A Framework for User-Interfaces to Databases

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
phone: +44-0131-455-5340 ; fax: +44-0131-455-5394

Abstract

A framework for user-interfaces to databases (IDSs) is proposed which draws from existing research on human computer interaction (HCI) and database systems. The framework is described in terms of a classification of the characteristic components of an IDS. These components, when progressively refined, may be mapped to a conceptual object-oriented language for the precise specification of the IDS. A prototype system is presented, showing the potential for automated mapping of a language specification to a fully functional implementation. As well as providing general support to any database interface developer, we believe that the framework will prove useful for researching a number of IDS issues.

Keywords: User-Interfaces to Databases, Human-Computer Interaction (HCI), Conceptual Modelling, Direct Manipulation Interfaces.

1. Introduction

Addressing the problem of designing interfaces to databases requires an inter-disciplinary approach. The results of research on both databases [44] and human-computer interaction (HCI) [23] require integration in a framework to promote organised mutual exploitation [20]. Therefore, this work takes existing models of databases and user-interfaces and fuses them in a conceptual framework for user-interfaces to databases (IDSs).

Although, models exist for user-interfaces in general and are applicable to IDSs, there is an imperative for models which address the particular needs of databases. Numerous recent workshops show that database researchers are concentrating efforts on developing interfaces from a database perspective [2, 3, 24, 26, 28]. In common with these efforts, this work is principally concerned with database issues for interfaces.

The primary organisational step for the creation of this framework is to determine the characteristic components of IDSs. A general framework requires detailed specification for practical application. Recognising this, our framework relates both abstractly to the general features of an IDS and in depth to its atomic components.

It is shown how a detailed classification of components under this framework may be mapped to a conceptual language which embodies the relationships and dependencies among the components of an IDS. The organisation of components in a well-defined framework represented in a concise conceptual language provides a sound foundation for further research on interfaces to databases.

With this framework, existing interfaces can be analysed and new interfaces can be designed. The detail provided in resulting language descriptions is sufficient to define a functional IDS implementation. A prototype is described which demonstrates the automatic generation of database interfaces from language definitions.

The following section describes the necessary background to provide a context for this work. Section 3 presents a framework defining the characteristic components of IDSs. Section 4 details the mapping from this framework to the conceptual language. Section 5 describes the prototype and demonstrates the practical realisation of this framework to innovative interface styles. Finally, conclusions and some further work are discussed.

2. Background

In applying IDS and HCI models with their own particular emphases, an overall structure is beneficial to localise their placement and therefore, gain a better appreciation of the scope of their application. In attempting to solve this problem of context, interaction frameworks may be of use.

The execution-evaluation cycle, by Norman [33] considers the interactive process as being a cyclic sequence of seven stages, from execution (1:establishing a goal, 2:forming an intention, 3:specifying a sequence of actions, and 4:executing the action), to evaluation (1:perceiving the system state, 2:interpreting the system state and 3:evaluating the *perceived* system state with respect to the goals and intentions of the user). Although, this influential framework provides a clear distinction of the stages in an interactive sequence, it does not address the details of the system, with respect to the *actual* system's state.

In extension to Norman's framework, Abowd and Beale's [1] interaction framework identifies *system* and *user* components which communicate via the *input* and *output* components of an interface (figure 1). This communication follows a similar cyclic sequence of steps from the user's *articulation* of a task, through the system's *performance* and *presentation* of the task, to the user's *observation* of this task's results, upon which the user can formulate further tasks.

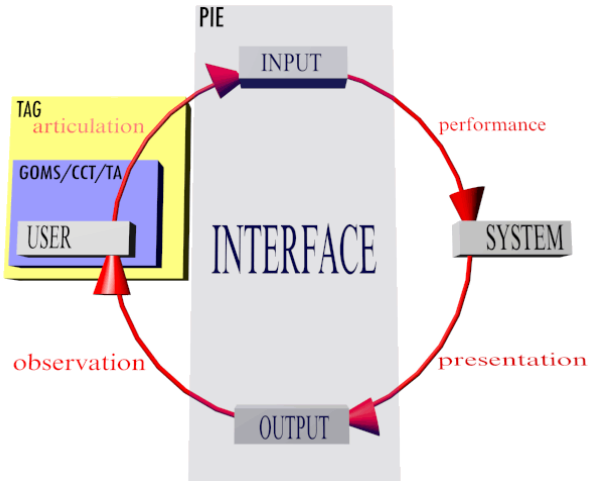

Fig 1 Abowd and Beale's interaction framework with the context of existing interaction models shown according to their particular emphasis.

Models in the field of HCI have covered many aspects of the user-interface towards the goal of satisfying various usability principles, such as, learnability, flexibility, and robustness [23]. Such models emphasise particular components of user-interfaces, thus determining their scope and how they are applied. Figure 1 shows the particular emphasis of some existing HCI models in the context of Abowd and Beale's interaction framework. The goals, operators, methods and selection (GOMS) model [17], cognitive complexity theory (CCT)[14] and task analysis (TA)[21] are all applied specifically to modelling users of interactive systems. Task action grammars (TAG)[37] are used to model a user's articulation of a task and the program, input and effect (PIE) model [22] emphasises the input and output of an interactive system.

