A Lightweight Presentation Model for Database User Interfaces

***Phil Gray*[•] *, Richard Cooper*[•] *, Jessie Kennedy*[•] *, Peter Barclay*[•] *and Tony Griffiths*[•]**

[•] Department of Computing Science, University of Glasgow, 17 Lilybank Gardens, Glasgow G12 8QQ, Scotland. http://www.dcs.gla.ac.uk/ {~pdg || ~rich }
[•] Department of Computer Studies, Napier University, Canal Court, 42 Craiglockhart Ave, Edinburgh, EH14 1LT, Scotland. http://www.dcs.napier.ac.uk/osg/
[•] Department of Computer Science, University of Manchester, Oxford Road, Manchester, M13 9PL, England. http://img.cs.man.ac.uk

Abstract. The Teallach project is building a system which eases the design and implementation of user interfaces (web-based or otherwise) to object-oriented database applications. Teallach takes a model based approach and is constructing its system around three main models – a domain model which describes the database structures with which Teallach can cope; a task model, in which the user-involved aspects of the application can be described; and a presentation model. The presentation model is intended to support the storage of user interaction objects, so that these objects can be found, customised and composed into user interfaces suitable for the tasks. In seeking an appropriate presentation model, we have found that previous models are either those underlying toolkits, which are overly concrete and detailed, or are abstract models based entirely around the look of the interaction objects, but not their intended role. We are implementing a model which describes each interaction object in terms of its purpose, the kinds of data with which the user is allowed to interact, constraints on its use, and a description of configurable aspects of the interface in terms of the ways in which they support interaction. This paper describes this model and gives examples of its use.

1 Introduction

1.1 A Design Space for User Interaction Components

The user interfaces commonly found when accessing the world wide web are simple and provide limited functionality. Built using HTML together with modest amounts of programming in scripting languages or Java, they are appropriate for the majority of the tasks for which the web has been found useful. However, the web is increasingly used as an interface to massively distributed data to which increasingly complex access is required.

In particular, many of the more complex web sites exist principally to give access to a database, frequently managed using an Object-Oriented Database System [Cooper, 97]. Quite often the database will need to be accessed by non-web interfaces as well, and so there needs to be coherent support for the development and maintenance of a mixture of interfaces. Furthermore, the development of the web interfaces might usefully re-use elements of already existing non-web interfaces. A common observation is that the construction of all kinds of user interfaces is complex and typically poorly achieved in the database context. Hence, even if a more complex technique for interaction with web data were to be envisaged, it is unlikely that it would be implemented since the implementation process is so much more complex than most web programming.

The Teallach project is building a system which offers assistance with this problem, since it will provide support for the process of constructing user interfaces to object-oriented databases. Such user-interfaces will include those which are deployed over local area networks, using standard user interface tools such as an SQL command interface, Visual Basic or other GUI toolkit, and ones which use the world wide web for distributed data access. The Teallach system is built in Java and

produces Java applications or applets. Therefore, the examples given throughout this paper are ones which use Java, but the principles discussed should not be thought of as Java specific.

Construction of a user interface is taken here to include design as well as implementation. Whether construction is performed by humans, machines or some combination of the two, among the tasks that have to be performed are:

- *inventing* or *finding* techniques for representing information and for communicating information across the user interface
- where more than one technique is available for a given job, *selecting* from those techniques
- *configuring* the technique so that it fits its context.

All of these tasks require some descriptive framework (a model and/or a language) in terms of which the tasks can be formulated, communicated and reasoned about. Such a model could be said to characterise a "design space" for these tasks, in the sense that it fixes what can be said about interaction techniques. Ahlberg and Truve [Ahlberg and Truve, 1995] say of such a design space that it removes "irrelevant details while isolating and emphasising those properties or artifacts and situations that are most significant to design." We designate such a design space, a presentation model.

Teallach aims to create a system which supports these tasks, by providing declarative models of the context and the user interaction. In this case, context is taken to mean the database structure (i.e. the schema) and the application structure. The interaction aspects (i.e., the means by which information is communicated to and from a user) are described using the presentation model described in this paper.

1.2 The Teallach Context for the Presentation Model

The Teallach System is built around three models:

The **domain model** describes the structure of the database so that the data, meta-data and meta-meta-data are available to the user interface builder – in ways which are not specific to the database or the database system. Since Teallach is concerned exclusively with object-oriented databases, we take the ODMG data model [Catell,1997] as the basis for our domain model – this being an emerging industry standard.

