9

Conceptual Modelling for Database User Interfaces

Richard Cooper[1], Jo McKirdy[1], Tony Griffiths[2], Peter J. Barclay[3], Norman W. Paton[2], Philip D. Gray[1], Jessie Kennedy[3] and Carole A. Goble[2]

[1]*Deepartment of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK.*

{rich, jo, pdg}@dcs.gla.ac.uk

[2]*Department of Computer Science, University of Manchester, Oxford Road, Manchester M13 9PL, UK.*

{griffitt, norman, carole}@cs.man.ac.uk

[3]*Department of Computing Studies, Napier University, Canal Court, 42 Craiglockhart Avenue, Edinburgh EH14 1LT, UK.*

{jessie,pete}@dcs.napier.ac.uk

Abstract Model-based user interface development environments show promise for improving the speed of production and quality of user interfaces. Such systems usually have separate description of domain, task and presentation structure. The Teallach system applies model based techniques to the important area of database interfaces, which increases the importance of domain information. This exists in the form of a schema and can be captured in a high-level format, so that the developer need not build a domain description arbitrarily. This paper describes such a Domain Model, how it is captured and how it contributes to the systematic development of a user interface.

Keywords: Model-based UIDE, Domain Model, Conceptual Modelling

1. INTRODUCTION

Database systems have long been criticised for providing inadequate facilities for user interaction. Building a user-interface previously required recourse to significant amounts of programming to augment the database. Although most commercial systems now supply integrated support for application user interface development, the support is neither systematic nor

H. Arisawa et al. (eds.), *Advances in Visual Information Management*

of complete generality. Developers cannot re-use components from previous applications, nor can they go beyond the limited scope of the interface development environment, without returning to the need to write programs.

The model-based approach [Szekely 1996] is more systematic and generalisable, the developer first constructing declarative specifications or models of the various aspects of the user interface, from which the application and user interface are produced. A *Model-Based User Interface Development Environment* (MB-UIDE) permits the description of the following aspects: a *domain model* which describes the structure of the data; a *task model* which describes the structure of the activities; a *presentation model* which describes the user interface features; a *user model* which describes characteristics of various user roles; and a *dialogue model* which describes the interaction between user and application. For a full account the capabilities of such models, the reader is referred to [Griffiths 1998].

Previous MB-UIDEs have been built in the context of applications in general and have not typically been applied to data-intensive applications. The Teallach system [Barclay, 1999] has been constructed to explore the possibility of using the MB-UIDE approach in the context of Java-based Object-Oriented Database Systems (OODBS). Teallach supports the specification of Domain, Task and Presentation Models for the application, providing meta-models for each of these – Teallach does not, as yet, have a User Model, while Dialogue aspects are integrated into the Presentation and Task Models. Teallach is supplied as a single piece of software in which the models can be developed and integrated before an application is generated.

Focussing on database applications brings the Domain Model (DM) into a central position. In previous MB-UIDEs, the DM has played a subsidiary role. Given database involvement, several considerations change. Firstly, there is already a version of a Domain Meta-Model – the logical data model of the database system. Secondly, when considering an OODBS, this model is not consistent. Moreover, when we restrict ourselves to the need to provide user interface development tools for existing databases, a DM already exists (the schema) and need not be developed from scratch.

This short paper briefly describes how domain information is extracted and used in Teallach. More details of the Teallach system can be found at the Teallach web-site.[Teallach,1999].

2. THE DOMAIN MODEL OF A MB-UIDE

The DM of an application in a MB-UIDE describes the structure of the data. This activity is very familiar to database programmers as it is nothing more than conceptual schema design. In fact, the task of specifying the DM

of the application has already been carried out, since the interface is being developed in terms of an existing database. Instead of the developer having to specify the DM as one of the main tasks, the task is to pick up the existing database schema and use it as the DM. To do this, Teallach connects to the database and retrieves the schema from the meta-data, creating an internal representation of it which is the DM. Teallach does not support (presently) the tasks of schema design/DM specification nor the modification of these.