If usability principles are to be applied in a database domain, models which emphasise data oriented components must be addressed, in particular, the role of data visualisation.

Recent work on interfaces from the database community has recognised the lack of data oriented models of visualisation [27] and have proposed a formalism for the process of visualising data in a system. This model makes particular concern for the use of metaphor in the visual representation of the data model constructs. Further, Catarci et al [18] have further specified this approach with consideration of the application domain and the transparency of metaphor used.
The framework presented in this paper draws from user, interaction, visualisation and data models where appropriate, but is primarily influenced by Abowd and Beale's [1] general interaction framework.

3. A Framework of Interfaces to Databases

Figure 2 shows the proposed framework defining the major characteristic components of an IDS adapted from Abowd and Beale's original. In addition, the lower boxes extend this framework by defining specialised features of each component, which are identifiable when applying the interaction framework specifically to databases.

The revised framework contains *database*, *interaction*, *visualisation* and *user* components which are derived from Abowd and Beale's *system*, *input*, *output* and *user* components, respectively.

Input is replaced with *interaction*, because in many modern direct manipulation interfaces the user's articulation of a task is involved with the simultaneous input *and* output of a system. For example, the classical drag-and-drop operation is performed by using a mouse (input) to move an icon (output). In this way, the framework regards the articulation of a task as a communative dialogue in conveying the intention of the user, such as, querying, browsing or updating. Importantly, this permits a more directed approach to realising a database user's articulation in this framework. *System* is replaced with d*atabase*, allowing the framework to deal specifically with the interaction between the user and the elements of the database. O*utput* is replaced with *visualisation*, which concentrates the framework on presenting the elements of the database. This permits the separation of the concerns of data visualisation as a distinct component from the interaction component. Clearly, the issues currently (and foreseeably) pertinent to IDSs are explicitly represented within the components of this framework.

The specialised features of each component have a bearing on the features of other components. Typically, the features of the user and database will determine a particular choice of interaction and visualisation component. Each component is detailed below with further explanation of their features.

An important property of this framework is that it is applicable to both general abstraction and concrete specification of the

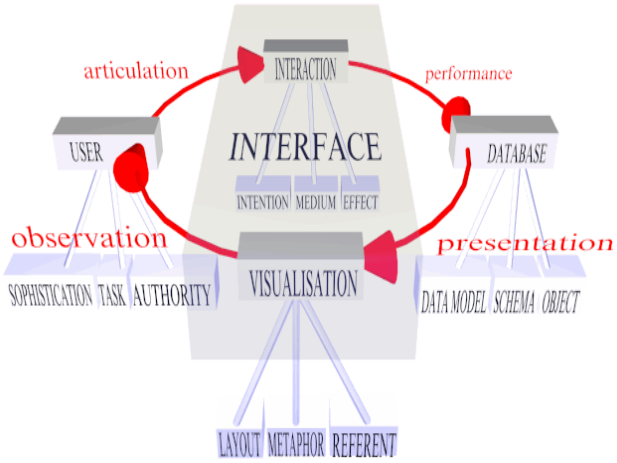

Fig 2 A framework for the characteristic components of user-interfaces to databases

particular IDS in question. For example, the intention of an interaction component may be conceptualised at a general level to provide data retrieval by means of interactively specifying a query, or at a detailed level to specify the intention of an atomic interaction component, such as pushing an 'ok' button to confirm the deletion of an object. In this way, each of the four major components may be sub-divided, in a step-wise refinement fashion, to specify aspects of an IDS at successive levels of detail. A direct mapping to the conceptual language is achieved when the atomic sub-components of the database, user, interaction and visualisation components are specified.

3.1 Database

Depending on the purpose of the IDS, the database component may relate to a representation of the data model, the schema described under the data model, the objects instantiated from classes in the schema or some combination of these.

Data Model. Data models define the types of data, constraints and operations that can legally be represented in a database. The set of data modelling constructs present in any particular data model determine the flexibility, behaviour, expressiveness and level of integrity possible in the data described. For certain purposes it may be appropriate to visualise the data model supported by the database, eg. to understand precisely the semantics of the schema [27].

Schema. A database schema precisely defines the structure and meaning of the data in a database, in terms of the data model. Typically, the design of database schemata entails the definition of the classes of objects to be stored and the relationships among them. The complexity and structures inherent in a schema will have an impact on the design of an interface. Providing an interface based on the schema may be advantageous for new users to the database to use as a 'map' of the database. In addition, it may be useful to a database administrator to visualise and update the schema. Thus, the stability of such schemata is also an item for consideration.