The **task model** describes the organisation of the activities which the user interface is intended to support. The task model permits a hierarchical description of tasks, in which compound tasks can be described as collections of sub-tasks. The temporal ordering of the tasks in the collection can be indicated to be sequential, order-independent, repeatable, parallel, interleaved, user selected, optional or conditional [Griffiths *et al.*, 1998b].

The **presentation model** describes the interaction objects which may be used to permit data use and task execution.

The system allows the designer to make use of as much as he or she knows about the application to help build the interface. It is therefore vital that the three models co-operate to support the design and implementation process. At one level this means that any support tools built by Teallach will look and feel the same whichever model is being used. However, more importantly, the tools must reflect appropriate aspects of each other in the way in which they are presented. As far as the presentation model is concerned, this means that the design space must be organised to bring out the appropriateness of fit of each interaction component to the task intended and to the data type in use. We therefore seek a model which includes this organisation.

The purpose of the presentation model is to support three principal design tasks and (at a stretch) some aspects of a fourth:

Task 1 The interface designer[1] selects between interaction objects which are available in the system;

Task 2 The interface designer customises the interaction objects and organises them into larger units - windows, dialogues, etc.;

Task 3 New interaction objects (constructed outside of the Teallach system) are registered for use with the system.

Task 4 New interaction objects are constructed inside the Teallach system by configuring other interaction objects, in as much as this process involves no more than activities we need to provide to support Task 2.

Thus we will support the process of putting together the user interface from pre-existing components, but we will not provide a full visual programming language for the construction of those components. Section 2 will discuss possibilities for supporting the tasks, but first we will look at the kinds of presentation model which are either in use or have been proposed.

1.3 Kinds of Presentation Model

There are two principal kinds of presentation model, which are used (either explicitly or implicitly):

User Interface Toolkits – These are the kinds of tool which are commonly used in building user interfaces. The "presentation model" is usually provided as a customisable and combinable set of widgets and is usually supported by some kind of design tool.

UIDEs - User Interface Development Environments provide an abstraction of an entire interactive system, including application functionality, task or dialogue structure and presentation components, plus tools for generating and configuring a user interface from this information. Model-based UIDEs represent this abstraction in the form of declarative models. Teallach's presentation model is an instance of this latter class.

We find two main problems with toolkit models, both of which are concerned with the level of abstraction at which they are presented. Firstly, they inevitably discuss user interaction at too concrete a level. Secondly, they are either extremely complex and hard to understand or they can only provide quite restricted interfaces. As a consequence, they do not adequately support the interface design tasks.

The principal function of a toolkit is to provide a *programmer* with the facilities to present the user interface, and sufficient information to build programs using those facilities. A toolkit will usually be provided as a class or function library together with an application programmer's interface and documentation. The documentation will describe the ways in which the classes are to be used in terms of the expected arguments and results of the methods provided.

Furthermore, to use the toolkit, the programmer must understand not just the purpose and features of the interaction objects, but also the ways in which they work. Thus, in a graphical user interface, the programmer needs to understand the event model and the ways in which the events are handled in the program. We consequently find that the programmer either opts for a simple and restricted set of facilities or has to invest considerable effort into mastering the complexities of the implementation. For instance, in building a user interface in Java, the options are to use the Abstract Windowing Toolkit [Java1.1, 1997], which is moderately difficult to understand and only provides the most basic kind of graphical interface object, or to use the emerging Swing toolkit [Java1.2, 1998]. The latter is a collection of around 400 classes, which provide a superior set of

[1] The "interface designer" might be human, machine or some combination of the two. The use of the term here does not imply any particular version of designer.

interaction objects, but whose use involves mastering a more complex and extensive interaction model.

Of course, a toolkit typically comes with some kind of software development tool – for instance Visual Basic. Such tools allow the user a graphical means to choose and place the interaction objects and are very effective, as long as the application and its intended interface are reasonably simple. Even in this case, though, the selection and construction process will usually be guided by the look of the interaction rather than its role and so it is easy for a designer to make errors due to the distance between the intended result and the interaction actually provided.

In summary, the toolkit approach fails to supply the designer with the level of abstraction appropriate to the principal tasks listed above. Instead of being able to find a facility by its intended use and appropriateness of fit, the designer is offered pictures of particular examples, component names (often obscure – e.g. the "combo box") and the API.

