Paton, G. al-Qaimari & K. Doan (1994) On Interface Objects In Object-Oriented Databases, *BNCOD,* 12:11, 153-169.

37. C.A. Pickover (1991) *Visualisation, Computers and the Imagination*, Alan Sutton Publishing.

38. M.H. Rapley (1994) *Three dimensional interface for an object oriented database*. MSc Thesis. Edinburgh: Napier University.

39. M.H. Rapley (1994) Three dimensional interface for an object oriented database, *2nd International Workshop on Interfaces to Databases,* 2:8, 133-158.

40. RenderWare (1994) The RenderWare API Reference (Version 1.4), *Criterion Software Ltd.*

41. G. Robertson, S. Card & J.Mackinlay (1993) Information Visualisation Using 3D Interactive Animation, *Communications of the ACM* 36, 57-71.

42. G. Santucci & F. Palmisano (1994) A Dynamic Form Based Visualiser for Semantic Query Languages, *2nd International Workshop on Interfaces to Databases,* 2:14, 235-250.

43. D.W. Shipman (1980) The Functional Data Model and the Data Language DAPLEX, *ACM Transactions on Database Systems*, 6:1.

44. B. Shneiderman (1983) Direct Manipulation: a Step Beyond Programming Languages, *IEEE Computer*, 16, 57-69.

45. F. Steinfath, K. Bohm & B. Lange (1994) Evaluation of Complex Information Processing Systems in 3D-Space, *FADIVA Workshop*, 1:2.

46. M. Stonebraker & J. Kalash (1982) TIMBER: A Sophisticated Relational Browser, *Proceedings of VLDB*, 8.

47. B. Stroustrup (1982) *The C++ programming language*. Addison-Wesley.

48. C. Upson, T. Faulhaber, D. Kamlins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz & A. van Dam (1989) The Application Visualisation System: A Computational Environment for Scientific Visualisation, *IEEE Computer Graphics and Applications*, 9:4, 30-42.

49. K.Y. Whang et al (1987) Office-by-Example: An Intergrated Office System and Database Manager, *ACM Transactions on Office Information Systems*, 5:4, 393-427.

50. P. Windsor & G. Storrs (1993) Practical User Interface Design Notation, *Interacting with Computers*, 5:4, 423-438.

51. R.H. Wolfe, M. Needels, T. Arias & J.D. Joannopoulos (1992) Visual revelations from Silicon Ab Initio Calculations, *IEEE Computer Graphics and Applications*, 12:4.

52. H.K.T Wong & I. Kuo (1982) GUIDE: Graphical User Interface for Database Exploration. *Proceedings of VLDB*, 8, 22-32.

53. S.B. Yao, A.R. Hevner, Z. Shi, & D. Luo (1984) FORMANAGER : An office forms management system, *ACM Transactions on Office Information Systems*, 2:3, 235-262.

54. K.Yap & G.Walker (1992) The Object User Interface to the Banksia Geographical Information System, *1st International Workshop on Interfaces to Databases*, 1:13.

55. M.M. Zloof (1975) Query by Example, *Proceedings of the National Computer Conference*, 431-437.

56. M.M Zloof (1982) Office-by-example: A business language that unifies data and word processing and electronic mail. *IBM Systems Journal*, 21:3, 272-304.

Appendix

Interface Class Definition

```
class Interface_Class
properties
  shape       : Shape ;;
  position    : Position ;;
  size        : Size ;;
  orientation : Orientation ;;
  colour      : Colour ;;
  extent      : Extent
operations
  move   : Move ;;
  select : Select
```

The above class definition shows the initial set of interface properties and operations considered. These are available to all interface classes in a NOODL database interface description. A brief description of each is given below.

Interface Class Properties and Operations

- **Shape** -
```
class Shape
property
  name : Text ;;
```

The shape of an object as it appears in the interface. A simple text string is used to refer to a shape. Alternatives include 2/3D objects, dialog forms, or simple textual descriptions. In the case of a dialog forms interface the Shape representation may include properties itself referring to fields in the form, *e.g.* shape.field.name="Fred".

- **Position** -
```
class Vector_3d
properties
  x : Real ;; y : Real ;; z : Real

class Position
ISA Vector_3d
operations
  translate Real x, Real y, Real z ;;
  is_inside Extent e : Boolean ;;
  put_inside Extent e
constraint
  PLC is -1.0 >= self.x >= 1.0 and
         -1.0 >= self.y >= 1.0 and
         -1.0 >= self.z >= 1.0
```

This represents the position of the object relative to the interface space. This definition is for a position in a 3D Cartesian coordinate interface space with real numbers holding the x, y and z coordinates. Positions in other interface spaces follow

a similar definition. The *position limit constraint* (PLC) defines the bounds of coordinate values permitted. *Is_inside* tests whether or not the position is within the bounds of an extent and *Put_inside* sets the position to a point within an extent.

- **Size** -
```
class Size
properties
  width : Real ;; height : Real ;; depth : Real
operation
  scale Real w, Real h, Real d
```

The size of the object relative to interface space. Again for the purposes of our example, this is particular to a 3D interface space. The size of the object made be scaled through its *scale* operation.

- **Orientation** -
```
class Orientation
  ISA Vector_3d
  operations
    rotate Real d ;;
    rotate Real x, Real y, Real z, Real d
```

The orientation of the object relative to interface space. The object may be rotated through *d* degrees, about either its own orientation or a specified axis of rotation.

- **Colour** -
```
class Colour
properties
  name : Text ;;
  red  : Real ;; green : Real ;; blue : Real
constraint
  CLC is 0.0 >= self.red    >= 1.0 and
         0.0 >= self.green  >= 1.0 and
         0.0 >= self.blue   >= 1.0
```

The colour of the object specified by red, green, blue intensity components. As with position vectors, a set of constraints defines the limits for these values. This may be specified by for example red, green, blue intensity components. or a textual colour name, e.g. “dark blue”, or through an alternative colour model, e.g. the hue, saturation, value (HSV) model. The choice of colour representation is entirely dependent on the modeller’s preferred colour model.

- **Extent** -
```
class Extent
properties
  lower_back_left : Position ;;
  upper_front_right : Position
operation
  holds Position p : Boolean
```

The bounding box which completely encloses the object, consisting of properties to define the lower back left and upper front right limits of the extent. *Holds* returns a boolean value depending on whether or not a given position lies within the extent.

- **Move** -
```
move : Move
class Move
properties
   event : Boolean ;;
   position : Position
```

This operation describes a response to the user moving the object in the interface. The position property represents the point to which the interface object has been moved. It is intended that a system interpreting this language is able to handle the operating system's events in order to recognise that the user has moved the object.

- **Select** -
```
select : Select
class Select
property
   event : Boolean
```

Select describes a response to the user selecting the object in the interface.