The DM is a conceptual model, and the Teallach DM has the goal of describing as much of the data structure as required for describing the user interface. The DM is therefor high level and describes the database in terms of concepts appropriate to a user's understanding of the application domain.

Teallach operates in the context of OODBS[Cooper, 1997] such as POET[POET, 1999] or Objectivity [Objectivity, 1999]] and hence the DM is object-oriented. The Teallach DM describes data in terms of classes, methods, inheritance and so on. However, the lack of a common and agreed OO model is a significant problem. Each OO concept (inheritance, information hiding, etc.) has a number of divergent semantics and the product of these individual variations gives rises to an enormous number of meanings of the term "object-oriented data model", many of which are realised in OOODBS products.

Since one of the goals of Teallach is to be platform independent, we need a single consistent model for our internal representation of the DM. Here we make concrete use of the main OODBS standardising - the ODMG standard, which comprises definitions of an object model, an object schema definition language, an object query language and language bindings to Java, C++ and Smalltalk. It is only the Object Model which interests us.

We would have liked to use the object schema definition language (ODL) to derive a standard description.. However, compliance with the ODMG standard does not extend to the use of ODL in any form. What we can expect from ODMG-compliance is the existence of a relatively standard form of the query language and language bindings. Schema descriptions are not standardised, nor is the underlying data model.

We had, therefore, to determine, on a product-by-product basis, the appropriate mechanism for retrieving the schema of a database. For instance, the POET 5.1 Java Binding[POET, 1997], uses an associated configuration file which names the classes to be found in the database. From this, we can begin to create a DM based on the classes named in the file. We then exploit the fact that we are developing our system in Java, in which case we can now make use of Java's introspection mechanisms. We find and load the classes in the database using this mechanism and introspect over them to discover the variables and methods in the classes. From this we create a complete description of the DM. In the next section, we consider the form that this description takes.

3. A DATABASE INDEPENDENT DOMAIN MODEL

The DM of Teallach is based strongly on the ODMG Data Model[Cattell *et al.*, 1997]. This was an important contribution to the success of the project since we did not have to invent a model; and could expect a fair degree of conformity between the "ODMG-compliant OODBMS" and this model. Consequently, capturing metadata in this model is greatly simplified. Here, we briefly review the ODMG model and what it supports, before we discuss the implementation which constitutes our DM.

3.1 The ODMG Data Model

The ODMG data model provides a standard to which OODBMS products should adhere. The model is for database use and so it is intimately concerned with issues of efficiency, data modelling and database management. The latter has meant that the ODMG model includes a view of how databases are structured, has a specification for transactions, has standard classes for domain types and collections, and provides for user-detectable keys.

The model comprises several parts: distinct parts which deal with *objects* and with *literals*; support for structured (i.e. *record*) values; the distinction between two kinds of class variable: *relationships*, which are objects with automatically maintained inverse references; and *attributes* which may be literals or objects, but which do not automatically support an inverse reference; multiple *inheritance* of behaviour, but single inheritance of state; *database specific extensions* including extents and keys; and *exceptions*.

This means that an ODMG schema constitutes a rich description of the structure of the data and provides the Teallach system with substantial support in building a description of the database that can be of great use in the interface development process.

3.2 The Teallach Domain Model

The Teallach Domain Model is based heavily on the ODMG Model. Unfortunately, since we require a concrete implementation of most of the above aspects, we have run into several instances of a lack of precision in the ODMG specification. At any point of ambiguity, we have had to take a particular view in order to complete our implementation.

The Domain Meta-Model is essentially a Java package that realises each of the ODMG concepts as a class. There is a repository class, whose instances corresponds to database schemata. The class descriptions are

created as instances of the metadata classes in the ODMG model, such as Operation, Parameter, Type, Exception and so on.

There have also been classes created for Databases, Transactions and OQL Queries. Each of these has special responsibility for encapsulating some aspect of user interaction with the database. The query class is important, as it means that the interface can exploit any query optimisation techniques which have implemented.