Abstract descriptions of the representational components of a user interface have been present in UIDEs for over ten years. In fact, one can identify some of the core ideas in the notions of logical input device and abstract interaction technique developed in the late 70s and early 80s. For example, the basic interaction techniques of Foley, Wallace & Chan [1984], including position, pick, point, choice, text and valuator, are organised around the application data types on which the techniques can operate. Olsen's MIKE and Mickey [Olsen 1986] systems capture the notion of an abstract representation of interaction elements independent of their appearance and concrete behaviour. This abstraction is built on the concepts of application data types to be represented and the application operation to be supported. Information about the actual appearance and behaviour is held separately and can be generated by default and subsequently modified without affecting the basic semantics of the interaction technique. Jade [Vander Zanden & Myers, 1989] is a system which automatically generates dialog boxes and menus based on an abstract description of the interaction plus additional information in the form of generation rules and parameterised interaction objects. Dialogue items are specified by value sets and abstract interaction techniques, including single choice, multiple choice, text, single choice with text, multiple choice with text, command and number in a range.

The form of the abstraction underlying Mike and Jade is based on a differentiation of:

- what the application designer needs to know about the presentational components in order to give meaning and semantic structure to interface components; and
- what the user interface designer needs to know in order to craft usable and effective concrete presentations.

However, these early UIDEs suffered from a concentration on specifying input-oriented interaction techniques and did not handle the representation of application data. Furthermore, their presentation frameworks tended to be ad hoc and to capture only the information needed for initial user interface generation. Other characteristics which form part of the semantics of the abstract interaction objects are only included implicitly or left for the designer to configure by editing the parameter values of the generated interaction object. Model based systems UIDEs [Foley et al, 1989; Elwert, and Schlungbaum, 1995; Luo *et al.*, 1993; Mitchell *et al.*, 1996; Szekely *et al.*, 1996] offer a richer and more expressive description of an interactive system. They support the design activity much more closely, by providing declarative models from which the interface can be generated. They typically provide a variety of models to tackle different aspects of the application building process. Here we concentrate on the presentation model, which [Griffiths et al., 1998b]:

> *"allows the designer to specify the characteristics of the interface components. This can apply to both the static (widgets, etc.) and dynamic (typically involving run-time application-dependent data) facets of the interface."*

In spite of greater expressiveness and more powerful generation and configuratrion tools, the models found in the literature are still a long way from offering to the user interface designer the same powerful abstractions which have been developed for some data visualisation systems [Derthick, 1997]. The presentation models surveyed in [Griffiths *et al.*, 1998a] suffer from a number of problems including

lack of abstraction - features of the model include elements which reflect the actual structure of a (family of) concrete interaction object libraries,

lack of extensibility - the model is *ad hoc* in the sense that it is difficult or impossible to generalise, specialise or extend, and

lack of relevant structure - features which are important for design are left out of the model and features which have different representational potential cannot be distinguished.

In particular, in many existing MB-IDEs we find no place for describing what role the interaction object is expected to play in user interaction. Furthermore, even though the configurable features of the object are describable, there is no attempt to indicate the impact of configuring each feature. In particular, it is clear that some features are central to the functioning of the object – the text string in a text field, for instance. Others are less vital, being configurable only to improve the look of the overall interface – background colour, for instance. In fact, distinguishing between the two is difficult, since, for instance, we might configure an alert box so that error messages are coloured red, while progression messages are coloured green – in this case background colour has become information bearing.A further problem that we find with the declarative models that we have examined, is that they only allow for rather simple mappings between the structure of application data and the interaction objects which represent that data. While this simplicity is acceptable for a system which generates new composite interaction objects from basic primitives for each user interface instantiated, the ability to desacribe more complex relationships is necessary if complex composite interaction objects are to be stored and retrieved by means of high-level descriptions. In the Teallach system, it is vital that the designer be able to find those components which are appropriate to particular kinds of complex data. In addition, we would like to be able to determine any constraints which can hold over the use of the component, both in isolation and in combination.In the Teallach presentation model, we seek to extend the describable aspects of the interaction components to encompass: the role of the component; the data that it can handle; the temporal nature of the relationship between application data and the presentation, the level of complexity of the component; the configurable features; and the constraints which may be imposed on its use.

