A VISUAL SHELL INTERFACE TO A DATABASE

by

Lawrence A. Rowe, Peter Danzig, and Wilson Choi

Memorandum No. UCB/ERL M87/2

13 January 1987

A VISUAL SHELL INTERFACE TO A DATABASE

by

Lawrence A. Rowe, Peter Danzig, and Wilson Choi

Memorandum No. UCB/ERL M87/2

13 January 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

A VISUAL SHELL INTERFACE TO A DATABASE

by

Lawrence A. Rowe, Peter Danzig, and Wilson Choi

Memorandum No. UCB/ERL M87/2

13 January 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A Visual Shell Interface to a Database [†]

Lawrence A. Rowe, Peter Danzig, and Wilson Choi

Computer Science Division - EECS
University of California
Berkeley, CA 94720

Abstract

This paper describes a visual command language for a workstation with a bit-mapped display and a mouse that can be used to create different user-interfaces. Primitive interface components can be combined into more complex user-interfaces. The user specifies interconnections between these components over which data and commands can be sent by pointing with a mouse.

Primitive interface components are described for creating several different database user-interfaces. The design and implementation of the software architecture is described including the primitives for database interfaces and the communication protocols used by the system.

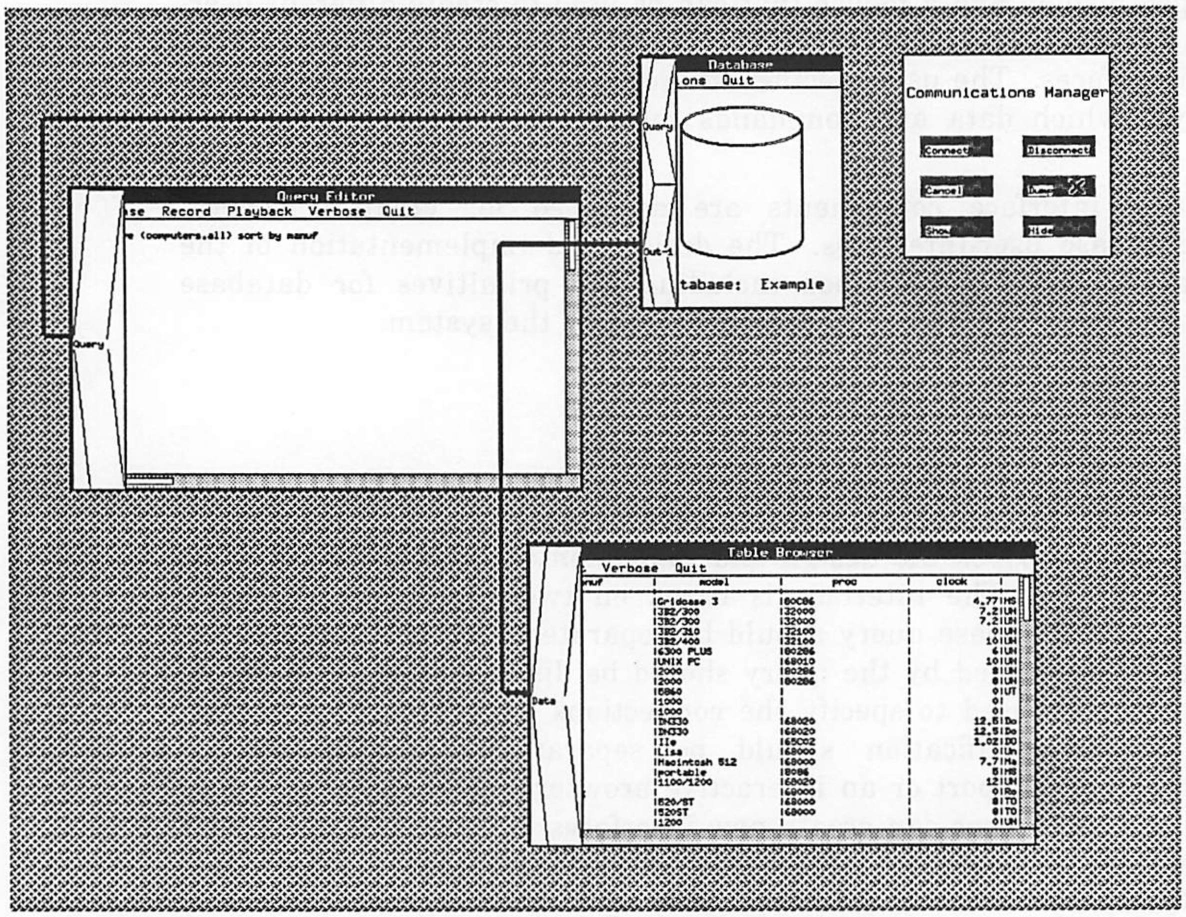
1. Introduction

This paper describes the design and implementation of a visual interface to a database. The interface is based on two simple ideas: 1) the specification of a database query should be separated from the specification of how the data specified by the query should be displayed and 2) a visual interface should be used to specify the connections between interface components. Query specification should be separated from the display specification (e.g., a report or an interactive browser that displays the data in a form) so that a user can create new interfaces by connecting different query specifiers to different data displayers. For example, a user could create a Query-By-Forms interface [RTI84] by connecting a forms-based query specifier to a browser that displayed data through a form or he could create a Query-By-Example [Zlo75] interface by connecting a visual,

[†] This research was sponsored by the National Science Foundation under grant DCR-8507256. The second author was also supported by an American Electronics Association Faculty Development Fellowship.

multiple relation query specifier to a displayer that prints the data in a tabular format. A visual interface is used to specify the connection of a query specifier to a data displayer because it simplifies the user-interface.

