A Framework for Describing Visual Interfaces to Databases

Norman Murray, Carole Goble and Norman W. Paton
Department of Computer Science
University of Manchester
Oxford Road
Manchester M13 9PL, U.K.
email:<murrayn, carole, norm>@cs.man.ac.uk

Abstract

In the field of HCI there exist many formalisms for analysing, describing and evaluating interactive systems. However, in developing and evaluating user interfaces to databases, we found it necessary to be able to describe presentation and interaction aspects that are catered for poorly or not at all in current formalisms. This paper presents a framework for the systematic description of data model, presentation and interaction components that together form a graphical user interface. The utility of the framework is then demonstrated by showing how it can be used to describe two existing visual query interfaces. These examples show that the framework provides a systematic method for the concise description of graphical interfaces to databases that can be used either during interface design or as a communication aid.

1 Introduction

Research in user interfaces for databases is gaining momentum with many recent conferences and workshops [10, 21, 22, 23, 36]. However, many papers on database interfaces give ad-hoc and imprecise descriptions of presentation and interaction aspects of interfaces, with many details being unclear or left to the imagination of the reader. This makes it difficult to grasp what systems can and cannot do, or to analyse precisely how specific tasks are carried out.

Traditionally, interfaces have been built from the system's viewpoint, by taking the capabilities of the system and building an interface onto the system to provide users with access to its functionality. As a consequence of this approach, most design methods focus on the behaviour that a system should support, and have limited facilities for describing presentation or interaction aspects. HCI sees the need to build the interface to a system from the user's viewpoint by taking the tasks they wish to accomplish and then building an interface that is suitable for performing these tasks. Many tools exist for supporting the construction of interfaces, but much less progress has been made with techniques for describing the user's interaction and the look and feel of the interface.

To design a suitable framework, the cycle of interaction between the user and the system needs to be examined, and the components that form this cycle identified. This paper provides a framework for describing and relating these components, so that the user interaction cycle can be described systematically, concisely, and much more precisely than is typically the case in informal descriptions. Previous methods such as task grammars and state charts have proved inadequate at fully describing the visual aspects of graphical interfaces. Our framework aims to address this issue and makes extensive use of visual icons to reflect more fully the operation of graphical interfaces that has been previously lacking. The framework can be used to generate descriptions of existing graphical interfaces for databases or as a communication mechanism for designers and implementers, allowing the comparison and evaluation of database interfaces.

The paper is structured as follows: Section 2 describes the interaction cycle and identifies the components of interest. Sections 3 to 5 describe the components that form the framework and include the definition of the framework. Section 6 shows examples of using the framework, in particular the graph-based query language Gql [27] and the form based query interface QBE [40]. Section 7 details related work on formalisms for describing interactive systems, and conclusions are presented in Section 8.

2 The Interaction Cycle

To build a framework we must first define the components that form the interactive system and select those components that the framework needs to model. The general interaction framework defined by Abowd in [1] consists of the system, the user and the inputs and outputs of the system.

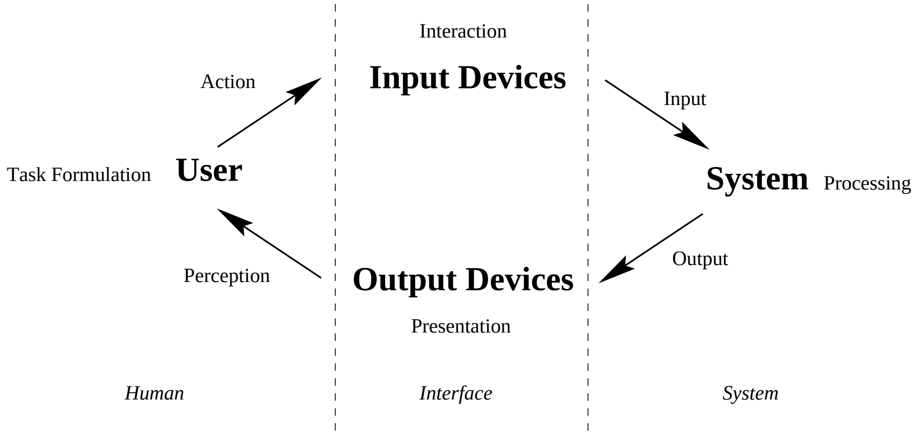


Figure 1: Interaction cycle

Figure 1 shows the cycle of interaction starting from the user observing the state of the system, deciding on a plan of action, and executing the task at the interface via the input devices. These actions are then transferred as input to the system which processes them, posting output to the interface for presentation via the output devices.

The framework presented here aims to address the look and feel aspects of a graphical database interface. Therefore, we are not interested in how users formulate their tasks, or how the system processes their input. The aspects that the framework needs to address are:

- The system concepts and operations, as these are the components that the user is interacting with. How these components are implemented in the system is not of interest (Section 3).
- The presentation of the system concepts and operations, as these form the interface that the user is presented with (Section 4).
- The user's tasks and the interactions with the system required to perform these tasks (Section 5). This is the behavioural component of the interface.

For a database interface, the above aspects can be considered together as the database environment:

Definition 1
Every Database Environment can be seen as a triple DBE=< DM, P, $\{T\}$ > defined as follows:
DM is the data model underlying the database.
P is the presentation associated with the data model's concepts.
$\{T\}$ represents the set of tasks that can be performed at the interface. These tasks will be composed of one or more interaction cycles the user needs to perform to achieve the task's goal. It should be noted that it may be possible to perform a task by differing combinations of interaction cycles.

Each of these components, the data model, presentation and tasks, and how they form the framework are detailed in the following sections.

3 Data Model

The framework has to be able to capture the database concepts on which the interface acts, as to use the interface the user must either know the system's concepts and operations, or be informed of them by the system. In general, user interfaces to databases act directly on the constructs defined by the data model.

A typical data model is composed of a database concepts part describing the database components and the rules according to which the database can be constructed, e.g. attributes, classes, database schemas, etc. , and a data manipulation part defining the operations that can be performed on the data concepts, e.g. updating, retrieval, and creation of the data. A data model can therefore be defined within the following framework:

Definition 2
Every Data Model can be expressed as a tuple DM=< DC, DO > where:

DC represents the database concepts, defining the components that form the database interface.

DO represents the presented operations that can be performed on the data, from schema creation to querying.

The database interface generally provides visual representations for the database concepts and operations and so will define the tasks that the interface could support.

There exist several data models such as relational, extended relational, functional and object oriented. The type of data model dictates the components of the environment that can be visualised and the operations that can be performed, so for each interface it will be the underlying data model that specifies how the above definition is populated. The database interface does not have to use all of the data model concepts, as the interface may only provide the user with a subset of the data model's operational capabilities. For example, with an object oriented database the database concepts could include the attributes, classes, methods, extents, etc., as well as the database schema, and query schemas if these were visualised by the interface. The database schema is a composition of database concepts, that is it is constructed from the classes that form the database. When visualised it could show the classes of the schema and their attributes and relationships within the classes, whereas the visualisation of an attribute may only be composed of its name. We shall examine this distinction in the next section.

4 Presentation

The concepts in the data model (e.g. data concepts such as the classes and instances in the object model, the database schema, and operations such as delete and select) each require to be visualised and thus made available for user interaction. The output devices attached to a system convert the data's internal representation into a form perceptible by the user. Database interfaces have historically moved from being textual (e.g. SQL [11]) to form-based (e.g. QBE [40]), diagrammatic (e.g. GUIDANCE [19], Gql [27]) and iconic (e.g. ICONICBROWSER [39]), from single to multi paradigm (e.g. [14, 9]), and from two to three dimensional (e.g Graphical Database Browsing [7], AMAZE [5]).

The spatial layout of the visualisation defines how the items that form the interface are arranged in relation to one another. To represent this we have chosen a simple, extensible set of functions, with each style of visual presentation having its own functions to describe its characteristics. The layout functions used in this paper are defined in Table 1.

A full set of layout functions cannot be defined due to the diversity of graphical interfaces, but this limited set has proven useful in applications of the framework. The framework is not rigid, in that it can be extended to cope with new styles of graphical presentation.

The data model (DM) defines the database concepts and operations that are to be presented by the interface, and each of these will have an associated set of presentations.

Definition 3
A concept can have a set of presentations associated with it that can be represented as: P=< DMC, {CV} >, defined as follows:

DMC is a data model concept from the data model that is to be visualised.

CV is the set of visualisations that are associated with the concept.

Layout function	Definition
grammar	Can be used to describe the textual aspects of an interface by using a Bakus-Naur Form, BNF, style of representation.
table	Is used to reference elements of a table by their row and column locations, table(row, col), e.g. table(1, 1..n) references all the items in the first row. This notation can be easily extended to handle tables of greater than two dimensions.
in	The in function accepts two arguments, e.g. in(x, y) which states that the concept x is located or placed inside y.
connected	Used to describe items that are connected such as nodes of a graph connected by an arc, e.g. connected(x, y, z) indicates that the nodes x and y are explicitly connected via an arc, z.
beside	Beside is similar to connected except that there is no visualisation of the connector or arc. The concepts are located in close proximity to one another due to some relationship between them, e.g beside(x, y).