2 Requirements for the Teallach Model

2.1 The Role of the User Interface in a Database Application

Most of the model-based approaches to user interface description were not conceived in the context of a database system. In building an interface using Teallach, the designer can expect to have access to a meta-model, which describes the structure of all data held by the database system and a schema which describes the structure of the particular database, as well generically provided tools such as query processors which can retrieve data appropriately specified. Consequently, creating a model for interaction is a somewhat less general problem than that tackled by the designers of interfaces to other systems.The meta-model chosen for Teallach is that produced by the Object Data Management Group consortium, since this promises to be an industry-wide standard [Catell, 1997]. This means that user interfaces designed using Teallach should run successfully on any ODMG-compliant database system. The meta-model is object-oriented and

therefore describes data in terms of classes. It further specifies a number of specific classes and meta-classes which largely provide different kinds of collection or different useful domain classes, such as times. The standard also describes a design language for schemata, a query language and bindings to object-oriented programming languages such as Java. A designer will have access to the meta-objects in the ODMG model, the query processor and the schema objects. These will be accessible both graphically and, in implementation detail, using Java's introspection facilities. The Teallach system will permit the specification of tasks and interaction objects associated with the schema and with each other. After the three models have been specified, the Teallach system will generate a Java application or applet which is built on top of the Swing toolset and the Java binding to the OODB being used.There is, therefore, a requirement on the presentation model to be able to guide the designer towards those interaction objects which best fit with the data under consideration. This means that the presentation model must have an aspect which reflects the data structures which the interaction objects can handle. It also means that the registry process must position the description in the most appropriate context. Registration of an interaction object can be expected to occur at two database levels. Firstly, general purpose interaction objects will be registered in the context of the meta model – an interaction object capable of displaying an object of any class will be registered against the class meta-object. Secondly, application-specific interaction objects will be registered against specific classes in the schema. The selection process will then make available to the designer all of the general-purpose and application-specific interaction objects which are available.The multi-level nature of the design space gives us a place to determine the appropriateness of any interactor for a particular purpose. For instance, we assert that an application-specific interaction object is likely to be more useful than a general-purpose one. Furthermore, an interaction object whose registered type is more specific will be favoured over one designed for a supertype, or one which uses fewer of the fields of the class.

2.2 The Presentation Model and Support for Designer Activities

The Teallach presentation model defines what can be said about interaction objects, their properties and their relationships. It can be viewed as a set of assertions about the presentational resources *potentially* available for use by a Teallach run-time system. However, the model does not guarantee that any given presentation conforming to the model can in fact be instantiated. This depends on the actual presentation resources available. In particular, while the model will make it possible to state that an interaction object is composed of component interaction objects arranged graphically in a certain way, creating such an object depends on the existence of tools and/or existing classes which are relevantly configurable.Although it is not the responsibility of the model to provide the necessary tools, useful interaction object classes and required degrees of configurability, the model has been designed to accommodate what we believe we will need in this regard. We have assumed that designers will perform their activities in the following way:In the first task, the Teallach system will support the designer by organising the interaction objects in such a way that the selection available to the designer is appropriately filtered. For instance, if the designer specifies the type of data to be manipulated, then only interaction objects which work for that type of data will be presented. Similarly, the choice of interaction object will be filtered by the purpose of the interaction object and possibly by a specified style as well.The second task can only be supported if the designer has access to the ways in which interaction objects can be modified and combined. Such customisation can vary the specific data access and tasks invoked by means of the interaction object, as well as the appearance and internal behaviour of the interaction object. Composite interaction objects can be "constructed" by associating components in a container. The visual layout of these components can be configured by providing appropriate parameters to the container's layout. Constructing new (i.e., layouts which cannot be specified by parameterising supplied layouts) will not be supported within Teallach.

Teallach designers will not be able to specify behavioural dependencies in a composite interaction object beyond what is offered by the container object's in-built parameters. For example, we do not anticipate being able to construct a set of radio buttons (with dependencies among the buttons) unless there is a radio button container class available. While this reduces the generality of the Teallach presentation model, we believe this is a practical necessity because of the variety of forms of composite interaction objects and the complexity of their internal behaviour.

Registration of interaction object classes into the Teallach system requires that the designer produce a characterisation of the class in terms of the presentation model. This means we need a language for our model.

In summary, the presentation model must provide a description of the purpose of the interaction object; the data types supported; the features which may be modified, the ways composites can be composed and the ways in which the interaction object can be positioned in the user interface. The registry process must specify those aspects in order to fit with Teallach.