The interface command language, called the *Visual Shell* (VS), allows the inputs and outputs of the interface components, called *tools*, to be dynamically interconnected to form different user-interfaces. The system runs on a workstation with a bit-mapped display and a mouse. Figure 1 shows an actual screen dump with three tools connected together to form a simple database browser. The tools are: a query editor, a table browser, and a database. The query editor allows the user to compose and edit queries



```
e (computers.all) sort by manuf
```

```
nuf   |model          |proc      |        clock |
      |Gridcase 3     |80C86     |        4.77|MS
      |3B2/300        |32000     |         7.2|UN
      |3B2/300        |32000     |         7.2|UN
      |3B2/310        |32100     |           0|UN
      |3B2/400        |32100     |          10|UN
      |6300 PLUS      |80286     |           6|UN
      |UNIX PC        |68010     |          10|UN
      |2000           |80286     |           8|UN
      |2000           |80286     |           8|UN
      |5860           |          |           0|UT
      |1000           |          |           0|
      |1000           |          |           0|
      |DN330          |68020     |        12.5|Do
      |DN330          |68020     |        12.5|Do
      |IIe            |65C02     |        1.02|DO
      |Lisa           |68000     |           0|Un
      |Macintosh 512  |68000     |         7.7|Ma
      |portable       |8086      |           5|MS
      |1100/1200      |68020     |          12|UN
      |               |68000     |           0|Ro
      |520/ST         |68000     |           8|TO
      |520ST          |68000     |           8|TO
      |1200           |          |           0|UN
```

Figure 1: A simple database interface.

written in a textual query language such as QUEL. The table browser allows the user to view or update data through a table. And, the database tool mediates between the other tools and the database.

Each tool has a window through which it interacts with the user and a collection of input and output ports that can be connected to ports in other tools. Data and commands are passed between tools over these connections. Tool connections are shown in the figure as dark lines connecting tool ports. Each port has a name which is shown in a window on the left edge of the tool. For example, the *Query* port in the query editor tool is connected to the *Query* port in the database tool. Ports and connections are normally hidden from view unless the user wants to change the connections. When they are hidden, the user can enter a query into the query editor tool and execute a menu operation that causes the query to be executed and the results to be displayed through the table browser tool. Using the table browser, the user examine or update the data.

This interface was created by executing the tool programs and specifying the connections between them. Tool programs are executed by entering a command to the command interpreter or by selecting the program from a menu. The user establishes a connection between two tools by selecting the two ports and executing the *Connect* operation in the communications manager tool (CM) shown in the upper right corner of the screen. Ports and operations are selected by positioning the mouse cursor on the desired symbol and pressing a mouse button.

The novel feature of this database interface is that different tools can be arbitrarily connected to create different user-interfaces. Conventional database interfaces support different query specification interfaces (e.g., textual, forms-based, or graphical [McS75,MET8?]) and a variety of display interfaces (e.g., forms, reports, graphs, and spreadsheets) but they limit the interconnection of these interfaces to only a few alternatives (e.g., QBF and QBE). Some systems allow different specifications for input and output (e.g., QMF allows queries to be specified in SQL or through a QBE query interface and reports to be specified through forms or in a textual language [IBM83]) but this flexibility is limited to a few specific cases. Other database interfaces that run on workstations support interesting query and display formats but, they either rigidly proscribe how the tools are interconnected or they require the user to program a new interface in a programming language [Cat84,FrE86,Goe85,Her80,StK82,The83].

VS was motivated by the *Data-Flow Manager* (DFM) developed by Haeberli for the IRIS workstation [Hae86]. Although the two systems are very similar, VS differs from DFM in several important respects. First, VS connections are bidirectional. The number of connections required in our database application is cut in half by the use of bidirectional connections. Several database tools discussed below synchronize by asking for and then receiving data from another tool. Twice as many connections would have been required if a one-directional connection had been used as in DFM which would complicate the user's model of the connection abstraction. Second, VS allows tools to create and destroy ports dynamically whereas DFM required tools to define all ports at compile time. The database tools we developed use dynamic port creation extensively to allow users to create interfaces with a variable number of output displayers. Third, VS uses the Fourth Berkeley Software Distribution (4BSD) socket mechanism to implement connections [LJF83] whereas DFM used shared files which required a kernel modification. Using sockets permits tools running on different hosts in a network to be connected together. Lastly, Haeberli developed tools for interactive graphics applications (e.g., a graphical object viewer, a slider for specifying view transformations, and a solid object viewer) while we have developed tools for interacting with a relational database.

VS is closely related to work on operating system command languages, most notably the Unix† shell language [Bou78]. The shell language allows simple programs (i.e., tools) to be combined in a flexible way to create more complex programs. Each tool has one input port and one output port that can be connected together to create more complex programs. This composition is referred to as *piping* the output of one program into the input of another program. In addition, the shell allows input and output to be redirected to files or I/O devices. The shell language is a textual command language that allows these complex programs to be created without writing a program in a programming language. For example, the following command

```
bib -tstda paper.n | tbl | ditroff -Pip -me -
```

was used to look up paper references in a database, format, and print this paper. The pipe operator ('|') signifies that the output of the left operand should be connected to the input of the right operand.

VS differs from the Unix shell in three ways. First, complex programs created with the shell language are almost always linear pipelines because most programs have only two ports and the command language encourages linear constructions. This limitation is not inherent in the system but, the vast majority of pipelines are linear. VS encourages users to develop tools with many ports because of the two dimensional notation for specifying connections. Second, Unix does not provide a mechanism to dynamically add and remove ports and to change connections while a program is running. VS provides both functions and, in fact, encourages users to construct new interfaces dynamically. Lastly, the Unix shell language was designed for an alphanumeric terminal and as a result does not support program

† Unix is a trademark of A.T.&T.

interaction with the user through a window with a mouse. VS, on the other hand, is inherently a visual interface. Other researchers have explored extensions to the Unix shell that incorporate more complex connections, but they have not supported dynamic connection changes or visual interfaces [Shu83].