The principal role of the DM is to represent the underlying application, and database connectivity and interaction. In addition, however, it models *auxiliary* data types that may be required to describe transient data vital to the runtime operation of the application and interface. Auxiliary data is also modelled using the ODMG derived building blocks in order that the representation of domain components is orthogonal to their source and persistence. It is vital to have the ability to model transient data, since any sophisticated user interface will require the ability to manipulate data which is passed between interaction components but never stored.

3.3 Creating a Teallach Domain Model

The Teallach system uses the DM component as a fixed point for interface development. The developer fetches a schema from the database system, which is then visualised as a hierarchical display of classes and their components. Teallach supports the processes of capturing a DM and describing the other models in any order. To do so it proceeds as follows:

- The list of persistence capable classes in the application is requested from the database system. In the case of POET5.1, the configuration file holds both the names of these classes and the application name.
- Each of the classes is then introspected in order to discover their internal structure in terms of their fields, methods, and exceptions.

This results in a DM represented as a collection of instances of the DM classes. The information contained within the DM for a given application can be used in two ways. Firstly, it can assist and inform the designer in the generation of the other models in the Teallach system. For example, the functionality available on a particular class can assist the designer in developing a part of a task model, while the type of a parameter in the signature of an operation can enable the designer to decide which presentation widget would be the most appropriate for displaying or obtaining that information. It can also be used by the Teallach system to generate components of other models for the given application automatically. The nature of the relationships between the different Teallach models will be discussed in the next section.

4. INTEGRATING THE DOMAIN MODEL WITH THE OTHER TEALLACH MODELS

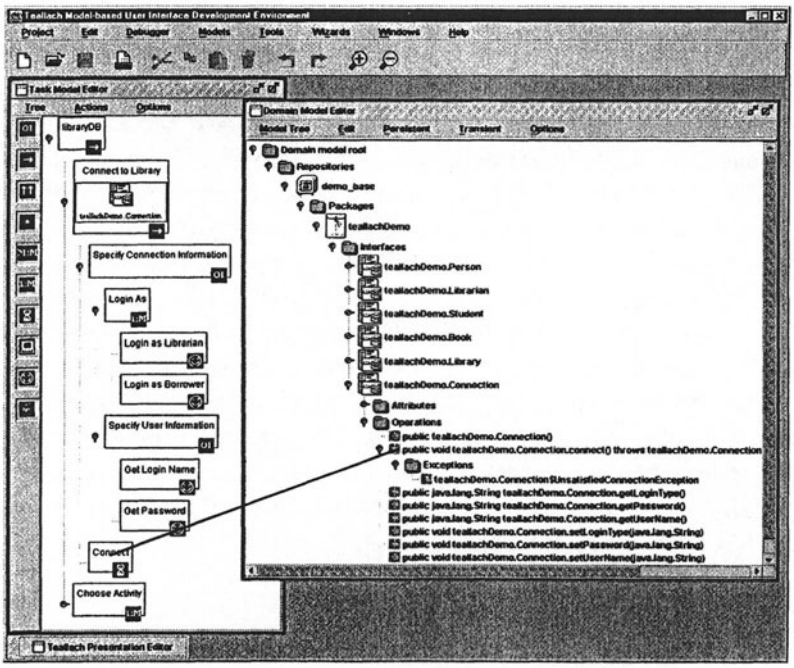
After the DM has been captured, Teallach allows a developer to create the rest of the interface description in any order. This section shows how the development process proceeds from the DM. We start by describing the design environment in general, then show how the Presentation Model is developed, followed by the Task Model. Finally, we show how the whole process is completed.

4.1 The Teallach Design Environment

The figure shows a screenshot of the Teallach design environment, which consists of editors for the three models within an overall tool environment providing project management, editing, model linking and code generation facilities. The toolkit uses a desktop metaphor and direct manipulation techniques for model construction.

This environment supports the following tasks:

- the capture of the Domain Model as described above;
- the creation and editting of a Task Model;
- the construction of a Presentation Model from an available toolkit;
- linking the various models; and
  generating the user interface thus described.