Objects. Objects are the basic units of information in a database and it is important to provide appropriate interfaces to this information. Elementary objects might hold one atomic item of information, whereas complicated objects typically have a variety of properties, operations, constraints and related objects. In designing an interface depicting the contents of the database a variety of issues will affect the appearance of an object. For example, it may not be practical to visualise a huge number of complicated objects given a limited amount of screen space. Alternatively, novel techniques may be applied to accommodate such large numbers [39]. The genericity of an object is also of concern, eg. an object which represents something tangible in the real world may yield a more concrete visual metaphor than one which represents a generic abstract concept. An object which changes only occasionally may not require constant visible representation in the interface.

These features combine to describe the database component, which may determine the purpose of the interface and affect the choice of features of user, interaction and visualisation components of the framework. For example, the WINONA [39] interface which combines schema and object information is suitable for browsing, querying, data entry and manipulation, whereas the NIOME [29] and OPOSSUM [27] interface which concerns only the schema information is not designed for querying data.

As a result, an IDS may be specifically involved with the visualisation of the database schema or alternatively, an IDS might require data entry for the objects of the database. Indeed, it may be useful to manipulate aspects of the data model through the interface, although this is less common.

3.2 User

Although user modelling is possible through a variety of sophisticated techniques, such as, TAGs, CCT, GOMS, etc., their specific relevance to database issues is limited. Therefore, an approach which facilitates such analyses, but remains particular to the requirements of a database user, was chosen. The major characteristic features of a user affecting the IDS are his/her sophistication, task, and authority.

Sophistication. The user's sophistication is defined in terms of his/her knowledge of the components of the framework, ie. the objects, schema or data model of the database, the style of interaction and the meaning of the visualisation. Clearly, the perceived sophistication of the user by the interface designer will affect the choice of data to present to the user. It is doubtful that a user with no database design experience will understand the meaning of a schema diagram using abstract symbols, but s/he may be comfortable with concrete visual representations of the objects in the database, eg. representing a person object by his/her photograph. A user's experience with a particular style of interaction may guide the designer towards providing that style. Conversely, a chosen style of interaction may preclude a group of potential users who are not proficient in that style.

Task. The general tasks which users must perform are pertinent to interface design. A user who has to perform frequent, verbose tasks would choose a more efficient, minimal interface in preference to a more deliberate, elaborative interface where many actions require confirmation. A wide range of tasks expected from a given user requires frequent reminders of the meaning of the interaction operations available. Individual tasks may be modelled using TAGs, CCT, GOMS, etc. to provide a detailed analysis of the functionality required by the interface, eg. the task of querying a database may be broken down into smaller component tasks, such as, defining the query, executing, and browsing the result.

Authority. A user's authority is defined by the level of access to data provided by the interface, eg. the permission to read, write or update this information. An end-user may only merit

a limited view of the data, whereas administrators frequently require a full view permitting updates of the schema as well as the objects. A user's view in database terms is a desirable feature of secure database systems [8].

These features combine to describe the user component, any of which may affect the choice of features of database, interaction and visualisation components. Authority is orthogonal to sophistication, eg. a managing director may be granted authority over the entire contents of the database, but still have a low degree of sophistication in the use of a particular interface.

3.3 Interface

In this framework, an interface is composed of visualisation and interaction components. A visualisation component is preferred to the more general output component in deference to the extensive body of research on data [19] and database [12, 15, 16, 30, 39, 43, 45] visualisation.

3.3.1 Visualisation

A visualisation component refers to the output of graphical information particular to the current purpose of the database application. The user's observation of and the database's presentation of this visualisation component is determined by the chosen layout and metaphor.

Multi-media databases storing sounds may be classified where the visualisation metaphor is interpreted as an audible noise. In such cases, the word visualisation can be interpreted broadly as a multi-modal presentation component.

Referent. The purpose of a visualisation is simply to communicate to the user some component of the database user interface, ie. the referent which the current visualisation is representing. Typically, this concerns the visualisation of the database's contents, from concrete visualisation of database objects [30] to abstract visualisation of schemata [29] or queries [45]. At a lower level of detail the visualisation is required to represent nested interaction components, eg. a menu or window. Indeed, it is sometimes useful to visualise user components representing other users [12].

Metaphor. A metaphor defines the symbol used to represent the component being visualised. A metaphor may range from the direct representation of the component, to an abstract symbol in some way related to the component in question. Formal definition of the mapping from the referent to the metaphor may be possible [18, 27].

Layout. A layout is defined as the position of interface components relative to other components in a common environment. This is particularly important in communicating the structure of the data to the user. Types of visual layouts include linear, circular [39], form, grid, hierarchical [43], scatter plot and graph [15] structures. Clearly, the structure of the schema in the database component will frequently determine the designer's choice of visual layout. When the referent of visualisation is to represent an interaction component issues may be drawn from their functionality, sequence, and frequency [23], eg. a frequently repeated interaction operation should be located close to hand and not obscured.