The remainder of this paper describes the design and implementation of the database interface tools and VS. Section 2 shows several different user-interfaces constructed with the primitive database interface tools. Section 3 describes the design and implementation of the database primitives and the protocol used between them. Section 4 describes the implementation of the CM and the library used by tool programs. Section 5 discusses extensions to the current system and section 6 summarizes the paper.

2. VS Examples

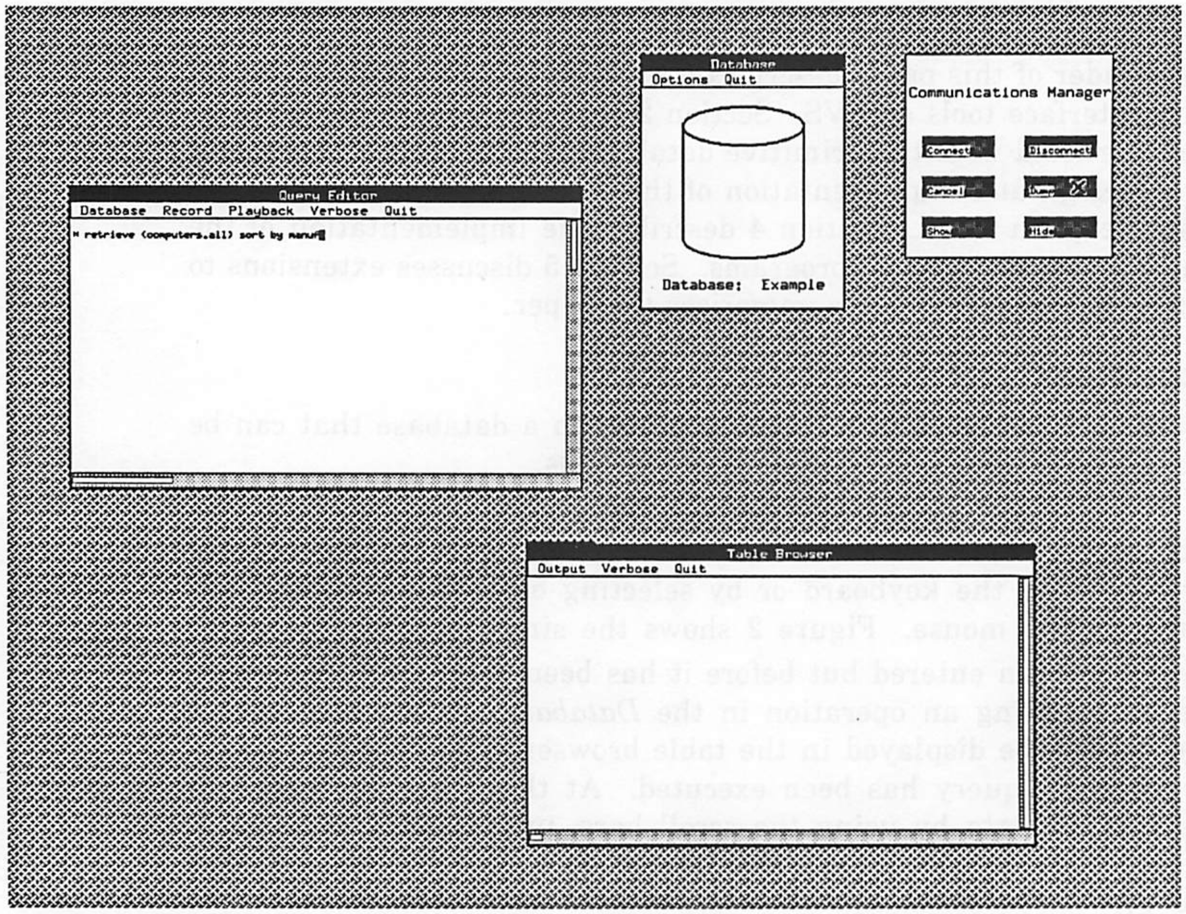
This section describes several user-interfaces to a database that can be constructed with VS and the primitive database tools.

After creating and connecting the tools, the user interacts with the interface by issuing commands to the individual tools. Commands are entered by typing on the keyboard or by selecting operations from a pull-down menu with the mouse. Figure 2 shows the simple database browser after a query has been entered but before it has been executed.[1] The query is executed by selecting an operation in the *Database* menu which causes the selected data to be displayed in the table browser tool. Figure 3 shows the screen after the query has been executed. At this point the user can browse the selected data by using the scroll bars, update it by entering a new value into the table, or change the data being browsed by executing a different query.

Figure 4 shows a more complex interface composed of multiple displayers for the data. A graph viewer tool has been added to the interface which displays a graph of the data specified by the query. This interface is created by executing the graph tool program, adding an output port to the database tool (*Out-2*), and connecting it to the input port in the graph tool (*Data*). The graph tool is connected to the database tool so that it will display all of the data specified by the query. If the user wanted to graph a different data set, he would have to create another database tool and connect the graph viewer tool to it.