Table 1: Symbols used in interaction

We noticed the distinction between simple data concepts that only present information relating to themselves, or more complex data concepts that are composed of information from other data concepts. For example, a class can be presented simply by displaying its name. Alternatively a complex presentation of the class could be composed of its name and associated attribute names. Therefore a data concept can be composed simply of its information, or be a composition of other data concepts and their associated presentations.

Definition 4
The presentation of the concept is composed of a simple visualisation or a set of data concepts and their presentations and a spatial layout attribute that can be represented as follows: CV=< V or {DMC}, L >, defined as follows:

V is the visualisation associated with the data model concept.

{DMC} are the set of data model concepts that together form the presentation of a more general data model concept.

L defines the layout function associated with the data model concept – the spatial layout of the concept.

5 Tasks

Interaction between the user and the system takes place via a set of input devices that form the medium of communication. As the user interacts with the input device, this interaction is transformed into input that the system can process. Interaction devices take the form of keyboards, mice, track-balls, touch screens, 3D trackers, etc. [13, 33].

Figure 1 defined the cycle of interaction between the user and the system. The users perform tasks by interacting with the input devices to achieve a specific goal. The performing of a task usually involves a sequence of interaction cycles or subtasks, with each cycle contributing towards the completion of the task and so satisfying the user's goal [12].

With Hierarchical Task Analysis (HTA) [2, 26], tasks can be broken down into subtasks. The subtasks that have arisen from the decomposition of the complex task may then be further broken down into subtasks. For example, in QBE, the user could start with the main task of completing a query on one table. This could be broken down into the subtasks of: selection of the table, the addition of attribute constraints, and the selection of attributes for output, concluding with the execution of the query. Each of these tasks could be further decomposed into subtasks. If we take the task of placing constraints on attributes of the table, this would involve selection of the query space for that attribute and entry of the query.

Figure 2 shows this simple task decomposition. Order in HTA can be specified in various ways, e.g. plans [37], and Jackson structured design (JSD) [20]. Figure 2 uses JSD to specify the order that the

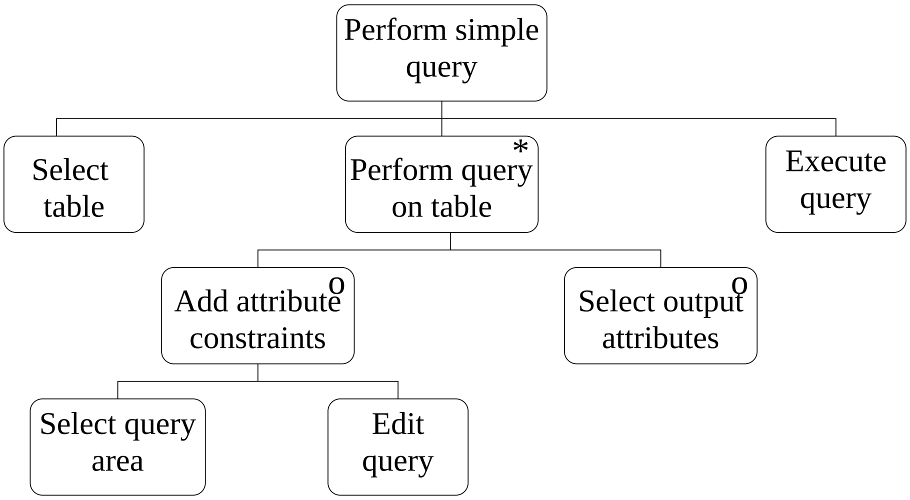


Figure 2: Example task hierarchy for a simple query in QBE

tasks are completed. The *Perform simple query* task is decomposed into three parts: select table, perform query on table and execute query, and are performed in sequence from left to right. That is, you cannot perform a query on a table until a table has been selected. Some of the tasks have either an asterisk or an 'o' in their top right hand corner. The asterisk represents iteration and the 'o' represents optional elements. This means that after a table has been selected, the tasks of adding constraints and selecting output attributes can be performed any number of times before the execution of the query.

The class of dialogues that can be represented using JSD in hierarchical task analysis are limited, but include many simple menu driven information systems. JSD will be used throughout the paper as it can adequately model the task hierarchies that will be presented. More complex dialogues can be represented by using plans as described by Shepherd [37].

From the analysis of Figure 2 we see that to perform a goal consists of some sequence of tasks. Each of these tasks can be decomposed into further subtasks, which either consists of another sequence of subtasks, given in definition 5 or will describe a user's interaction cycle, as given in definition 6.

Definition 5
Complex tasks can be represented as the interaction cycle tuple I=< GL,{ST}, RP> where:
GL is the goal or aim that the user is attempting to achieve.
{ST} is the ordered set of subtasks which need to be performed to accomplish the complex task.
RP is the resultant presentation on the completion of the complex task.

If a task is not a complex task it can be defined as consisting of one interaction cycle. This will consist of a goal, that defines the purpose of the task, the device through which the input is conveyed, and the data model concept selected by the input, with its associated presentation and layout. After selection of this data model concept some aspect of the display may change giving an intermediate result that may be visible to user.

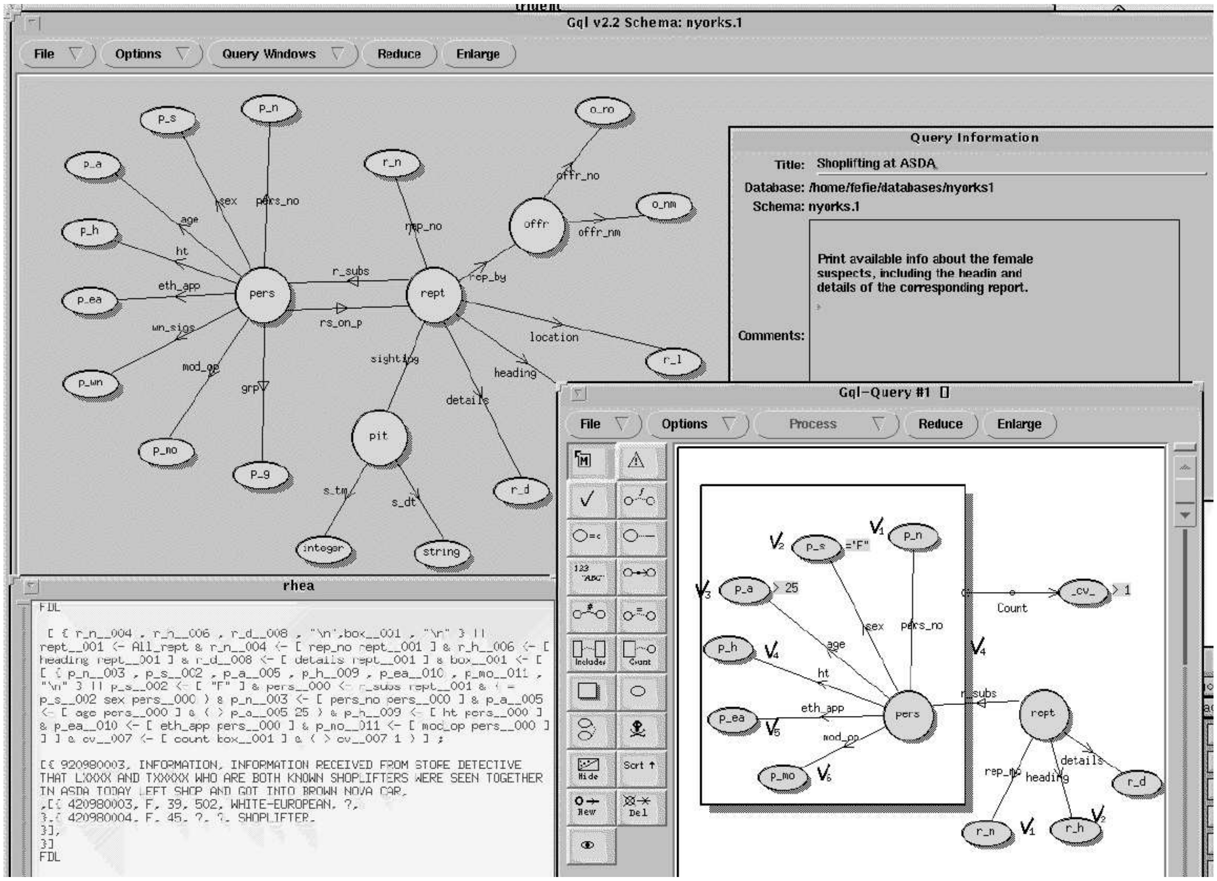
Definition 6
A single subtask can be represented as ST=< GL, D, DMC, L, P, IR> where:
GL is the goal or aim of the task.
D are the devices used to convey the input.
DMC is the data model concept/s selected by the user input.
L is the layout associated with the DMC.
P is the presentation associated with the DMC.
IR is the intermediate result from selecting the data model concept.