3.3.2 Interaction

An interaction component refers to the input of information articulating the user's intention to the interface. This intention is communicated through the interaction component's medium and performed to achieve a specified effect.

Intention. A major feature of an interaction component is its intention, ie. the specific interface action which satisfies some goal. This concerns both the subject of the interaction component and its practical function. The subject will be some selection of database, user and interface components and its use may concern the entry, manipulation, browsing, or querying of data. For example, a form may concern a new object in the database and its function may be for entering the information required for this object.

Medium. When communicating a user's intention, the medium through which this is achieved is important in the design of an interface. This medium concerns both physical and logical aspects of the interface; physical aspects being the mouse or keyboard and logical aspects being buttons, scroll bars, menus, text fields, dialog boxes and objects. The logical aspects of an interaction component's medium is directly associated with a visual component's referent in the interface.

Effect. The effect of an interaction component can change any aspect of any component in the IDS framework. For example, the effect of a button whose intention is to delete an object in the database will (obviously) remove an object from the database.

This model of interaction proposed is similar in concept to the PIE model [22], where the effect of a particular input may change the internal state of the system and/or the visible system's state. The removal of an object from the database component may be displayed by a change in the visualisation component, eg. a decrease object counter or the disappearance of a shape previously used to represent the object. Whereas the PIE model is a *black-box* model with concern for only external behaviour, our model is necessarily concerned with the specification of internal changes in state in order to map the identified components of an IDS to a database implementation. However, at an abstract level the predictability and observability of a database's state can be formally analysed with respect to this model.

3.3.3 Style and Complexity

A combination of visualisation and interaction components determines the style and complexity of the database user

interface. Both these features are themselves abstracted from the above features of visualisation and interaction components.

Style. The style of an interface to a database can be classified in abstracts terms as textual, forms-based, graph-based, three dimensional, etc. In forms-based interfaces the form can act as a medium for both input and output (in the terms of this framework, interaction and visualisation). This is an example of the style being determined by a combination of visualisation and interaction components. Within this framework, the style of an interface component may be characterised by the metaphor and layout of its visualisation component and the medium used by its interaction component.

Complexity. The number, size, and frequency of recurrence of visualisation and interaction components combine to define the complexity of the interface component. For example, a large number of different symbols densely arranged will be confusing to a user. However, visualisations of large numbers of objects can be simplified by representing objects in conjunction with schema information [39]. The frequency of recurrence concerns how often and in what sequence in time interface components are accessed, eg. if a form is expected to be frequently accessed, then it should provide a means for rapid processing by the user. Alternatively, if a form is rarely used, then its features may require explanation. In general, the complexity of an interface may be measured quantitatively using Olsen's [34] interface quality metrics and applied as an issue in the design of IDSs in this framework.

4. Mapping an IDS Classification to a Conceptual Language

For this purpose, we have chosen the Napier Object Oriented Data Language (NOODL) for IDS specification. NOODL is a combined data definition language, data manipulation language and query language for object oriented data. It is based on the modelling approach described in [4]; it has been used to model [5] and to support the implementation [7] of novel database applications, and also for the investigation of specific modelling issues such as declarative integrity constraints and activeness [6] and the incorporation of views [8] in object oriented data models. It also includes a query language [10].

A NOODL schema contains a list of class definitions, which show the name and ancestors of each class. A class definition also includes the names, sorts, and, optionally, definitions of the properties of each class. A class definition may also contain operations, constraints, and triggers. NOODL will be further introduced later through examples; full details may be found in [9].

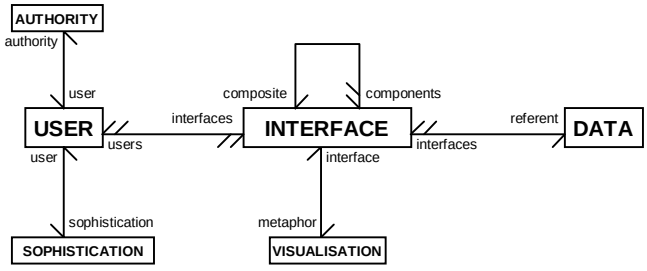

Fig 3 A NOODL meta-model of the IDS framework

In Abowd and Beale's interaction framework, each of the four components may be defined using its own language and it is suggested that the ease of translation between these languages determines the efficiency of the interface. However, given a complete detailed classification of an IDS according to the framework proposed here, each atomic component has a direct mapping to a construct in NOODL. Therefore, one integrated language is used for all components. This achieves a uniform, natural way of describing databases and their interfaces. The use of NOODL for specifying the framework provides the following advantages:

- as a data modelling language it is immediately applicable to data definition and manipulation
- it eliminates a potential impedance mismatch, which arises if different languages are used for each IDS component
- it permits the application of object oriented concepts to the modelling of interface and user components
- interface and user components may be stored and retrieved along side database components in a persistent system.
- as a conceptual language it promotes communication between users and IDS designers

The resulting NOODL schema defines the relationships and dependencies among the database, user, interaction and visualisation components embodying any IDS.