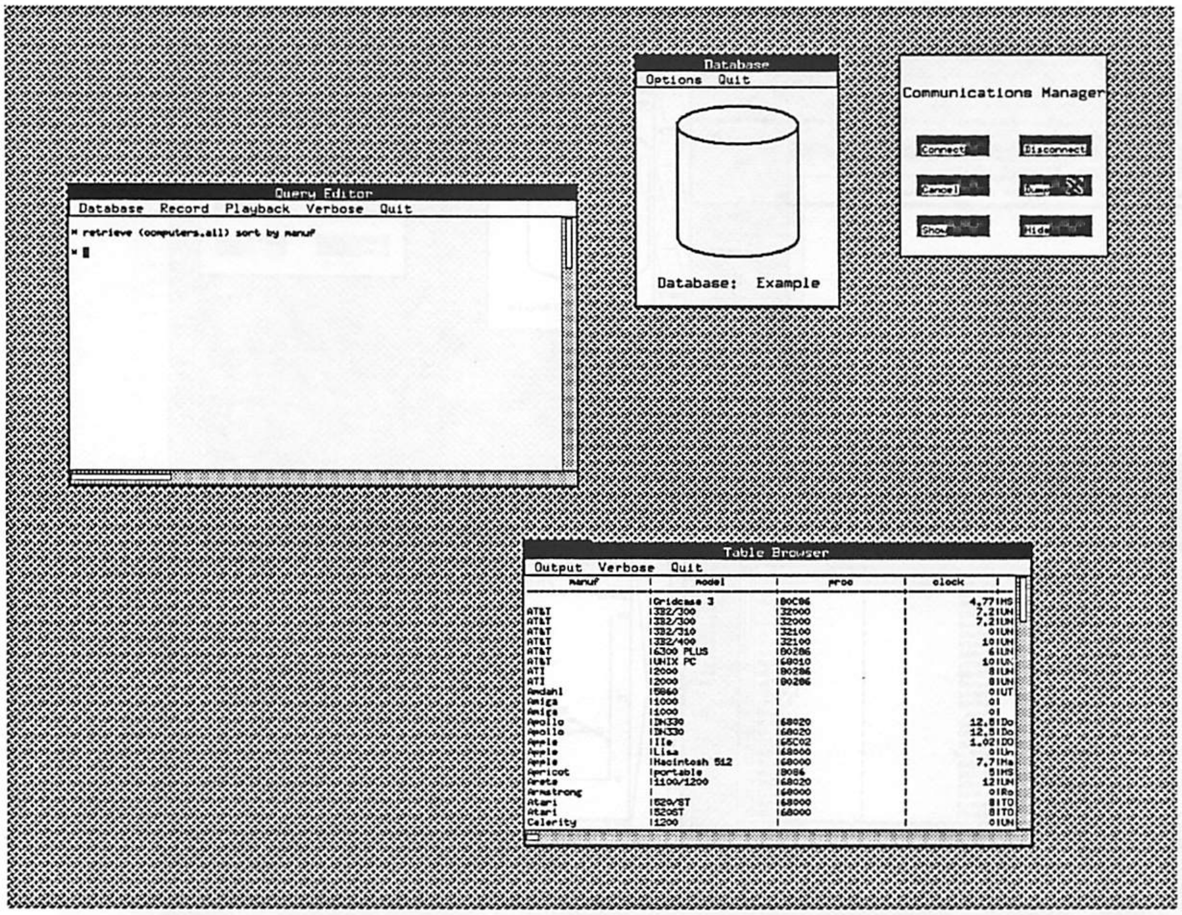
The interface shown in figure 4 allows the user to examine a graph of the entire data set while at the same time browsing the detailed data. The

[1] The example database used throughout the paper is a single relation that contains data about computer products along with configuration and performance attributes (e.g., manufacturer, model, dhrystone rating, etc.).



```
* retrieve (computers.all) sort by manuf
```

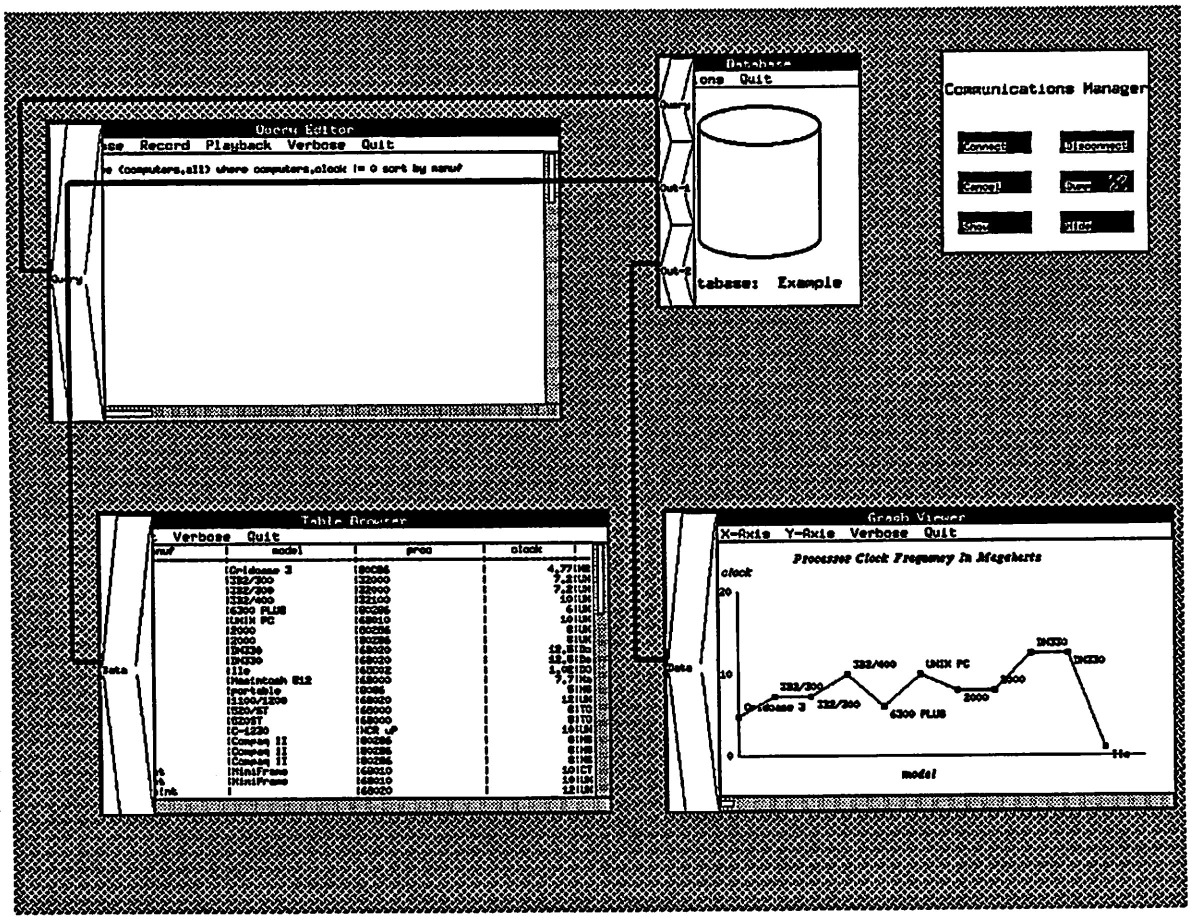
Figure 2: The browser after a query is entered.



```
* retrieve (computers.all) sort by manuf
* 
```

```
      manuf        |     model      |     proc       |   clock  |
-------------------+----------------+----------------+----------+
                   |Gridcase 3      |80C86           |      4.77|MS
AT&T               |3B2/300         |32000           |       7.2|UN
AT&T               |3B2/300         |32000           |       7.2|UN
AT&T               |3B2/310         |32100           |         0|UN
AT&T               |3B2/400         |32100           |        10|UN
AT&T               |6300 PLUS       |80286           |         6|UN
AT&T               |UNIX PC         |68010           |        10|UN
ATI                |2000            |80286           |         8|UN
ATI                |2000            |80286           |         8|UN
Amdahl             |5860            |                |         0|UT
Amiga              |1000            |                |         0|
Amiga              |1000            |                |         0|
Apollo             |DN330           |68020           |      12.5|Do
Apollo             |DN330           |68020           |      12.5|Do
Apple              |IIe             |65C02           |      1.02|DO
Apple              |Lisa            |68000           |         0|Un
Apple              |Macintosh 512   |68000           |       7.7|Ma
Apricot            |portable        |8086            |         5|MS
Arete              |1100/1200       |68020           |        12|UN
Armstrong          |                |68000           |         0|Ro
Atari              |520/ST          |68000           |         8|TO
Atari              |520ST           |68000           |         8|TO
Celerity           |1200            |                |         0|UN
```