6 Case Studies

We shall now show how the framework can be used to describe some typical graphical database interfaces and the user's interactions with these interfaces. We have chosen to examine the Gql and QBE style of interfaces. We have chosen to compare these two interfaces as although they are both query interfaces, they exhibit different interface paradigms, thus allowing comparison of how the framework can be used to describe form based interfaces with QBE, and graphical/diagrammatic interfaces with Gql. With Gql it is also shown how textual displays are treated. Database activities other than querying are not examined here but these interfaces and tasks can be described in a similar vein, e.g. schema creation in a graphical interface can be similar to query building, etc.

6.1 Gql Database Interface Presentation

The Gql interface [27, 28], is a graph-based visual query language (see figure 3). This presentation consists of a graph representing the database schema, with entities represented as circles and their associated attributes represented by ovals, with the attribute name appearing inside the oval, and with functions represented by labelled directed arcs. Along with the database schema window there is the database query schema window, in which a query is constructed using operations represented as icons located on the toolbar. The results of a query are displayed in a window as text. Interaction is achieved through the use of a keyboard and a mouse.



Title: Shoplifting at ASDA
Database: /home/fefie/databases/nyorks1
Schema: nyorks.1
Comments: Print available info about the female suspects, including the headin and details of the corresponding report.

```
FDL

 [ { r_n__004 , r_h__006 , r_d__008 , "\n",box__001 , "\n" } ||
rept__001 <- All_rept & r_n__004 <- [ rep_no rept__001 ] & r_h__006 <- [
heading rept__001 ] & r_d__008 <- [ details rept__001 ] & box__001 <- [
[ { p_n__003 , p_s__002 , p_a__005 , p_h__009 , p_ea__010 , p_mo__011 ,
"\n" } || p_s__002 <- [ "F" ] & pers__000 <- r_subs rept__001 & ( =
p_s__002 sex pers__000 ) & p_n__003 <- [ pers_no pers__000 ] & p_a__005
<- [ age pers__000 ] & ( > p_a__005 25 ) & p_h__009 <- [ ht pers__000 ]
& p_ea__010 <- [ eth_app pers__000 ] & p_mo__011 <- [ mod_op pers__000 ]
] ] & cv__007 <- [ count box__001 ] & ( > cv__007 1 ) ] ;

[{ 920980003, INFORMATION, INFORMATION RECEIVED FROM STORE DETECTIVE
THAT LXXXX AND TXXXXX WHO ARE BOTH KNOWN SHOPLIFTERS WERE SEEN TOGETHER
IN ASDA TODAY LEFT SHOP AND GOT INTO BROWN NOVA CAR,
,[{ 420980003, F, 39, 502, WHITE-EUROPEAN, ?,
},{ 420980004, F, 45, ?, ?, SHOPLIFTER,
}],
}]
FDL
```

Figure 3: Gql session showing the schema, query and result window (from [27])

Figure 4 shows a Gql query to find all orders where more than 800 of each part have been ordered. The results are to include the quantity ordered, the part name, and the supplier name. The inclusion of a box around the *Order* and *Part* section of the query causes the nesting of the results. The query is constructed by selecting the entities and attributes that are to form the query from the database schema window. These are then placed in the query schema window and using the operations in the toolbar of the query schema window queries can be placed on the attributes.

We shall now show how the framework can be used to describe the presentation (the visualisation and spatial layout) of the interface. We shall then look at a subset of the tasks and describe interaction in Gql.

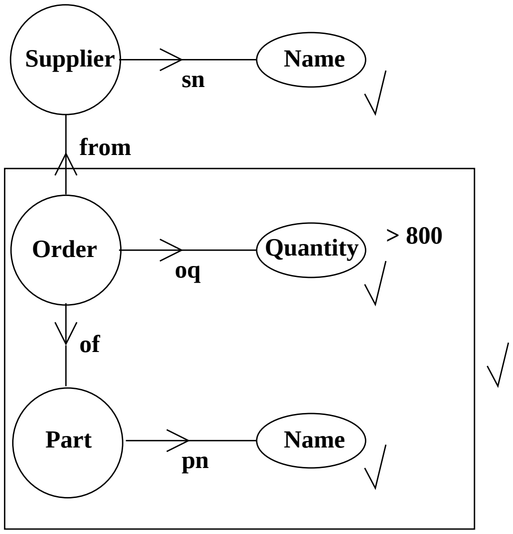


Figure 4: A Gql query

Definition 1 covered the database environment which consists of:

- Data Model (DM) - Gql utilises the functional data model, FDL [32]. The functional data model represents the data by entities and relations between them. There are two kinds of entities, those which relate to real world concepts (e.g. student, course, etc) and lexical entities (e.g. the string "Orange", the number 3.13, etc). The model also defines relations or functions over domains. For example, with the domains X and Y, a relation can be modelled with two functions, one from domain X to an element of domain Y and vice versa, with the option that these functions may also be multivalued.
- Presentation (P) - the presentation of the complete interface resides in a WIMP (windows, icon, menus, pointing devices) environment, described in Table 2.
- Tasks {T} - the main task supported by the interface is of querying the database. The decomposition of this task can be seen in Figure 5.

Definition 2 covered the database concepts and operations that are presented by the interface:

- Database Concepts (DC) - the database concepts that are visualised by Gql consist of the database schema (Table 3) the query schema (Table 4) and the query results (Table 5).
- Database Operations (DO) - the operations presented by the interface allow for the creation of the query schema, e.g. selecting results, performing arithmetic on attributes, etc. A selection of these are listed in Table 6.

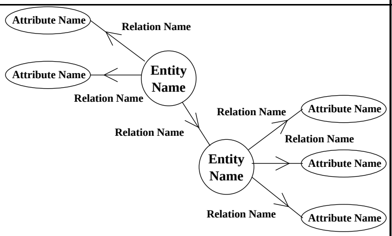
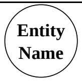
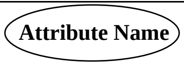
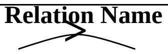
Definition 4 stated how a visualisation is composed of a concept with an associated visualisation and layout. It was also shown how a database concept could be composed of other database concepts (e.g. the database schema is a composition of entities and functions). Before explaining the database concepts we specify them in a table describing the overall system and the presentation and interaction devices that combine to create the overall system. The Gql database environment is composed of several database concepts and interaction and presentation devices, as outlined in Table 2. This table gives a high level view of the overall system and how it is presented in a WIMP environment. Along with the interface environment it lists the separate concepts that are displayed, in this case the database and query schemas, the available operations and the results environment. Each of these data concepts is defined in the associated table. Also listed in this table are the presentation and interaction devices that are available to the user of the interface. We shall now look at how the presentation of each of these data concepts is composed.

Interface	**Visualisation**	**Layout**
gql_int	WIMP environment	WIMP environment
Composition		
Data Model Concept	**Table**	
Database Schema	Table 3	
Query Schema	Table 4	
Query Results	Table 5	
Database Operations	Table 6	
Presentation and Interaction Devices		
Presentation Device	Monitor	
Interaction Devices	Table 7	

Table 2: Gql interface

6.1.1 Gql Database Schema

Each of the data concepts of FDL has an associated visualisation and layout which has to be defined within the framework. Table 3 shows how the database schema is composed. The table shows the name of the data concept, an example of a visualisation of the data concept and the spatial layout of the data concept, i.e. the database schema is placed in a window. The database schema is a composition of the data model concepts entity, attribute and relationship. Each of these has an associated visualisation and spatial layout defined. The layout of the relationship concept is defined as connected(entity, y, relationship), y ∈ {entity, relationship}. The connected function takes three arguments. The first two are the nodes of the relationship, and the final argument is the relation. In this example we can visualise two forms of relationship, either entity to entity relationships or entity to attribute relationships.

Data Model Concept	**Visualisation**	**Layout**
Database Schema		in(Database Schema, Database Schema Window)
Composition		
Data Model Concept	**Visualisation**	**Layout**
Entity		
Attribute		
Relationship		connected(,y,), y ∈ { , }

Attribute Name
Relation Name
Attribute Name
Entity Name
Relation Name
Relation Name
Relation Name
Attribute Name
Relation Name
Entity Name
Attribute Name
Relation Name
Attribute Name
Entity Name
Attribute Name
Relation Name
Entity Name
Relation Name
Entity Name
Attribute Name

Table 3: Gql database schema

In summary, table 3 defines there to be a database schema data concept which is presented in a window, where the database schema is composed of entities, attributes and relationships that can be connected by entity/entity or entity/attribute relations.