The next section describes the aspects of the model in more detail and provides a *preliminary* list of categories for each aspect.

3 The Presentation Model

The Presentation Model describes interaction objects in terms of the following aspects:

- **The Role**. This is the (primary) purpose of the interaction object – i.e. whether it is for displaying, editing, or notifying, etc.
- **The Data Type(s)**. This is a description of which types of data can be manipulated by the interaction object.
- **The Complexity** This indicates whether the interaction object is composed of constituent perceivable elements.
- **The Features**. These are the aspects of the interaction object which can be modified during the process of bringing the interaction object into a user interface.
- **The Constraints**. These restrict the ways in which the interaction object can be used and combined with other interaction objects.

Each of these will now be discussed in turn.

3.1 Roles

The *role* of an interaction object is the purpose for which it is intended. The notion of purpose as we are using it does not refer to the designer-significant purpose (as in the models in [Ahlberg & Truve, 1995] or [Tweedie, 1997]), nor does it refer to the purpose of the interaction object from the standpoint of the interactive application as a whole. Rather it refers to the purpose of the interaction object from the viewpoint of the task and domain elements which use the object. For example, a progress bar might have the design-significant role of helping a user plan his or her activity; it might have the application role of displaying the percentage of a file saved; but it has the Teallach presentation role of dynamically displaying a bounded integer value.

At present we distinguish the following roles:

Display	Such an interaction object receives a (potentially complex) value and makes it perceivable (e.g., places a representation on a screen).
Editing	A value to be represented is received and then returned after potential modifications by a user
Selection	A selection object receives a collection of values and returns a subset

Notification A notification simply returns a value, e.g., a token signalling an event. Typically, a notification's result will be used to invoke an operation in the application.

Related to, but orthogonal with, role is the interaction object's *timeliness*. This property refers to the degree to which the interaction object's received or generated value reflects the current state of the visible representation or the state of the value it represents. Initially, we propose that timeliness be two-valued: *static* or *dynamic*. Dynamic objects perform their role automatically whenever the represented data or any visible representations change. Static objects require some extrinsic trigger to cause them to perform their role.An interaction object can, and often will, combine the two kinds of role. Here is a selection of typical combinations:

- **Static Display**. Such an interaction object receives a (potentially complex) value and places a non-editable representation of it on screen – the display is not then automatically refreshed.
- **Dynamic Display**. Such an interaction object receives one or more objects and places non-editable representations of them on screen. Activities elsewhere in the application which modify the values of the objects will then automatically update the value on the screen.
- **Static Editable Display**. Such an interaction object receives one or more objects and places editable representations of them on screen. The user may modify these representations and submit the edited version to a task.
- **Dynamic Editable Display**. Such an interaction object receives one or more objects and places editable representations of them on screen. The user may modify these representations in which case the values in the object are automatically updated as part of the interaction.
- **Selector**. Such an interaction object is provided with a collection and permits the user to select one or more of them which are returned to a task.
- **Invocation**. Such an interaction object allows the user to invoke a task by generating a notifying value to the application.

3.2 Data Types

Each interaction object can only function in the context of particular kinds of data – selection interaction objects require a collection for instance. The role of this aspect of an interaction object is to list all of the different types of data with which the interaction object can cope. To achieve this we can specify data types using the Domain (i.e. ODMG) Model. The kinds of thing we can specify include generic data types like Collection Of, List Of, etc.; specific types from the ODMG model, such as integer or timestamp; and specific types from the application, such as Person, List of Person, etc.

This aspect assumes that inheritance is available so that subtypes are automatically appropriate if a supertype is specified. It further assumes that you can specify as many types as you like. In general, the specification will be of the *most general* types with which the interaction object can deal.

3.3 Complexity

Interaction objects can be characterised in terms of two different notions of complexity, one relating to the data type of the application data (mapping complexity) and the other relating to the composition of the interaction object (compositional complexity).

Mapping Complexity. We identify three important kinds of interaction object in terms of mapping complexity. A collection takes a set of homogeneous data items for representation, an aggregate takes a disparate combination of data elements (e.g., a

record) and an atomic interaction object takes a single value. Each of these kinds share important features which makes it useful to differentiate them.

Compositional Complexity. A composite has a set of components which are themselves interaction objects. Non-composites (primitives) have no explicit components. Composites represent windows or views which contain and organise interaction with their components. We anticipate that composites will be the only type of interaction object which can be constructed within the Teallach system. While some composites will have a related data type which corresponds to an actual data element in the application (e.g., a form for editing a record), others may well only be associated with the union of otherwise unrelated data types of their components.