Figure 3: The browser after the query has been executed.



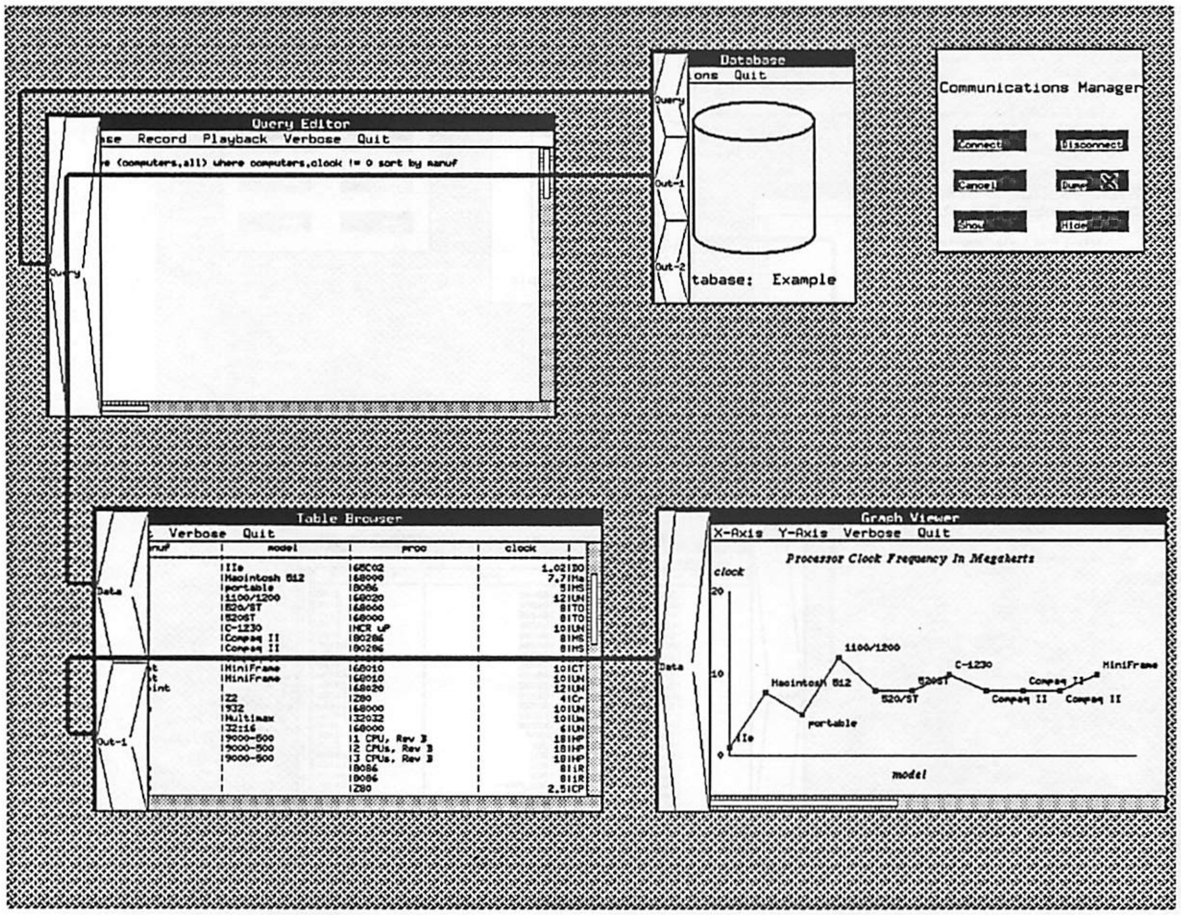
e (computers.all) where computers.clock != 0 sort by manuf