The *Task Editor* supports the specification of both the structure of user tasks and the flow of information between the models when carrying out the user's tasks. A Task Model is a goal-oriented task hierarchy, with leaf nodes representing user interaction or database tasks. Non-leaf-nodes specify the temporal ordering of the children, e.g. indicating sequential, parallel or concurrent sub-task execution. The Task Pane allows the task hierarchy to be edited by modifying, adding or removing tasks. Furthermore, leaf tasks can be associated with domain or presentation components as described below.

A *Presentation -Model* describes the appearance and surface behaviour of the user interface. It has both a concrete and an abstract aspect. Concretely it is a widget set out of which components of the user interface are built. Abstractly it partitions the space of widgets in terms of their purpose, e.g. display, edit, choose, invoke action.. Teallach supports a repository of widgets registered in terms of both concrete and abstract representations. The Presentation Pane shows a tree-view of the widgets and the ability to toggle between abstract and concrete views.

Having constructed the three models in the separate sub-windows, there are two ways of associating elements from different models: *linking* and *generating*. *Linking* creates an association between existing components in two models, for instance linking a task to a presentation or domain action. *Generation* creates new components in one model based on components in another, for instance generating a task from a DM operation.

Linking is achieved by drawing a rubber-banded line from an element of one model to an element of another, to associate the two. This can be seen in the figure, which shows a link being created between the task model `Connect` action task and the domain model `connect` operation which will mean that if the generated application executes this task, the database connection operation in the DM will be invoked. Teallach currently uses a simple hyperlink metaphor to show associations between linked model components; this allows the designer to jump to an associated component by invoking its *show linked components* operation from a pop-up menu.

Generating components in one model from components in another is achieved by drag-and-drop. The designer simply drags a component from one model and drops it at the desired location in the target model. For example, the designer may construct a partial task hierarchy corresponding to some constructed presentation (or *vice versa*). Once the new structure has been generated, the relationships between components are maintained through the services provided by the Teallach store.

The final step takes the three completed and inter-connected models and using these for the generation of Java source code to present the application using the desired interface. The final code will consist of: calls to Swing, calls to the DM classes and a set of calls mirroring the task structure.

4.2 Presentation and the Domain Model

As described above, there are two ways in which the DM can be used to determine the Presentation Model: by *linking* DM components to existing interaction objects; or by *generating* presentation objects from DM components. The former is used if the developer has created an interface and wants to link it to the database. The second is used if the database structure is used to generate the interface directly.

There are several ways in which the DM components can be linked to Presentation Model components. Among these are:

- associating an interaction object with a DM operation – e.g., a button might be placed on the interface to summon an operation;
- associating an object type with a container e.g. a dialogue box – in which case, dialogue box content will be subsequently added; and
- associating an atomic DM component with a primitive interaction object –for instance, linking a string property with a text field.

There are also several ways in which DM components can be used to generate Presentation Model components. Among these are:

- dropping an operation onto a presentation object type will generate an instance capable of invoking the operation;
- dropping a class onto a complex presentation type will generate a default structure; and
- dropping a primitive object type, such as a string, onto the presentation generates a new interaction object, such as a text box.

In fact, the developer can interleave fragments of the linking and generating processes if so required.

4.3 Tasks and the Domain Model

The DM can also be used by linking domain information to previously defined tasks, or as a basis for generating task descriptions.

The former may be used to:

- link a class to a compound task, which, for instance, edits objects of that class; or
- link an operation with an action task which is a leaf node of the task hierarchy.

The latter is used to:

- generate a new leaf task corresponding to an operation; or
- generate a new compound task corresponding to the editing of an object.

These possibilities are exactly equivalent to the connections between Domain and Presentation Models. In this case either the whole Task Model

is laid out first and elements of it are then linked to the DM, or elements of the Task Model are generated as default activities relating the DM components.