These categories are clearly orthogonal. Examples of combinations include:

atomic primitive - textfield;collection primitive - an encapsulated starfield viewer;aggregate composite - a form representation of a record;atomic composite - a scrollbar.

3.4 Features

Each interaction object has a potentially huge number of features which determine its appearance and behaviour. Under this heading, we will allow the person registering the interaction object to list as many as he or she likes. The only requirement is that any such features must be configurable; that is, features are themselves data, distinguishable from the data type of the interaction object, but capable of having their value set.

3.4.1 Properties of Features

A feature can be a colour, font, number, text, etc. This is a potentially open-ended list. The only requirement is that each feature is typed. This will be important both to make it configurable and also to assist in the linking process.

A feature is linked to the application if its value is dependent on some value in the application. Like the interaction object itself, linked features may be static or dynamic, depending upon the update policy.

Features can have default values.

Features can be mandatory or optional. For example, the colour of a button is mandatory, but its label is optional.

3.4.2 Categories of Features

The features can be categorised in a number of ways. The categories which will be supported by the presentation model are as follows:

Linkable Features: Representational vs Decorative

A typical feature of an interaction object is its colour (perhaps subdivided into foreground and background colour). If the colour is linked, we call it *representational*. If it is not linked, the colour is (merely) *decorative*, although careful choice can affect how easy and pleasant the interaction object is to use.

In principle, any feature of an interaction object *could* be linked to some application data so that information is conveyed. However, it may be useful from a design point of view to identify those features which have a particular representational potential. For example, the image in an icon is a stronger candidate for being treated as a representational feature than is the icon's border colour. One way of handling this is to place the features into representational or decorative categories by default, but allow a designer to move them from one category to the other. Only features in the representational category could be linked to application data.

Linked features correspond to *intrinsically* representational features. That is, there is a relationship between the feature and some aspect of the application data, which is maintained by the system. Features can also be representational *extrinsically*. For example, a designer might put in an icon an image which looks like the data the icon stands for. The relationship between the underlying data and the icon is not maintained by (or represented in) the system itself; the relationship depends solely on shared knowledge between the designer and the user. We do not intend to capture extrinsic representational features in our model.

Interaction Support Features

A fundamental distinction can be drawn between features which support the interaction in some way and those which do not. For example, the label on a *textfield* supports interaction with the text by giving a cue to its meaning. Similarly, the axes, gridlines, legend and title of a graph support the interpretation of the graph's data and smooth interaction with it.

The presentation model provides the means to describe such supporting features, usually in the context of a layout. These features are subject to the same kinds of customisation as the linked features described above, For instance, the look (grid points, labelling and so on) of a set of axes can be customised.

3.5 Constraints

Constraints limit the use of an interaction object. There are (at least) two aspects to this:

- **Constraining Acceptable Values**. For instance, restricting a string display to ten characters. We consider such constraints to be just one kind of feature (as discussed above) and should be treated as such.
- **Constraining Customisation.** Rather different from this is the notion that the interaction object cannot appear on its own, but only in a window, or must have one or more other interaction objects configured with it. Providing such kinds of constraint are currently beyond the scope of the presentation model.

4. Some Examples

There follows a first attempt to give a description in terms of our model of actual interaction objects, in this case from the Java Swing library. We follow this list with a discussion with some examples of how these descriptions would be used.

4.1 The Swing Progress Bar

Description	A component that displays an integer value within a bounded interval. A progress bar typically communicates the progress of an event by displaying its percentage of completion and possibly a textual display of this percentage
Type:	atomic primitiveRole: dynamically updated display
Data type:	bounded Integer
Features:	supportive: none linkable border: Line, optional, default=?? decorative: orientation : {HORIZ, VERT} , mandatory, default=HORIZ
Constraints:	acceptable values: none configurational: none

4.2 An Alert Box

Description	A component that displays a text string to the user.
Type:	atomic primitive
Role:	statically updated display
Data type:	text
Features:	supportive: none linkable the text to display decorative: colour
Constraints:	acceptable values: none
configurational:	none