Figure 3 depicts the classes and properties of a meta-model of the framework using NOODL constructs. In mapping the framework to NOODL, certain components are depicted as NOODL classes, some as properties, while other are modelled using operations, constraints or triggers, examples of these are shown in sections 4.1-4.3. An example schema based on the above template is given in the appendix.

In common with Rumbaugh's user interface modelling approach [41] multiple interface objects are associated with each data object. This permits database updates to be broadcast to each relevant interface object. Thus realising the facility for multiple-coordinated views [42]. Much work in the field of HCI reflects this conceptual organisation. Conceptual architectures for user interface management systems (UIMS) typically involve the identification of the system (data) in separation from input and output (interface) components. This is evident in the archetypal Seeheim workshop model with application interface, dialogue control and presentation components; Smalltalk's model, view, and controller (MVC) paradigm; and the presentation, abstraction and control model (PAC)[35].

Of particular concern to IDSs is the presentation of large amounts of data. With VDU real-estate at a premium, the layout of interface components is increasingly important. The above

model caters for layout organisation through the use of interface object composition. Each interface object may contain a set of other interface objects, which may be arranged algorithmically.

In addition, consideration for IDS users are explicit in this model. This enables applications to provide user specific interfaces, based on the details of the particular user model thereby realising the role of the database user in the interface. The following describes the mapping of each feature of the components in the framework to a set of constructs in NOODL.

4.1 Mapping Database Component Features to NOODL

In an IDS there must be some mechanism for presenting the information stored within the database to the user, ie. it must be possible to present each of the features identified in the database component. This is achieved by linking each database component to an appropriate interface component. In NOODL, this corresponds to a pair of obverted properties in both *data* and *interface* classes [30] (using the *ref* keword). A data class represents the features of a database component. Likewise, an interface class represents the features of an interface component (described later). In its simplest form, a data class is equivalent to a class in the database's schema, where the interface property defines the link which binds this data class to an interface class, eg. representing a museum artefact in a NOODL database would be defined as below,

```
class Artefact
properties
    interface : Artefact_Interface ref referent
    name : Text
    catalog_id : Number
```

As mentioned in section 3.1, it is sometimes necessary to visualise the database schema. In this case it is not the individual artefact that requires presentation, but the general concept of an artefact. This requires a meta-representation of the database component's schema, eg. representing an artefact's class in NOODL would give,

```
class Artefact_Class
properties
    interface : Artefact_Class_Interface ref referent
    class_name : Text
    property_details : #Text
```

Again, an interface property defines the link to an interface class. If the IDS is providing a manipulation of the features of the database component's data model, the information defining concepts such as, class, property, operation, constraint, etc., must be specified. This requires a meta-representation of the database component's data model, eg. representing the data model's concept of a class gives,

```
class Class
property
    interface : Class_Interface ref referent
```

4.2 Mapping User Component Features to NOODL

An individual user is mapped to an object in the NOODL data model. A group of users with common features may be defined in a user class (Figure 4). The *accessor* property defines the set of interface objects accessible to the user.

```
class User
properties
    accessors : #Interface ref users ;
    sophistication : Sophistication ref user ;
    authority : Authority ref user
operations
    task1 is self.task1a, self.task1b ;
    task1a is self.interface.intention1 ;
    task1b is self.interface.intention2

class Sophistication
property
    user : User ref sophistication

class Authority
property
    user : User ref authority
```

Fig 4 A user class template with access to interface class

The sophistication and authority of a user are specified by obverted properties. These are abstract classes, which may be derived for a particular user model. Related to this is the task of the user, ie. the set of possible tasks a user can carry out are limited by the extent of his/her view of the IDS. If a user's task is necessarily represented explicitly (eg. for modelling the user through hierarchical task analysis [23]), a structured sequence of intentions may be defined as shown in the operations of the above class.

4.3 Mapping Interface Component Features to NOODL

Modelling of interface components based on an object oriented data model is well established, eg. the user interface design environment (UIDE) [25]. Recent work has shown that it is possible to use NOODL for this purpose [30]. As in section 4.1, a NOODL interface description can associate each class in the database's schema with an interface class, which defines the appearance and interface actions related to the data. A description of how the features of a NOODL interface class can be derived from the features of an interface component in the framework follows.

class Interface
properties
referent : Data **ref** interface
metaphor : Visualisation **ref** interface
users : #User **ref** accessors
composite : Interface **ref** components
components : #Interface **ref** composite
operation
intention **is** *medium*
trigger
intention => *effect*

class Visualisation
properties
interface : Interface **ref** metaphor

Fig 5 NOODL interface class templates

The referent of a visualisation component corresponds to the object referenced by the obverted *referent* property (figure 5). This property may specify either a data, user or interface class, allowing a visual representation of a database, user or interface component. In a direct manipulation interface [42], interface classes will mainly reference data classes. The visualisation metaphor is defined by the *Visualisation* class referenced through the metaphor property. This separation of visualisation classes from interface classes allows flexible linkage for alternative visualisations.