6.1.2 Gql Query Schema

The schema window gives a diagrammatic description of the database schema. The user selects items from the schema and copies them across to the query window to express a query. Table 4 shows the presentation, layout and composition of the database query schema. Again, this shows the name of the data concept, a sample visualisation and its spatial layout. This is then followed by a description of each of the data model concepts and how they can be visualised and laid out to create the database query schema. The connected function has already been described so we shall look at the other layout functions used, which are *beside*, and *in*.

For example, the selection data concept is an operation to mark an attribute, computed value, union box or collection box in order to include their output in the result. The spatial layout function *beside* accepts two arguments which define two concepts that will be located in close proximity to one another due to some relationship between them. The example beside(x,√), x∈{(Attribute Name), (_cv_) UNION, □}, as well as highlighting the spatial layout of the select operation is also specifying the set x which contains the valid concepts for the select operation. As well as allowing for the selection of attributes for output of the query, Gql also allows for the selection as output of computed values which are values derived from data in the database; the union box which is similar to "or"; and the collection box that allows for nesting of queries. Each of the boxes can be *ticked* to obtain the results of the query inside the boxes.

As another example of layout, the *in* function accepts two arguments. The first argument is the concept that will be placed inside the concept specified by the second argument. The union representation has placed within it collection boxes. The example in(□^{x}, UNION),x≥1 shows that if there is a union box present in the database query schema then it will contain only collection boxes, and there must be a minimum of one collection box within the union box.

The description of the layout of the query schema also makes use of the *grammar* operation. An example of this is given in the next section.

6.1.3 Gql Query Results

The results of a Gql query are returned as text in a separate window. Table 5 describes this presentation. A query returns an object's name and selected attributes which are visualised as text.

In Figure 4 we saw a Gql query to find all orders where more than 800 of each part had been ordered. The results included the quantity ordered, the part name, and the supplier name. The inclusion of a box around the *Order* and *Part* section of the query causes the nesting of the results:

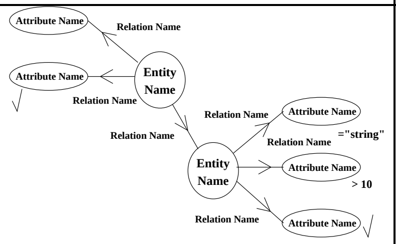
```
[{ Smith,
  [{ Nut, 900 }
   { Bolt, 1200 }
  ]
 }
 { Jones,
  [{ Nut, 850 }
  ]
 }
]
```

The grammar to describe the results follows:

```
Results        -> [ Attribute_List ]
               |  [ ]

Attribute_List -> { Attributes }
               |  { Attributes , [ Attribute_List ] }

Attributes     -> ATTRIBUTE_VALUE, Attributes
               |  ATTRIBUTE_VALUE
```

Data Model Concept	**Visualisation**	**Layout**
Query Schema	 Attribute Name Relation Name Attribute Name Entity Name Relation Name √ Relation Name Relation Name Attribute Name Relation Name ="string" Entity Name Attribute Name > 10 Relation Name Attribute Name √	in(Query Schema, Window)

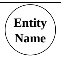
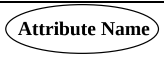
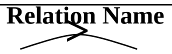
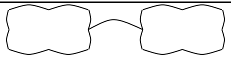
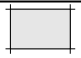
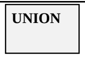
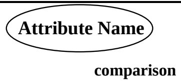
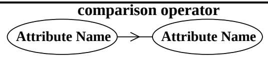
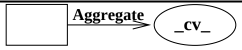
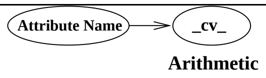
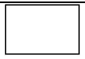
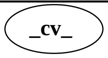
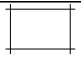
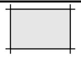
Composition		
Data Model Concept	**Visualisation**	**Layout**
Entity		
Attribute		
Relationship		connected(Entity Name,y, $\xrightarrow{\text{Relation Name}}$), y ∈ {Entity Name, Attribute Name}
Selection	√	beside(x,√), x∈{Attribute Name, _cv_, UNION, □}
And	not visualised	
x Or y		connected(in(x,☁), in(y,☁), ⌒)
Not x		in(x,▨)
Union		in(□ x, UNION),x>1
Attribute scalar comparisons		beside(grammar(comparison), Attribute Name) comparison ∈{ =, >, <, >=, <=, <>}
Attribute attribute comparison		connected(Attribute Name, Attribute Name, $\xrightarrow{\text{comparison operator}}$)
Aggregates∈{count, sum ,etc}		connected(□, _cv_, $\xrightarrow{\text{Aggregate}}$)
Arithmetic		connected(Attribute Name, beside(_cv_, grammar (arithmetic)), →)
Collection Box		in(query,□)
Computed Value		see Aggregates and Arithmetic
Exist		in(Entity Name, □)
Not Exist		in(Entity Name, ▨)

Table 4: Gql query schema

Data Model Concept	**Visualisation**	**Layout**
Query Results	text	in(grammar(Results), Results Window)

Composition		
Data Model Concept	**Visualisation**	**Layout**
Object Name	text	
Attribute Value	text	

Table 5: Gql query results

The first rule, *Results*, indicates that the results may be an empty list, [ ], or a list of attributes. The following rules further specify how the results are nested. The *ATTRIBUTE_VALUE* field comprises the actual values that are output. This grammar describes how the results are presented to the user, so they will not need to know this grammar, but they will need to understand how the presentation relates to the query.

6.1.4 Gql Database Operations

The database operations, Table 6, take the form of icons in a toolbar in the database query schema window (for reasons of brevity only a selection of the icons are shown in the table.) These operations allow the user to construct a query on the entities and attributes selected from the database schema.

Data Model Concept	**Visualisation**	**Layout**
Commands	icon buttons	in(Toolbar, Query Schema Window)

Composition		
Command	**Visualisation**	**Layout**
Exists	□	in(□, Toolbar)
Arithmetic	Attribute _cv_	in(Attribute _cv_, Toolbar)
Select	√	in(√, Toolbar)

Table 6: Gql database operations

6.1.5 Interaction

The previous sections defined how the visualisation and the spatial layout of the graphical interface to Gql can be represented using the framework. This section shows how the user's interaction can be described and visualised. As specified in Section 5, the first part of the interaction we need to specify is the interaction devices used by the interface. Table 7 names the interaction devices used by Gql, a keyboard and a mouse, and contains the representation of the devices used in the description of the user's interaction.

An Interaction Task Definitions 5 and 6 specified that a complex task was composed of subtasks. The complex task consists of a goal, some subtasks and a resulting presentation. The subtasks comprise each interaction cycle performed by the user. This involves the use of a device on an interface concept, that can result in an intermediate presentation and attainment of the subgoal.

The main task that the user can perform over this interface is to perform a query. This task breaks down into the selection of entities to include in the query, the application of various query operations to these entities, and execution of the query with the resultant presentation of the complex task taking the form of the output of the query. Figure 5 shows the part of the Gql task hierarchy that we will be examining. These cover some of the operations that are available to the user to perform a query.

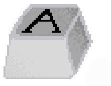
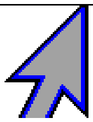
Device	Representation
Keyboard	
Mouse	

Table 7: Gql interaction devices

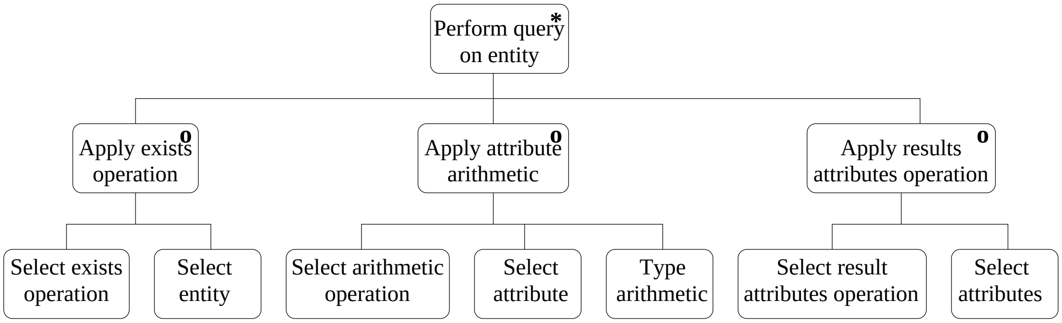


Figure 5: Part of the Gql task hierarchy

Again we have used JSD to specify the order in which the tasks are to be undertaken. In this task hierarchy we see that the main task shown, *Perform query on entity*, can be performed any number of times, with the option of performing either the exists, arithmetic or results selection task each time. We can now take these decomposed tasks and describe them via the interaction cycles as defined in definition 6.

The task of applying the *exists* operator in Gql is shown in Figure 6. This task is composed of two interaction cycles to perform the operation, first the operation is selected and then the entity to apply the operation to is chosen. The *exists* operation is used when the user is only concerned in the existence of an entity, e.g. to identify which lecturers have been allocated at least one course to teach. This example shows the selection of one entity for inclusion in the query. Further examples will highlight how iteration within a task can be accomplished.