5. CONCLUSIONS

The paper has described a novel use for conceptual modelling which brings together two similar technologies that have typically been kept quite separate. On the one hand, the world of Model Based User Interface Development Environments is concerned with providing just as much application description as is necessary to complete a user interface description. On the other hand, the world of conceptual data modelling promises a similarly high-level description, but one from which an implementation data structure can be inferred. Both kinds of model are high level, since they provide intuitive descriptions, but are implementation-oriented since implementations must be developed from them.

It is the high-level nature of both of these models which brings them together as the DM of the Teallach system. In Teallach, the DM is extracted from the schema of an existing Object-Oriented Database. The object-orientation implies the existence of a higher level description than is found in classical database systems. Moreover, having restricted our attention to Java databases, we have tools for discovering database structure and transforming it into a consistent internal structure.

Having achieved such an internal structure, we use it as the basis of interface development, by adding two other declarative models – one for presentation and one for the task structure. These can either be developed completely separately and then linked to the domain description, or can be partially developed from the domain information and subsequently enhanced and completed.

We have turned the usual conceptual methodology around – performing a kind of reverse engineering. Instead of generating a low-level description from a high level model, we have created a high-level model from an installed database and then used it as the basis of application development. The installed database, being object-oriented, contains enough self-description to allow a complete representation to be abstracted from it.

ACKNOWLEDGEMENTS

This research is funded by the UK Engineering and Physical Sciences Research Council (EPSRC), whose support we are pleased to acknowledge.

We would also like to acknowledge the useful comments of Karen Renaud on a draft of this paper.

REFERENCES

Barclay PJ, Griffiths, T., McKirdy, J., Paton, N., Cooper, R. and Kennedy, J (1999) "The Teallach Tool: Using Models For Flexible User Interface Design, in proceedings of CADUI'99, Louvain-la-Neuve (Belgium), 21-23 October 1999

Cattell, R.G.G. et al (1997): The Object Database Standard: 2.0, *Morgan Kaufmann.*

Cooper, R. (1997) "Object Databases", *International Thompson Press.*

Foley, J., Kim, W. C., Kovacevic, S., & Murray, K. (1989) Defining Interfaces at a High Level of Abstraction. IEEE Computer, 25-32.

Griffiths, T., McKirdy, J., Forrester, G., Paton, N., Kennedy, J., Barclay, P., Cooper, R., Goble, C.& Gray, P.D. (1998) Exploiting Model-Based Techniques for User Interfaces to Databases, Proc VDB4, Y. Ioannidis and W. Klas (eds), *Chapman&Hall.*

Johnson, P., Johnson, H., & Wilson, S. (1995) Rapid Prototyping of User Interfaces Driven by Task Models. In Scenario-Based Design, (Carroll, J. ed). *John Wiley & Son*, 209-246.

Mitchell, K., Kennedy, J., Barclay, P.(1996) A Framework for User Interfaces to Databases. In Proc. AVI, ACM Press.

Objectivity (1999) www.objectivity.com

POET (1997) POET Software, "POET Java SDK Programmer's Guide", POET Software.

POET (1999), www.poet.com

Schlungbaum, E., Elwert, T.(1996) Automatic User Interface Generation from Declarative Models. In Proc. CADUI, 3-18.

Szekely, P., Luo, P., & Neches, R. (1992) Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In Proc. CHI 92. 507-515.

Szekely, P.(1996) Retrospective and Challenges for Model-Based Interface Development, Proc. DSVIS, F. Bodart and J. Vanderdonckt (eds), 1-27, *Springer-Verlag.*

Teallach (1999), www.dcs.gla.ac.uk/research/teallach/

BIOGRAPHIES

Teallach is an EPSRC funded research project which has been running since October 1996. It is a collaboration between the universities of Manchester, Glasgow, and Napier, with industrial collaborations including IBM, ICL, British Geological Survey, and Criterion Software. The focus of our investigation lies in the systematic and generic support for the development of user interfaces to object-oriented database applications.

The principal aim of our research is the development of models, a software architecture, and tools which will facilitate the rapid development of user interfaces to ODMG compliant databases.