The interaction component's intention is defined by the operation, intention. The interaction *medium* corresponds to the sequence of events listed in intention definition, eg.

self. metaphor.select

The interaction *effect* is defined by the interface class' trigger action, eg.

self. referent.update

An example NOODL schema, based on the mappings described above, for the interface shown in figure 7 is detailed in the Appendix.

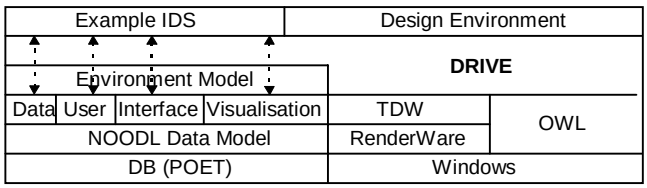

Fig 6 Layered IDS architecture generated with the DRIVE UIDE

5. Prototype Interface to a Database

Given such schema specifications a prototype interface may be generated automatically by applying the above mapping systematically to an implementation. The realised prototype interface may be used in empirical studies of its effectiveness or in practical applications. This has been partially fulfilled in the DRIVE (database representation independent virtual environment) user interface development environment (UIDE). Although it is beyond the scope of this paper to detail the process of automatically mapping a NOODL schema to a working interface, the architecture of the resulting interface is shown in figure 6; full details may be found in [31].

It has been implemented on a PC-platform under Microsoft Windows in C++, using the Borland's ObjectWindows Library (OWL)[35] and Criterion Software's RenderWare [40] graphics dynamic link library for the 3D representation of visualisation objects. A NOODL data model layer (similar to ObjectStore's Meta Object Protocol [32]) implemented with the POET [13] persistent C++ extension provides a means of dynamically creating persistent database interface schemata together with their data.

Data, user, interface and visualisation classes are linked and maintained by the environment model. This model contains a scheme for handling interaction events and mapping them to the appropriate interface object. A visualisation may be derived in conjunction with existing interface widgets, such as OWL windows, and advanced interface widgets, such as the 3D widget set's (TDW)[16] access widget.

Figure 7 shows a display from the prototype IDS specified in the appendix within the DRIVE UIDE. The display shows various views of the museum which may be altered interactively with the mouse in *designer* mode. The dialog box in the upper middle of the display shows the list of objects of an artefact's data class. The dialog box in front of this one displays the object's information together with a set of buttons intended for editing its properties, operations, constraints, triggers or associated interface class.

6. Conclusion

This paper has presented a specialised framework for interfaces to databases. The definition of the components of an IDS within this framework provides a means of mapping to a concise conceptual language (NOODL). In adapting Abowd and Beale's general interaction framework [1] to database interaction, we have identified the major features and components of an interface with particular relevance to database issues. This promotes advances in IDS research whilst remaining in the context of existing research in the field of HCI.

Given an IDS specification in NOODL, we have demonstrated the potential for its realisation using an interface prototyping tool (DRIVE). This results in a fully functional, practical interface to an object oriented database. Although this framework is

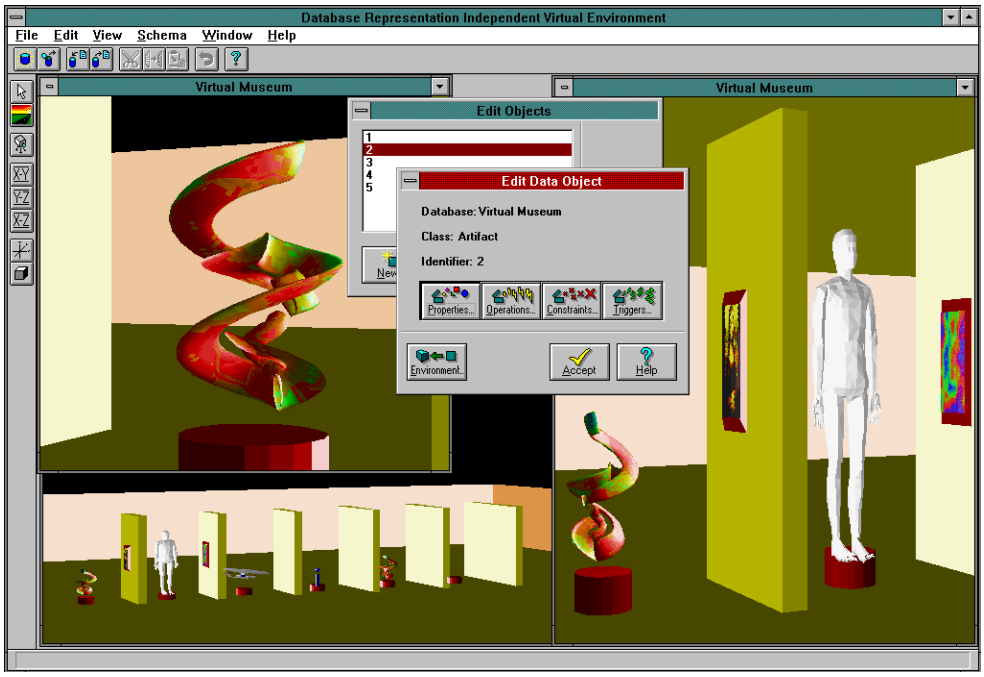

Fig 7 A prototype interface to a museum database created within the DRIVE UIDE

presented for an object oriented database, it is broadly applicable to interfaces to databases supporting any data model.