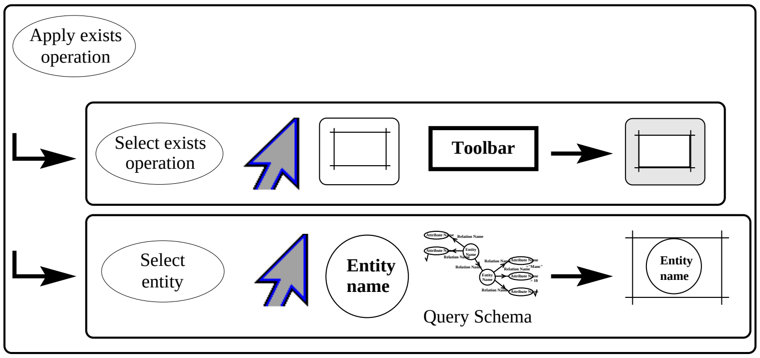


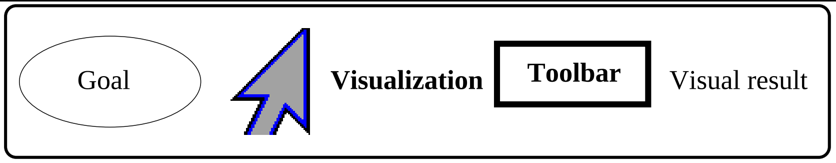
Figure 6: The Gql exists operation

Figure 6 describes the interaction involved in completing the *exists* operation. The complete complex task is contained in an oval box. The main task is formed of a goal presented in an oval, ⬭, and a list of the subtasks as they will be performed to achieve the goal. Each of these subtasks is also contained in an oval box preceded by an arrow, ↳. The user will go through a one or more interaction cycles/subtasks, attaining subgoals, until the complex task is completed. The interaction cycle in Figure 6 proceeds after the complex task goal by the user selecting and performing an action on a concept. Here we see that the user is performing the task of applying the exists operation to an entity in the query schema. The user begins by selecting the exists operation icon, □, from the toolbar using a mouse driven pointer. On completion of this subtask the user is informed of the result of their operation. The completion of a subtask is shown by the arrow, → followed by the visualisation that results from attainment of the subgoal. In this case it is shown by highlighting the toolbar icon, ▣. The interaction cycle proceeds with the user selecting an entity. This is performed by the the user selecting an entity, ○, from the database query schema with the pointing device, the completion of which causes an existence box to be placed around the entity that was selected, concluding the complex task.

This has shown a simple example of how the user's interaction can be specified, showing the user goals, interaction devices, and the interacting objects visualisation and layout. Further examples will highlight how arguments, text entry, and iteration are specified in the interaction.

Similar Tasks If we examine the first interaction cycle of the Gql exists operation in Figure 6 we see that this involves the user selecting one of the icons from the toolbar. As there are 16 icons in this toolbar, many tasks will begin in a similar fashion. To simplify this we can incorporate the first part of this task into a common task and specify the differences as arguments.

Select Toolbar Commands



Goal	Visualisation	Visual Result
Select results attributes operation	√	√
Select exists operation	□	▣
Select attribute arithmetic operation	Attribute _cv_	Attribute _cv_

Table 8: Selecting toolbar commands task in Gql

Table 8 shows the common task pattern where the goal, the icon to be selected and the visual result are specified as arguments to the common task. In performing this abstraction we have highlighted an area of consistency within the interface. With this table, the exists operation in Figure 6 can be specified by reference to Table 8 and Figure 7.

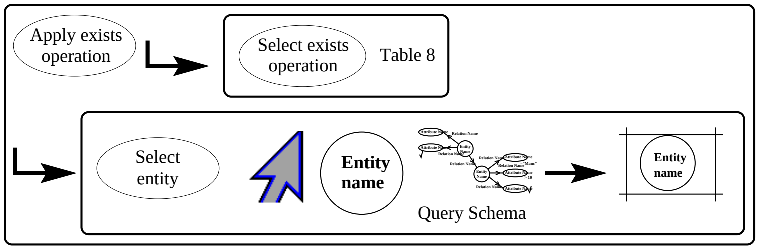


Figure 7: Gql exists operation

Interaction and Grammars Gql is intended as a visual language, with the user manipulating a graphical representation of the query, but some textual input via the keyboard is still required. Figure 8 shows the task of computing a new value by performing some arithmetic operation on an attribute in the database query schema.

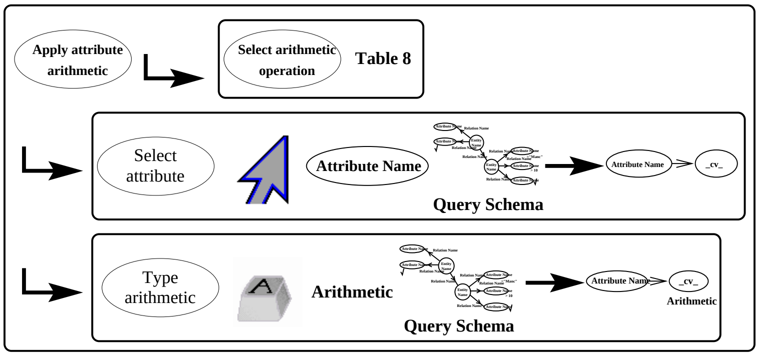


Figure 8: Gql arithmetic operation

First the user selects the appropriate operation icon, Attribute _cv_ as shown in Table 8. Then the user selects the attribute that they wish to use in the computed value operation from the database query schema. This results in the attribute having a computed value icon placed next to it. The final interaction cycle of the complex task involves a request for the user to type in an arithmetic expression involving the attribute previously selected. This expression has to comply with the grammar for the arithmetic. A simple example of what this grammar could be like follows:

```
Arithmetic  ->  Attribute_Name + Value
            |   Attribute_Name - Value
            |   Attribute_Name * Value
            |   Attribute_Name / Value
```

This grammar would allow the user to apply a single arithmetic operator to the attribute value, but the real grammar allows more complex expressions to be entered. However, it can be seen from this example that the user is requested to enter an arithmetic expression, and that without prior knowledge of the associated grammar and operations available, they may need to resort to some other method of enquiry as to what constitutes a valid entry, such as on-line help.

Interaction and Iteration Certain tasks may involve repetition of a subtask, e.g. the user may wish to select a number of objects for deletion which could be performed by selecting each item individually and then applying the delete operation to them. To specify iteration we use the symbols in Table 9.

Symbol	Meaning
[ ]	groups tasks
∗	task is performed 0 or more times
+	task is performed 1 or more times

Table 9: Symbols used in interaction

The [ ]s enclose a task or subtask that is to be repeated, with the * and + symbols indicating the number of iterations. Figure 9 shows the operation of selecting the results that are to be displayed by a query.

After the user has performed the task of selecting the icon, ☑, shown in Table 8, the user's next goal is to choose the required values to be displayed on execution of the query. With the pointing device the user can select items from the set x, i.e. an attribute, Attribute Name, a computed value, _cv_, a union box, UNION, or a collection box, □. The user can select any one of these with the resultant visualisation that a tick, √, is toggled, ±, from beside the selected item. By enclosing this task in [ ]'s and applying the ∗ operator to the enclosed task we can specify that the operation can be performed any number of times. We also see another way of presenting arguments to a task by including them as a set of possibilities for the task to choose from.

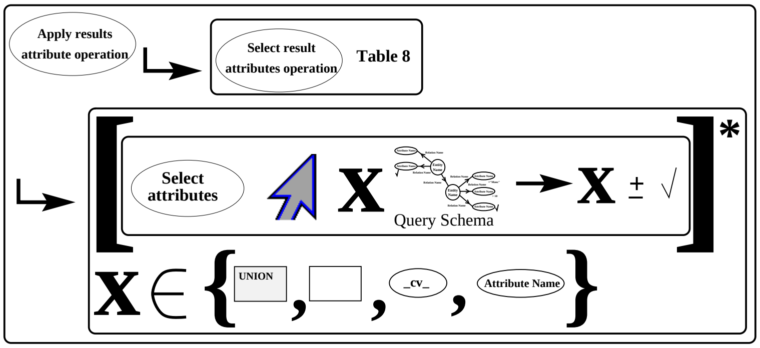


Figure 9: Gql select attributes for result

6.2 QBE Database Interface Presentation

Query By Example, QBE, is a forms based presentation for accessing relational databases [40]. We shall use the framework to describe the QBE interface as implemented in Paradox for Windows [4], with the interface residing in a WIMP environment, as shown in figure 10.