4.3 A Button

Description	A component that can be selected to trigger some activity.
Type:	atomic primitive
Role:	invocation
Data type:	boolean (i.e. selected or not)
Features:	supportive: label or icon indicating meaning linkable the activity that it invokes decorative: colour, border, etc.
Constraints:	acceptable values: none configurational: position within button group

4.4 The Swing List

Description	A component that displays a list of values for selection.
Type:	collection compositeRole: statically updated selection
Data type:	a collection of any kind
Features:	supportive: none linkable the collection to choose from decorative: colour
Constraints:	acceptable values: one or many to be selected configurational: none

4.5 The Swing Table

Description	A component that displays a data in a rectangular form with columns named.
Type:	collection compositeRole: dynamically editable display
Data type:	collection of records of primitive data types
Features:	supportive: none linkable the collection to display decorative: colour, border
Constraints:	acceptable values: specifable ranges on columns configurational: none

4.6 A Dialog Box

Description	A component that contains a number of interaction objects.
Type:	atomic/collection/aggregate composite
Role:	statically updated display (although the components may be dynamic)
Data type:	none
Features:	supportive: none linkable the components decorative: colour
Constraints:	acceptable values: none configurational: none

4.7 A Person Object Data Entry Form

Description	A dialog box that contains editors for name, age and address fields, and submit and cancel buttons.
Type:	aggregate composite
Role:	statically editable display
Data type:	any record or class with name, age and address fields
Features:	supportive: none linkable the components decorative: colour
Constraints:	acceptable values: none configurational: none

4.8 Discussion

When building the user interface, the designer has a list of tasks which require an interface and a description of the schema. Given these and the presentation model, the interface can be specified. For instance, if the application must provide ways to add new objects, query objects and display objects, the development of the interface will include:

- Building an introductory window or page. Since this will include initiating the major tasks, the designer will look for interaction objects which invoke tasks and may choose the button described in 4.3.
- Providing a way of inputting new data objects. If the objects represent people, the form entry described in 4.8 might be selected.
- Displaying the value of an object. The Swing List described in 4.4 would be one of the interaction objects offered to the designer to choose which object.

At each stage, the designer has a task to support and a range of interaction objects to choose from. Once chosen, the features of the objects are fixed. If, in this process, specialised forms of interaction object are created which might be useful in future, these can be registered with the system.

5. Summary and Proposals

We have described a presentation model which supports database application user interfaces. The focus of the model is on the purpose of the interaction objects and how they support interaction with the database data. The data is presumed to be object-oriented and the intended goal is the generation of Java programs.

During the paper, we have tried to identify the precise level of abstraction which would be most useful to the designer. We feel that this should neither be at the level of the appearance of an interaction object, nor at the level of how it is implemented, nor yet by how the code implementing

it is invoked, but rather in terms of the use to which it might be put. Accordingly, we have made the role of the interaction object the most important feature of the model.

To the role, we have added components which describe how the object can be configured and constrained, which kind of data it can handle and how it is organised in terms of other interaction objects. Each of these can be specified, although default values will be available for most aspects of the description. The work in this paper represents only the beginnings of a presentation model for Teallach. Finding just the right level of abstraction and the features needed for our presentation model to meet the needs of the Teallach system and Teallach users requires understanding the "language" of modern user-computer interaction, including the semiotics of the different presentational structures and interaction techniques. For example, our model should be able to characterise what makes a drop-down menu the same as, and different from, a scrollable list of selectable items. and what characteristics of each are modifiable without breaking the conventions of use and appearance which apply to each.We are just beginning to implement a preliminary version of the model as described, in the expectation of creating a useable component for the Teallach system. Once this has been achieved, we hope to explore a number of other issues. Firstly, although we are at pains to specify the model in an implementation independent manner, any implementation will be carried out in the context of Java and Swing. We will hope to ensure language independence by providing alternative implementations.

Secondly, we are very interested in any method for reducing the work in accomplishing the various tasks discussed in the introductory section. For instance, the detailed description of the model implies a cumbersome and the time-consuming registration process. We expect, by the use of defaults and wizards to cut down the effort involved here considerably. However, until we have had experience of our model, it will not be clear what assistance would be beneficial.