The use of a framework for interfaces to databases supports the investigation into the salient features of systems providing interaction with data. Issues arising in such interactions are clearly identified within this framework. Comparisons between IDSs using this framework can be made with the potential for highlighting the applicability of particular interface styles. This promises the expression of well defined solutions and rules for the design of future IDSs.

7. Further Work

Both interface style and complexity are abstractly defined by a combination of all the above mappings. However, it may possible in the future to specify an interface style and generate the appropriate interface classes accordingly; particularly in the case of environment classes.

The use of meta modelling to represent the features of a database component enables a possible mapping to the language. However, future extensions to this language with specific meta modelling constructs may provide a more direct mapping and therefore further facilitate the visual representation a database's data model and schema components.

The range of database components accessible to a given class of user may be specified by a database *view*. Barclay and Kennedy [8] have presented a scheme for modelling a view as an object, defined by a NOODL view class. Each user class may define a view in a similar manner. This technique may be used to determine the presence of related interface classes when the user is using the IDS, defining his/her authority.

A formal definition of the mappings from the framework to the NOODL and from NOODL to an IDS implementation would provide a rigorous definition of this work. Theoretical rules and limitations regarding this framework may be identified

8. References

1. G.D. Abowd & R. Beale (1991) Users, systems and interfaces: A unifying framework for interaction, *HCI'91: People and Computers*, 4, 73-87.

2. ASV (1992) *International State-of-the-Art Conference on Animation and Scientific Visualisation*, Hursley Park, UK.

3. AVI (1994) *AVI'94, Workshop on Advanced User Interfaces*, Bari, Italy.

4. P.J. Barclay & J. Kennedy (1991) Regaining the conceptual level in object oriented data modelling. In: *Proceedings of BNCOD* (Jackson and Robinson, eds). Wolverhampton: Butterworths. 9, 269-305.

5. P.J. Barclay & J.B. Kennedy (1992) Modelling Ecological Data. In: *Proceedings of International Working Conference on Scientific and Statistical Database Management*. Ascona. 6, 77-93.

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, C.M. Fraser & 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.

8. P.J. Barclay & J.B. Kennedy (1993) Viewing Objects, In: *Proceedings of BNCOD*, 11, 93-110.

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

10. P.J. Barclay & J.B. Kennedy (1994) A conceptual language for querying object-oriented data, *British National Conference on Databases*, 12:13, 187-204.

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

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

14. S. Bovair, D.E. Kieras & P.G. Polson (1990) The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human-Computer Interaction*, 5:1, 1-48.

15. 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.

16. J. Boyle & K. Mitchell (1995) Embedding three dimensional graphics inside a user interface development framework, *Technical Report submitted for publication.* Robert-Gordon University, Aberdeen.

17. S.K. Card, T.P. Moran & A. Newell (1983) *The Psychology of Human Computer Interaction.* Lawrence Erlbaum Associates.

18. T. Catarci, M.F. Costabile, A Massari, & G. Santucci (1994) Database Interaction: 3D or not 3D, *Proceedings of FADIVA Workshop*, 1,8.

19. B.M. Collins (1992) Data Visualisation - Has it all been seen before? *International State-of-the-Art Conference on Animation and Scientific Visualisation*, 1-33.

20. R. Cooper (1993) The interaction between DBMS and User Interface Research, editorial of, *Interfaces to Database Systems 1992*, 1-5.

21. D. Diaper *ed* (1989) *Task Analysis for Human-Computer Interaction*, Ellis Horwood.

22. A.J. Dix (1991) *Formal Methods for Interactive Systems*, Academic Press.

23. A.J. Dix, J. Finlay, G.D. Abowd & R. Beale (1993) *Human-Computer Interaction*. Prentice-Hall.

24. FADIVA (1994) *1st FADIVA Workshop*, Seeheim, Germany.

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

26. G. Grinstein *ed* (1993) *Workshop on Database Issues for Visualisation*, San Jose, California.

27. E.M. Haber, Y.E. Ioannidis & M. Livny (1994) Foundations of Visual Metaphors for Schema Display, *Journal of Intelligent Information Systems*, 3.