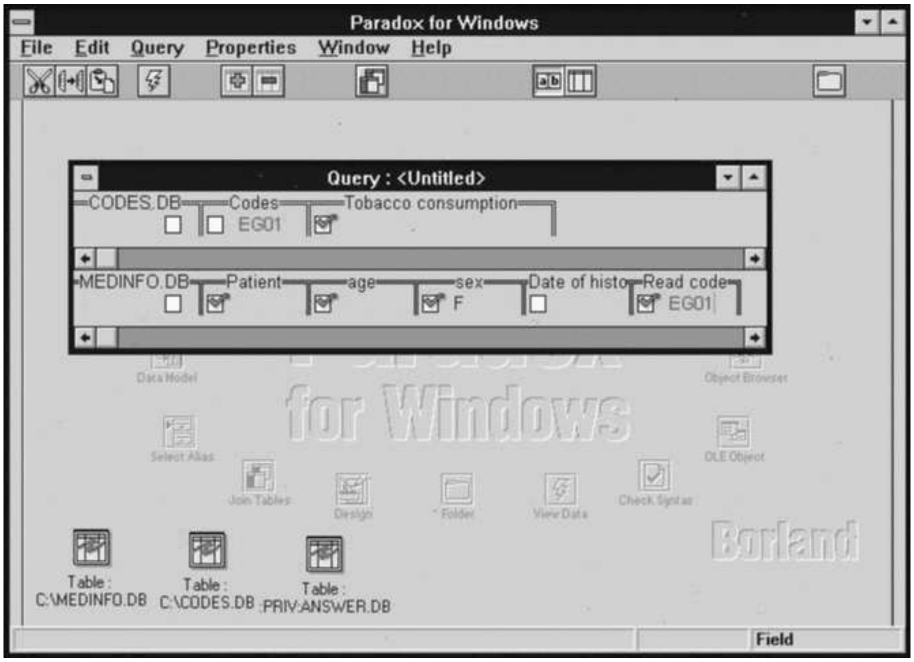


Figure 10: QBE interface

Definition 1 covers the database environment in QBE this consists of:

- Data Model (DM) - the interface resides on the relational data model.
- Presentation (P) - the presentation of the overall interface resides in a WIMP environment, see Table 10.
- Tasks {T} - the QBE interface supports the task of querying the database. The breakdown of this task can be seen in the task hierarchy of Figure 11.

Definition 2 defines the database concepts and operations that are presented by the interface.

- Database Concepts (DC) - the database concepts that are visualised in QBE consist of the query schema represented as tabular forms (Table 11) and the results displayed as text in a table (Table 12).

- Database Operations (DO) - the operations presented by the interface allow for query construction, including selection of forms, joining tables, executing queries, etc., these are shown in Table 13.

Queries in QBE are built by placing example values or filters in the tabular forms, with attributes to be output by the query having a tick placed by them. Table 10 gives a high level view of the composition of database concepts, operations and interaction devices that form the system, that we will now examine in more detail.

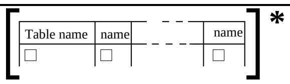
Interface	**Visualisation**	**Layout**
Paradox QBE	WIMP environment	WIMP environment

Composition	
Data Model Concept	**Table**
Query Schema	Table 11
Query Results	Table 12
Database commands	Table 13
Presentation and Interaction Devices	
Presentation Device	Monitor
Interaction Devices	Table 7

Table 10: QBE interface

6.2.1 QBE Query Schema

Table 11 defines the query schema form. The form is composed of row 1 comprising the table name, table(1, 1), and the attributes associated with the table, table(1, 2..n). The following rows in the form, table(2..n, 1..n), are composed of a selected attribute box into which the user can toggle a tick, √, to show inclusion of the attribute in the results. To the right of the selected attribute box there is a statement area that is not immediately apparent, as when unselected it has no apparent visualisation. The statement area allows the user to enter some search criteria.

Data Model Concept	**Visualisation**	**Layout**
Query Schema		in(Query Schema, QBE Window)

Composition		
Data Model Concept	**Visualisation**	**Layout**
Table Name	text	table(1, 1)
Attribute Name	text	table(1, 2..n)
Selected Attributes	□	table(2..n, 1..n)
Statement Area		table(2.n, 1..n)
Search criteria	text	in(statement area, grammar(search condition))

Table 11: QBE query schema

6.2.2 QBE Query Results

Table 12 describes how the results of a QBE query are presented. The database concept is the results which are visualised as a table, , with the layout component specifying that the table is placed in a window. The information contained within the table is composed of the selected attribute name, visualised as text and placed along the first row of the table, table(1, 1..n). The related attribute values are placed in the following rows, presented as text, table(2..n, 1..n).

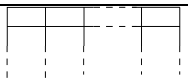
Data Model Concept	Visualisation	Layout
Results		in(Results, Window)
Composition		
Data Model Concept	**Visualisation**	**Layout**
Attribute Name	text	table(1, 1..n)
Attribute Value	text	table(2..n, 1..n)

Table 12: Description of results presented in a table

6.2.3 QBE Database Operations

The database operations, are visualised as icons on a toolbar in the main QBE window. Table 13 highlights a selection of these that are used in describing the user's interaction.

Data Model Concept	Visualisation	Layout
Commands	icon buttons	in(Toolbar, QBE Window)
Composition		
Command	**Visualisation**	**Layout**
Add Table	⊞+	in(⊞+, Toolbar)
Remove Table	⊞−	in(⊞−, Toolbar)
Join Tables	⊞	in(⊞, Toolbar)
Run Query	⚡	in(⚡, Toolbar)

Table 13: QBE database operations

6.2.4 QBE Interaction

In Figure 11 we see a decomposition of some of the subtasks of the complex task of performing a query. The user can first perform the *Build query* task any number of times before executing their query. The *Build query* task allows them to select or deselect the tables they wish to query and also to create filters, join tables and select the attributes they wish to see in the results. Each of the subtasks in the hierarchy will now be described.

Paradox's version of QBE utilises the same interaction devices as Gql, namely a mouse and a keyboard, as shown in table 7. Table 14 highlights the similar subtask of selecting an icon from a toolbar. The first example shows the complete task of executing a query by the user selecting the *run query* icon, ⚡, which concludes with the display of the result presentation, as described in Table 12.

Figure 12 extends coverage of the second and third operations in Table 14, where the user has selected either the *add table* icon, ⊞+, or the *remove table* icon, ⊞−. After the user has selected either icon, they are confronted with a list of tables from which they can select 0 or more tables. As they select a table from the list the highlighting of the table name is toggled. On completion of the task the selected tables are added to or subtracted from the QBE window.

The final row in Table 14 allows the user to join tables. On selecting the *join table* icon, ⊞, the user's cursor has *join* annotated to it, ↖Join. The user then selects a statement area from two tables to create a join, which results in the term *join-x* being placed in the statement area, where x is some number. On completion of these two join interaction cycles the cursor loses the join annotation.

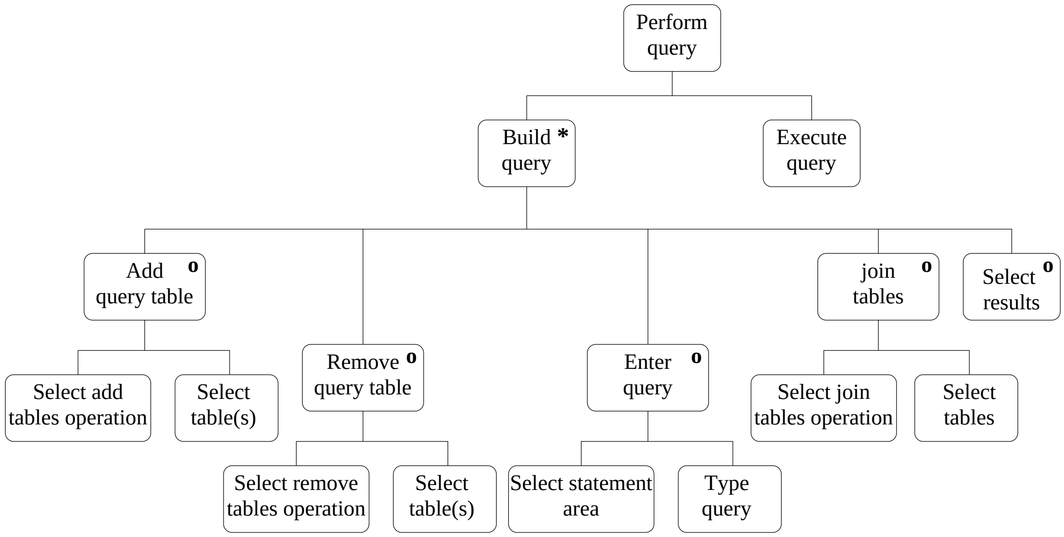
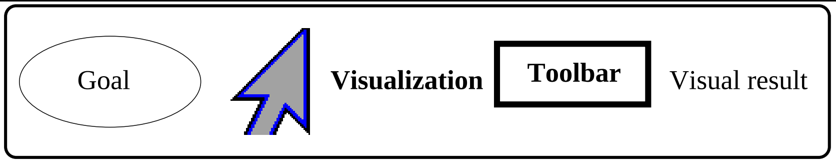