nuf	model	proc	clock	
	Gridcase 3	80C86	4.77	ME
	332/300	32000	7.2	UN
	332/300	32000	7.2	UN
	332/400	32100	10	UN
	6300 PLUS	80286	6	UN
	UNIX PC	68010	10	UN
	2000	80286	8	UN
	2000	80286	8	UN
	DK330	68020	12.5	Da
	DK330	68020	12.5	Da
	IIe	65C02	1.02	DO
	Macintosh 512	68000	7.7	Ma
	portable	8086	8	ME
	1100/1200	68020	12	UN
	520/ST	68000	8	TO
	520ST	68000	8	TO
	C-1230	NCR uP	10	UN
	Compaq II	80286	8	ME
	Compaq II	80286	8	ME
	Compaq II	80286	8	ME
t	MiniFrame	68010	10	CT
t	MiniFrame	68010	10	UN
int		68020	12	UN

Figure 4: A user-interface with multiple data displayers.

graph tool has menu operations that allow the user to change the attributes displayed in the graph and other properties of the graph itself (e.g., axis labels, line style, etc.). If the user changes the data set by executing another query, the data displayed in the table and the graph is updated.

A different interface can be created by connecting the graph tool input port (*Data*) to an output port in the table browser as shown in figure 5. This interface displays a graph of the data currently visible through the table. It is created from the multiple viewer shown in figure 4 by adding an output port to the table browser (*Out-1*), deleting the connection from the graph tool to the database tool, and adding a connection between the graph tool and the table browser. With this interface, the graph is changed whenever the user changes the data displayed in the table.



```
e (computers,all) where computers,clock != 0 sort by manuf
```

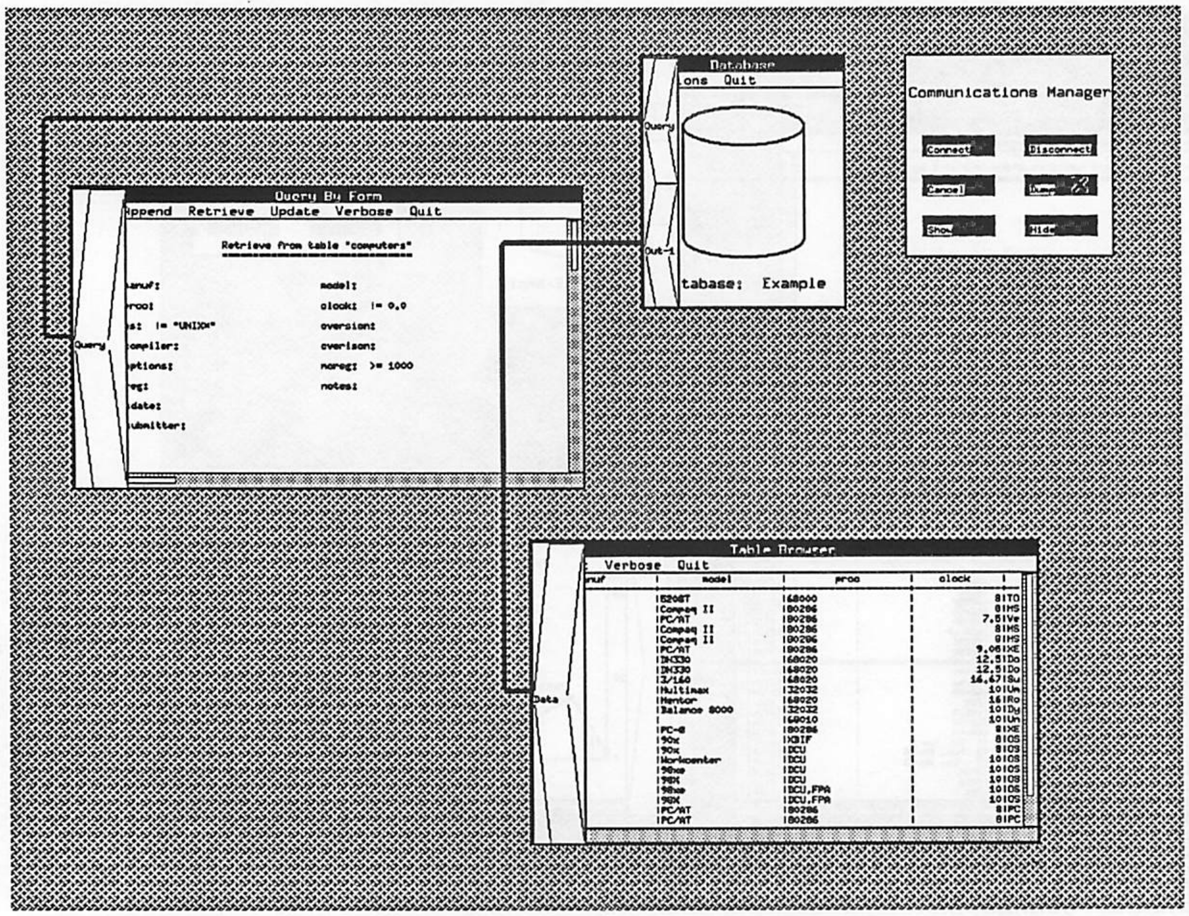
```
nuf |          model |           proc |        clock |
    |IIe             |65C02           |          1.02|DO
    |Macintosh 512   |68000           |           7.7|Ma
    |portable        |8086            |             5|MS
    |1100/1200       |68020           |            12|UN
    |520/ST          |68000           |             8|TO
    |520ST           |68000           |             8|TO
    |C-1230          |NCR uP          |            10|UN
    |Compaq II       |80286           |             8|MS
    |Compaq II       |80286           |             8|MS
t   |MiniFrame       |68010           |            10|CT
t   |MiniFrame       |68010           |            10|UN
int |                |68020           |            12|UN
    |Z2              |Z80             |             4|Cr
    |932             |68000           |            10|UN
    |Multimax        |32032           |            10|Um
    |32:16           |68000           |             6|UN
    |9000-500        |1 CPU, Rev 3    |            18|HP
    |9000-500        |2 CPUs, Rev 3   |            18|HP
    |9000-500        |3 CPUs, Rev 3   |            18|HP
    |                |8086            |             8|iR
    |                |8086            |             8|iR
    |                |Z80             |           2.5|CP
```

Figure 5: Another user-interface with multiple data displayers.