28. IDS (1994) *2nd International Workshop on User-Interfaces to Databases*, Ambleside, UK.

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

30. K.J. Mitchell, J.B. Kennedy & P.J. Barclay (1995) Using a Conceptual Language to Describe a Database and its Interface, *British National Conference on Databases*, 13:7, 101-119.

31. K.J. Mitchell & J.B. Kennedy (1996) DRIVE: An Environment for the Organised Construction of User-Interfaces to Databases, *Technical Report, submitted for publication*. Napier University, Edinburgh.

32. M.O.P. (1994) ObjectStore : Meta Object Protocol. Object Design Ltd.

33. D.A. Norman (1988) *The Psychology of Everyday Things*. Basic Books.

34. A.M. Olson (1992) Object-oriented Analysis of Visual Computer-Human Interfaces, *Journal of Visual Languages and Computing*, 3, 399-414.

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

36. N.W. Paton, R.L. Cooper, D. England, G. al-Qaimari & A.C. Kilgour (1994) Integrated architectures for database interface development, *IEE Proceedings of Computers and Digital Technology*, 141:2, 73-78.

37. S.J. Payne & T.R.G. Green (1986) Task-action grammars: a model of mental representation of task languages. *Human-Computer Interaction*, 2:2, 93-133.

38. G. Pfaff & P.J.W. ten Hagen *eds.* (1985) *Seeheim Workshop on User Interface Management Systems*.

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) *RenderWare API : Reference Guide*. Criterion Software.

41 J.Rumbaugh (1995) Modelling models and viewing views: A look at the model-view-controller framework, *Journal of Object Oriented Programming*, , 14-22.

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

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

44. S.B. Zdonik & D. Maier *eds.* (1989) *Readings in Object-Oriented Database Systems*, Morgan-Kaufmann.

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

9. Appendix - Example IDS Specification

This schema specifies the prototype IDS shown under the DRIVE UIDE in figure 7. The specification concerns a museum's database which is interacted with through a desktop virtual reality user interface. The data of the database is specified by the *Artefact* class, which holds its name, description and reference_id. This information is displayed through the linked interface classes, *Artefact Interface* and *Artefact Detail Interface*.

Users of this system are modelled by the class *Visitor*. Sophistication and authority properties have been omitted, because no particular sophistication or authority is appropriate to this example. The user interacts with a number of *Window Interface* instances, through which s/he may browse the artefacts of the museum.

Each window uses a *3D window metaphor*, which provides a virtual environment for the to navigate. The referent of all a particular user's windows, is a *Museum Interface* object, which uses a *Shape* metaphor.

The museum interface is composed of a collection of *Artefact Interfaces* and an *Artefact Detail Interface*. Each *Artefact Interface* also uses a *Shape* metaphor and the position of its shape must lie within the bounds of the museum's shape. If the user's intention is to browse a particular artefact, then the artefact detail interface's referent will be set to the selected artefact's referent. This has the effect of showing a form describing the details of the currently selected artefact.

```
schema Virtual_Museum

class Artefact (* Data Class *)
properties
    interface          : Artefact_Interface ref referent ;
    detail_interface   : Artefact_Detail_Interface ref referent ;
    name               : Text ;
    description        : Text ;
    reference_id       : Number

class Visitor (* User Class *)
property
    accessors          : #Window_Interface ref users
operation
    browse is self.accessors.museum.artefacts.browse

class Window_Interface (* Interface Class *)
properties
    museum             : Museum_Interface ref interfaces ; (* referent *)
    users              : #Visitor ref accessors ;
    metaphor           : 3DWindow ref interface ;

class Museum_Interface (* Interface Class *)
properties
    interfaces         : #Window_Interface ref referent ;
    metaphor           : Shape ref interface ;
    artefacts          : #Artefact_Interface ref museum;(*components*)
    detail             : Artefact_Detail_Interface ref museum

class Artefact_Interface (* Interface Class*)
properties
    referent           : Artefact ref interface ;
    metaphor           : Shape ref interface ;
    museum             : Museum_Interface ref artefacts (* composite *)
operation
    browse is self.metaphor.select
constraint
    self.metaphor.position.is_inside(self.museum.metaphor.extent)
trigger
    browse => self.museum.detail.referent(referent)

class Artefact_Detail_Interface (* Interface Class *)
properties
    referent           : Artefact ref detail_interface ;
    metaphor           : Form ref interface ;
    museum             : Museum_Interface ref detail (* composite *)

class Shape (* Visualisation Class*)
properties
    interface          : Artefact_Interface ref metaphor ;
    name               : Text ;
    position           : Position ;
    extent             : Extent ;
    orientation        : Orientation ;
    colour             : Colour
operations
    select ;
    move

class Form (* Visualisation Class *)
properties
    interface          : Artefact_Detail_Interface ref metaphor ;
    fields             : #Text

class 3DWindow (* Visualisation Class *)
properties
    interface          : Window_Interface ref metaphor
    ...
end (* Virtual Museum *)
```