Figure 11: Part of the QBE task hierarchy

Select Toolbar Commands



Goal	Visualisation	Visual Result
Execute query	⚡	Query Results
Select add table operation	⊞+	Table List
Select remove table operation	⊞−	Table List
Select join tables operation	⊞	↖ **Join**

Table 14: Selecting toolbar commands task in QBE

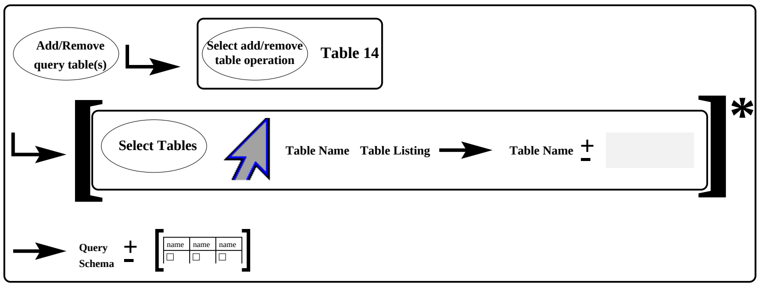


Figure 12: QBE select/deselect table(s) for query

Figure 14 shows how the user selects the attributes to be included in the result. Using the pointer, the *select attribute* box, □, which appears in the query schema, can be chosen. This will toggle a tick, √ in the box.

Figure 15 describes the task of entering search criteria using comparison and logical operators applied

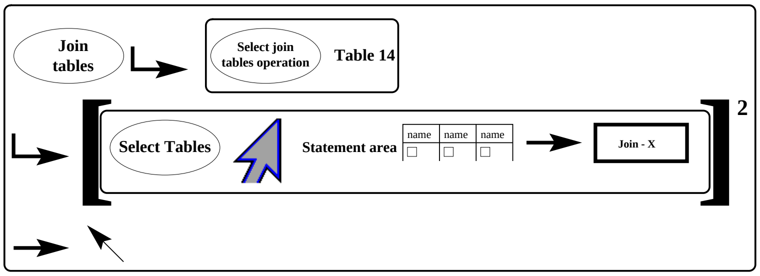


Figure 13: QBE join tables

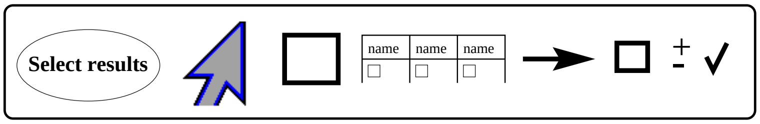


Figure 14: QBE select/deselect attribute for query

to attribute values. We can see from the figure that the user selects the statement area from the query schema using the pointing device. This results in the statement area being highlighted and ready to accept the user's input via the keyboard. The user then enters a *query grammar construct*. This corresponds to a query that is correct according to a specified grammar. The grammar required to describe the query typed into the statement area is large when compared to the grammar required in Gql but is obviously smaller than that required to describe a language such as SQL. As stated previously, the more a user interface makes use of a grammar to describe its interaction, the more knowledge the user must have or remember about the system.

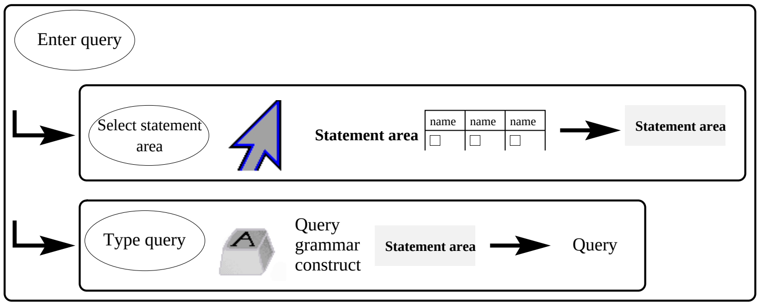


Figure 15: QBE select statement and edit query conditions area

In this section we have seen how the database interfaces of Gql and QBE can be described using the framework. Using the framework we first described the database environment, defining the data model used, the database concepts and operations that are presented and the hardware devices used for interaction. Each of the data model concepts was then described showing the composition, visualisation and layout of the concepts. Hierarchical task models are used to describe the structure of the tasks, followed by a complete description of the tasks showing the composition of a task, the users goals and feedback and the visualisation and spatial layout of the components used in the task. Only a section of the Gql task hierarchy has been defined but an almost complete description of the QBE interface and the task hierarchy has been defined. We shall now compare this framework with some other notations that are available.

7 Related Work

There exist many formalisms in HCI that attempt to predict, analyse and describe properties of interactive systems. A recent taxonomy of this area [6] has identified evaluation criteria for these present formalisms. Three of the twelve evaluation criteria most pertinent to this framework examine how well the formalisms represent:

1. the user's tasks and actions;
2. the interface state and system feedback;
3. the presentation of the interface.

Table 15 presents an evaluation of some formalisms with regards to the above three criteria. An evaluation is given on a scale of support ranging from none, through to minimal, partial and extensive.

Area	Formalism	Tasks and Actions	Interface State and Feedback	Presentation
Calculus Theory	Petri Nets	none	partial	none
	STCs	none	partial	none
	OSTCs	none	partial	partial
Cognitive Science	TAL	extensive	none	none
	GOMS	extensive	none	none
	CCT	extensive	none	none
	TAG	extensive	none	none
	CLG	extensive	minimal	none
	UAN	extensive	extensive	minimal
Databases	Haber et al	none	none	partial
	Murray et al	extensive	extensive	extensive

Table 15: Evaluation of existing formalisms

Petri Nets [30], State Transition Charts (STCs), [25, 15] and Object State Transition Charts (OSTCs), [26], from Calculus Theory, can be described as constructional (relating to how the interface is implemented) rather than behavioural (relating to how users accomplish their tasks). They don't describe the system from the user's viewpoint but highlight state changes within the system, e.g. the system is in state X; Y is input so the system changes to state Z. OSTCs use state transition charts to define the input to interface objects and so partially support the presentation of the system, but this is still a system's rather than a user's view. Furthermore, these formalisms do not support the state, feedback and presentation of the interface as can be seen in Table 15.

From Cognitive Science we begin with Task Action Language (TAL) [34]. TAL uses a formal grammar for describing the user's interaction with a system to support evaluation of its design. The grammar is used to describe the user's interaction (e.g. mouse movements, typing, etc.), forming an action language for the interactive system. Using grammars from different designs, comparisons can be made on the basis of the grammar's simplicity and consistency on the assumption that the cognitive factors of learning and remembering are related to these factors [16, 35]. TAL is therefore an attempt to perform predictive analyses on different interface designs. It does address the issue of describing the user's interaction with the system but does not cater for our need to specify feedback and presentation.

Goals, Operators, Methods and Selection Rules (GOMS) [8], Cognitive Complexity Theory (CCT) [31], and Task Action Grammar (TAG) [29], all follow on from TAL in that they attempt to perform predictive or competence analysis of interface design specifications by using more psychologically valid models than TAL, but these models tend to be piecemeal, using only partial cognitive psychology theories [3]. Each represents the user's tasks and actions but has no need to describe the feedback and presentation of the design as they are attempting to perform predictive analysis of the designs.

Descriptive formalisms have also emerged from the area of Cognitive Science. Command Language Grammar (CLG) [24], can be considered a descriptive formalism as it does not include any characteristics of the user or any metrics or criteria for performing predictive analysis on the design. CLG is a linguistic dialogue model, so it can only be used to describe command language interfaces. It does give a representation for describing the user's interaction but does not support descriptions of the interface presentation and feedback as the physical component of CLG has not been defined.

User Action Notation's (UAN) [38, 18] main role is as a descriptive tool. UAN tasks are described in a table showing the user actions, the interface feedback and state. Table 16, from [18] describes the task of selecting a file icon with a mouse in a desktop environment.

User actions	*Interface feedback*	*Interface State*
∼[file_icon] M∨	file_icon-!: file_icon!, ∀ file_icon'!: file_icon'-!	selected = file
M∧		

Table 16: Selecting a file icon in UAN

The user action " ∼[file_icon]" means "move the mouse pointer over the file icon to be selected" with "M∨" meaning "press the mouse button". When this action occurs the feedback received is that the icon is highlighted if it is not already highlighted - "file_icon-!: file_icon!". As a consequence of this action, all other icons that were previously highlighted are de-highlighted - "∀ file_icon'!: file_icon'-!". The variable "selected" is then set to "file". "M∧" is the notation for releasing the mouse button and completes the task. UAN does support the user's interaction and interface state and feedback, but this is done textually, so support for interface presentation is minimal.