The user could also create an interface with two graph displays: one for the entire data set and one for the data displayed through the table viewer. This interface is created by executing two graph tools and connecting one to the database tool and the other to the table browser.

The last example shows how a different query specification tool can be used with the same data displayers. Figure 6 shows a simple database browser similar to the one in figure 1 except that the textual query specifier has been replaced by a forms-based query specifier. The interface behaves the same way as before except queries are specified differently.

This section showed how different database interfaces could be created by connecting together simple interface tools. VS provides an abstraction



```
Retrieve from table "computers"

anuf:                      model:
roc:                       clock:  != 0.0
s:  != "UNIX"              oversion:
compiler:                  overison:
ptions:                    noreg:  >= 1000
eg:                        notes:
date:
ubmitter:
```

```
nuf  |     model        |    proc       |   clock   |
     |5208T             |68000          |          8|TO
     |Compaq II         |80286          |          8|MS
     |PC/AT             |80286          |        7.5|Ve
     |Compaq II         |80286          |          8|MS
     |Compaq II         |80286          |          8|MS
     |PC/AT             |80286          |       9.06|XE
     |DN330             |68020          |       12.5|Do
     |DN330             |68020          |       12.5|Do
     |3/160             |68020          |      16.67|Su
     |Multimax          |32032          |         10|Un
     |Mentor            |68020          |         16|Ro
     |Balance 8000      |32032          |         10|Dy
     |                  |68010          |         10|Un
     |PC-8              |80286          |          8|XE
     |90x               |XBIF           |          8|OS
     |90x               |DCU            |          8|OS
     |Workcenter        |DCU            |         10|OS
     |90xe              |DCU            |         10|OS
     |90X               |DCU            |         10|OS
     |90xe              |DCU,FPA        |         10|OS
     |90X               |DCU,FPA        |         10|OS
     |PC/AT             |80286          |          8|PC
     |PC/AT             |80286          |          8|PC
```

Figure 6: A simple browser with a forms-based query specification tool.

and a visual interface for creating these interfaces.

3. Database Tools

This section describes the design and implementation of the primitive database tools. The next section describes the CM tool and the library used by tool programs to manage ports and connections.

The query tools are relatively simple. The user enters a query and selects a menu operation to execute it. The query tool sends a message containing a textual representation of the query to the tool connected to its *Query* port. After sending the message, the tool waits for another command from the user.

The database tool is more complicated than the query tools because it buffers data from the database and responds to requests sent to it by the display tools. The tool has a *Query* port, one or more data output *Out-i* ports, and a buffer containing the data set specified by the current query.[2] When a query arrives on the *Query* port, the previous query, if any, is terminated and the new query is executed.

After the data has been retrieved from the database into the buffer, the display tools are sent a "new data set" message to notify them that a new query has been executed. When a display tool receives the message, it sends a message to the database tool requesting a description of the new data set (i.e., the number of attributes and the names and types of the attributes). After the data set description is received, the displayer sends a message requesting the database tool to send it some number of data records. The number of records requested depends on the displayer and the amount of data needed. For example, the graph tool requests all records while the table browser requests only enough records to fill the table displayed on the screen.

The table browser tool maintains a buffer of data that is currently displayed on the screen. Other tools can access this data by connecting to an output port in the table browser which supports the same protocol implemented by the database tool. "New data set" messages sent to the table browser are propagated to the other tools when they are received so they can synchronize with the data being displayed in the table. "New data set" messages are also sent when the user scrolls the table. This feature was illustrated in figure 5.

Notice that messages are sent in both directions over the connections between the database tool and the displayers and the table browser and the other displayers. Also, notice that the only state maintained by the database tool is the current data set. Each display tool keeps track of the data currently being displayed.