Other assistance might come through support for *styles*. Clearly, a designer would hope that any interface provided for an application would have a consistent look-and-feel. One method of ensuring that is to allow the specification of a style which will then provide a consistent set of defaults for any interaction objects used in the interface. This will include suggesting particular interaction objects to the designer more strongly than others, and also automatically setting the values of features.In the definition of the data type of an interaction object, we have asserted that we will be able to specify several specific applicable types. We would also like to be able to specify the data type in such a way that *type inference* is used to checked for applicability. That is, we might specify the data type as *Class(name:string, age:int)* and then make the interaction object applicable to objects of any class that has at least those two fields.Fundamentally, though, we believe that a significant advantage will accrue from the production of a presentation model based around the intended use of the interaction objects which will make up the user interface.

Acknowledgements

This work is funded by the UK's Engineering and Physical Sciences Research Council (EPSRC) under grant GR/L02692, whose support we are pleased to acknowledge. We also acknowledge the significant contributions made by other members of the Teallach consortium who have significantly influenced the design, including Norman Paton and Carole Goble of Manchester University and Andrew Dinn of Napier University.

Bibliography

Ahlberg and Truve, 1995

Ahlberg, C. and Truve, S., "Exploiting Terra Incognita in the Design Space of Query Devices", in *Engineering For Human-Computer Interaction*, Chapman and Hall, 1995.

Catell, 1997

Cattell, R.G.G. et al.: "The Object Database Standard: 2.0". Morgan Kaufmann Publishers, Inc. 1997.

Cooper, 1997

Cooper R., "Object Databases", International Thomson Computing Press, 1997.

Derthick, 1997Derthick, M., Kolojejchick, J. A., and Roth, S. F. **An Interactive Visual Query Environment for Exploring Data.** *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '97),* ACM Press, October 1997, pp 189-198. Elwert, and Schlungbaum, 1995

Elwert, T., Schlungbaum, T. "Modelling and Generation of Graphical User Interfaces in the TADEUS Approach", in *Design Specification, and Verification of Interactive Systems* (eds. P. Palanque and R. Bastide). Wien, Springer, 193–208, 1995.

Foley et al, 1984

Foley, J., Wallace, V.L., & Chan, P. "The human factors of computer graphics interaction techniques", IEEE Computer Graphics and Applications, 1984, pp. 13-48.Foley et al, 1989

Foley, J., Kim, W.C., Kovacevic, S, & Murray, K. "Defining Interfaces at a High Level of Abstraction", IEEE Software, January 1989Fowler, 1997

Fowler, M. with Scott, K., "UML Distilled", Addison Wesley Object Technology Series, May 1997.

Griffiths *et al.*, 1998a

Griffiths, T., McKirdy, J., Forrester, G., Paton, N., Kennedy, J., Barclay, P., Cooper, R., Goble, C., & Gray, P., "Exploiting Model-Based Techniques for User Interfaces to Databases", in Proceedings of VDB-4, Italy, May 1998

Griffiths *et al.*, 1998b

Griffiths, T., McKirdy, J., Paton, N., Kennedy, J., Cooper, R., Barclay, P., Goble, C., Gray, P., Smyth, M., & Dinn, A. (1997), "An Open Model-Based Interface Development System: The Teallach Approach", DS-VIS'98, Cambridge, June 1998

Java 1.1, 199

Javasoft Web Site, http://java.sun.com/products/jdk/1.1/

Java 1.2, 1998

Javasoft Web Site, http://java.sun.com/products/jfc/swingdoc-api/

Luo *et al.*, 1993

Luo, P., Szekely, P., & Neches, R., (1993), "Management of Interface Design in HUMANOID", in *Proceedings of InterCHI'93*, Amsterdam.

Mitchell *et al.*, 1996

Mitchell, K., Kennedy, J., Barclay, P: "A Framework for User Interfaces to Databases". In *ACM International Workshop on Advanced Visual Interfaces*. Gubbio, Italy, 1996.

Olsen 1986

Olsen, D. "MIKE: The Menu Interaction Kontrol Environment", ACM Trans Graphics, 5,4 (Oct 1986), pp. 318-344.Szekely *et al.*, 1996

Szekely, P., Sukaviriya, P., Castells, P., Muhtkumarasamy, J., Salcher, E. "Declarative Interface Models For User Interface Construction Tools: The MASTERMIND Approach", in *Engineering For Human-Computer Interaction*, 1996.

Tweedie, 1997

Twedie, L., "Characterizing Interactive Exernalisations", CHI 97, ACM, 1997.

Vander Zanden and Myers, 1990 Vander Zanden, B. & Myers, B. "Automatic, Look-and-Feel Independent Dialog Creation for Graphical User Interfaces", CHI '90, pp. 27-34.