From the area of databases, Haber et al [17] defines a visual metaphor as being a mapping between the data model and a visual model. In this respect it is very similar to Definition 4 of the framework but goes into more depth regarding the use of metaphors and their quality. With this framework, we are not interested in evaluating the presentation of each individual database concepts, but in obtaining an overall description of the system and thereby evaluating the presentation of the database concepts as they are utilised by the user as they perform their tasks. Frameworks such as Haber's allow for the evaluation of different visualisations and metaphors for schemas to provide richer displays.

The framework presented here supports an emphasis on the description of presentation and interaction within interfaces that is different from that supported by other formalisms. The framework addresses the description of tasks, feedback and presentation in a way that supports the description of user interfaces to databases more thoroughly than earlier proposals. This is not particularly a criticism of the other approaches mentioned here – the specific role of our framework, namely the systematic description of existing or proposed database interfaces, has not received much attention in the past.

8 Conclusion

The tendency for user interfaces to databases to be described in an informal manner has motivated the development of a framework for describing visual interfaces to databases. The framework presented in this paper describes interfaces from a user's perspective, focusing on support for the construction of concise, clear descriptions of interaction and presentation issues, rather than on the semantics of operations on the underlying database.

The framework is intended to be used in two principal contexts:

1. The description of existing interfaces, with a view to giving consistent coverage of interface features, thereby providing better understanding of such systems and allowing straightforward comparisons to be made.

   This role has been illustrated in the paper through the description of two visual query interfaces, namely Gql and QBE. These interfaces were chosen as they seemed representative of visual database interfaces, but they were also unusual in that the published specifications were clear enough to allow the framework to be applied to their description. Many published descriptions contained ambiguities and omissions, which were identified when attempts were made to describe them using the framework.

2. The design of new interfaces, with a view to prompting designers on the principal issues to be addressed, and to providing the systematic documentation of designs as the software development process proceeds. Designs within the framework can be used to guide the development of mock-up, prototype and production level interfaces.

The framework is different from earlier proposals for describing user interfaces, in that it aims to be non-textual, and addresses all three areas of tasks and actions, interface state and feedback, and presentation, that can be identified in the interaction cycle.

The framework is currently being used to prototype a 3D query language for object databases. We have found that producing descriptions of the language using the framework allowed for easier communication of ideas in a consistent manner as opposed to the imprecision of textual descriptions. This advantage far outweighed the slight increase in time to create the descriptions although this is comparable with using any of the frameworks and formalisms discussed in Section 7.

References

[1] Gregory D. Abowd and Russell Beale. Users, Systems and Interfaces: A Unifying Framework for Interaction. In *People and Computers VI, Proceedings of the HCI '91 Conference*, pages 73–87, 1991.

[2] J. Annett and K. Duncan. Task Analysis and Training Design. *Occupational Psychology*, 41:211–221, 1967.

[3] Phillip J. Barnard. Cognitive resources and the learning of human-computer dialogs. In John M. Carroll, editor, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, chapter 6, pages 112–158. MIT Press, London, England, 1987.

[4] Borland. *Borland Paradox for Windows User's Guide.* Borland International Inc., 1994.

[5] J. Boyle, S. Leishman, and P. M. D. Gray. From WIMP to 3D: the development of AMAZE. *Journal of Visual Languages and Computing*, 7:291–319, 1996.

[6] Philippe Brun and Michel Beaudouin-Lafon. A Taxonomy and Evaluation of Formalisms for the Specification of Interactive Systems. In *People and computers X : Proceedings of HCI '95*, pages 197–212. Cambridge University Press, 1995.

[7] Michael Caplinger. Graphical Database Browsing. In *Proc. 3rd ACM SIGOIS*, pages 113–121, 1986.

[8] S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale NJ, 1983.

[9] T. Catarci, S. K Chang, and G. Santucci. Query Representation and Management in a Multiparadigmatic Visual Query Environment. *Journal of Intelligent Information Systems*, 3(3):299–330, 1994.

[10] R. L. Cooper, editor. *Proc. 1st Int. Workshop on Interfaces to Database Systems.* Springer-Verlag, 1992.

[11] C. J. Date. *A Guide to the SQL Standard.* Addison-Wesley, 1987.

[12] D. Diaper. Task Observation for human-computer interaction. In D. Diaper, editor, *Task Analysis for Human-Computer Interaction.* Chichester, Ellis Horwood, 1989.

[13] Alan Dix, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction.* Prentice Hall, 1993.

[14] D. K. Doan, N. W. Paton, A. C. Kilgour, and G. Al-Qaimari. A Multi-Paradigm Query Interface To An Object-Oriented Database. *Interacting With Computers*, 7(1):25–47, 1995.

[15] D. C. Engelbart and W. K. English. A Research Center for Augmenting Human Intellect. In *AFIPS Conference Proceedings, 1968 Spring Joint Conference, Washington, DC.* Thompson Books, 1968.

[16] Jonathon Grudin. The Case Against User Interface Consistency. *Communications of the ACM*, 32(10):1164–1173, October 1989.

[17] Eben M. Haber, Yannis E. Ioannidis, and Miron Livny. Foundations of Visual Metaphors for Schema Display. *Journal of Intelligent Information Systems*, 3(3):263–298, 1994.

[18] H. Rex Hartson, Antonio C. Siochi, and Deborah Hix. The UAN: A User-Oriented Representation for Direct Manipulation Interface Designs. *ACM Transactions on Information Systems*, 8(3):181–203, 1990.

[19] David Haw, Carole Goble, and Alan Rector. GUIDANCE: Making it Easy for the User to be an Expert. In *Proc. 2nd Int. Workshop On Interfaces to Database Systems*, pages 19–43. Springer-Verlag, 1994. P. Sawyer (Ed).

[20] M. A. Jackson. *System Development.* London: Prentice Hall, 1983.

[21] J. B. Kennedy and P. J. Barclay, editors. *Proc. 3rd Int. Workshop On Interfaces to Database Systems.* Springer-Verlag, July 1996.

[22] E Knuth and L. M. Wegner, editors. *Visual Database Systems II.* North-Holland, 1992.

[23] T. L. Kunii, editor. *Visual Database Systems.* North-Holland, 1989.

[24] Thomas P. Moran. The Command Language Grammar: A Representation for the User Interface oa Interactive Computer Systems. *International Journal of Man-Machine Studies*, 15:3–50, 1981.

[25] W. M. Newman. A System for Interactive Graphical Programming. In *AFIPS Conference Proceedings, 1968 Spring Joint Conference, Washington, DC.* Thompson Books, 1968.

[26] William M. Newman and Michael G. Lamming. *Interactive System Design.* Addison-Wesley, 1995.

[27] A. Papantonakis and P. J. H. King. Syntax and Semantics of Gql, a Graphical Query Language. *Journal of Visual Languages and Computing*, 6:3–25, 1995.

[28] Anthony Papantonakis and Peter J. H. King. Gql, A Declarative Graphical Query Language based on the Functional Data Model. In S. Levialdi T. Catarci, M. Costabile and G. Santucci, editors, *Proceedings Advanced Visual Interfaces*, pages 113–122, 1994.

[29] S. J. Payne and T. R. G. Green. Task-Action Grammars: A Model of the Mental Representation of Task Languages. *Human Computer Interaction*, 2:93–133, 1986.

[30] J. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3):223–252, 1977.

[31] Peter G. Polson. A quantitative theory of human-computer interaction. In John M. Carroll, editor, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, chapter 8, pages 184–235. MIT Press, London, England, 1987.

[32] Alexandra Poulovassilis and Peter King. Extending the Functional Data Model to Computational Completeness. In *Advances in Database Technology - EDBT'90 International Conference on Extending Database Technology*, pages 75–91, 1990.

[33] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey. *Human-Computer Interaction.* Addison-Wesley, 1994.

[34] Phyllis Reisner. Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Transactions on Software Engineering*, 5(2):229–240, 1981.

[35] Phyllis Reisner. What is Inconsistency. In *INTERACT'90: Proceedings IFIP 3rd International Conference on Human-Computer Interaction*, pages 175–181, 1990.

[36] P. Sawyer, editor. *Proc. 2nd Int. Workshop on Interfaces to Database Systems.* Springer-Verlag, 1994.

[37] A. Shepherd. Analysis and training in information tasks. In Dan Diaper, editor, *Task Analysis for Human-Computer Interaction.* Ellis Horwood, Chichester, 1989.

[38] Antonio C. Siochi and H. Rex Hartson. Task-Oriented Representation of Asynchronous User Interfaces. In *Proceedings CHI '89 Human Factors in Computing Systems*, pages 183–188, 1989.

[39] K. Tsuda, M. Hirakawa, M. Tanaka, and T. Ichikawa. Iconic Browser: An Iconic Retrieval System for Object-Oriented Databases. *Journal of Visual Languages and Computing*, 1(1):59–76, 1990.

[40] M. Zloof. Query-By-Example: A Data Base Language. *IBM Systems Journal, Vol. 4*, pages 324–343, December 1977.