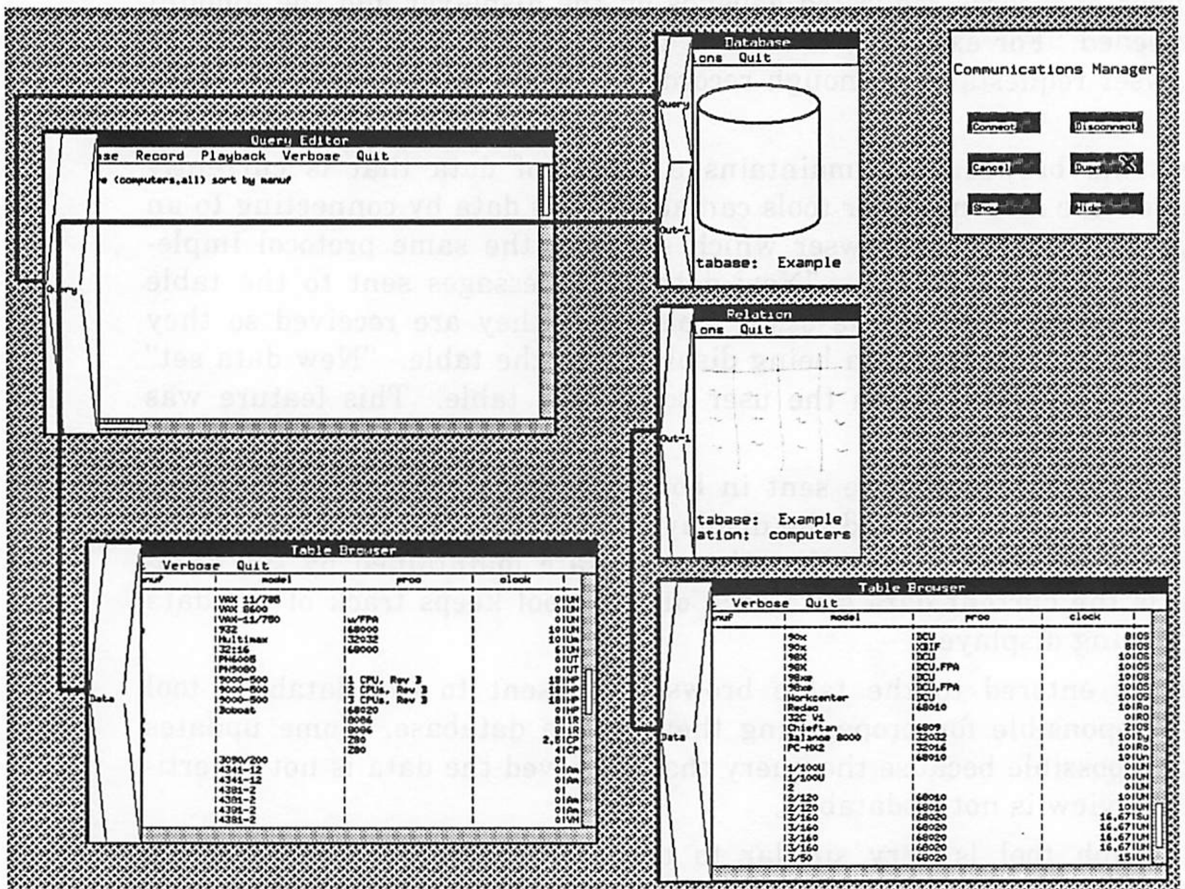
Updates entered to the table browser are sent to the database tool which is responsible for propagating them to the database. Some updates may not be possible because the query that retrieved the data is not invertible (i.e., the view is not updatable).

The graph tool is very similar to the table browser. However, the current implementation does not support output ports nor does it allow data to be updated. These restrictions were made to simplify the

[2] The current implementation keeps the data set in a file. A future version will only buffer a subset of the data. The optimum implementation would use a database system that supported the *portal* concept [StR84,StR86].

implementation.

The fact that these tools are useful yet simple to implement shows that this approach to building database interfaces is flexible and powerful. For example, suppose you wanted to build a real-time relation viewer that is dynamically updated when the relation changes. This interface could be built by implementing a relation tool that accepted the name of a database and a relation and that responded to the same data protocol used by the database tool. This tool could be connected to different displayers (e.g., the table browser, the graph viewer, etc.) to create different real-time interfaces. Figure 7 shows a sample relation tool connected to a table browser. To



```
e (computers,all) sort by manuf
```

```
nuf         | model         | proc              | clock |
            |VAX 11/785     |                   |     0|UN
            |VAX 8600       |                   |     0|UN
            |VAX-11/780     |w/FPA              |     0|UN
            |932            |68000              |    10|UN
            |Multimax       |32032              |    10|Um
            |32:16          |68000              |     6|UN
            |PH6005         |                   |     0|UT
            |PH5080         |                   |     0|UT
            |9000-500       |1 CPU, Rev 3       |    18|HP
            |9000-500       |2 CPUs, Rev 3      |    18|HP
            |9000-500       |3 CPUs, Rev 3      |    18|HP
            |Boboat         |68020              |     0|HP
            |               |8086               |     8|IR
            |               |8086               |     8|IR
            |               |Z80                |   2.5|CP
            |               |Z80                |     4|CP
            |3090/200       |                   |     0|
            |4341-12        |                   |     0|Am
            |4341-12        |                   |     0|Am
            |4381-2         |                   |     0|
            |4381-2         |                   |     0|Am
            |4381-2         |                   |     0|Am
            |4381-2         |                   |     0|VM
```

```
nuf         | model         | proc              | clock |
            |90x            |DCU                |     8|OS
            |90x            |XBIF               |     8|OS
            |98X            |DCU                |    10|OS
            |98X            |DCU,FPA            |    10|OS
            |98xp           |DCU                |    10|OS
            |98xp           |DCU,FPA            |    10|OS
            |Workcenter     |DCU                |    10|OS
            |Redeo          |68010              |    10|Ro
            |32C V1         |                   |     0|RO
            |Chieftan       |6809               |     2|OS
            |Balance 8000   |32032              |    10|Dy
            |PC-MX2         |32016              |    10|Ro
            |               |68010              |    10|Un
            |1/100U         |                   |     0|UN
            |1/100U         |                   |     0|UN
            |2              |                   |     0|UN
            |2/120          |68010              |    10|UN
            |2/120          |68010              |    10|UN
            |3/160          |68020              | 16.67|Su
            |3/160          |68020              | 16.67|UN
            |3/160          |68020              | 16.67|UN
            |3/160          |68020              | 16.67|UN
            |3/50           |68020              |    15|UN
```

Figure 7: A relation viewer interface.

implement the real-time updating behavior, the relation tool sets a database alerter on the relation to notify it whenever the relation was changed [BuC79].

4. Communication Manager

This section describes the CM tool and tool program library which together define the VS abstraction. The VS software architecture, the protocol used to implement CM operations (e.g., "show/hide ports and connections," "create/destroy connection," and "add/remove port"), and the general structure of a tool program are described.

The CM program and tool library are composed of about 3000 lines of code in C which runs on 4BSD Unix and uses the X window manager [Get86]. Figure 8 shows the process structure and interprocess communication (IPC) connections for the simple browser shown in figure 1. The CM and each tool run as a separate process. IPC connections are opened

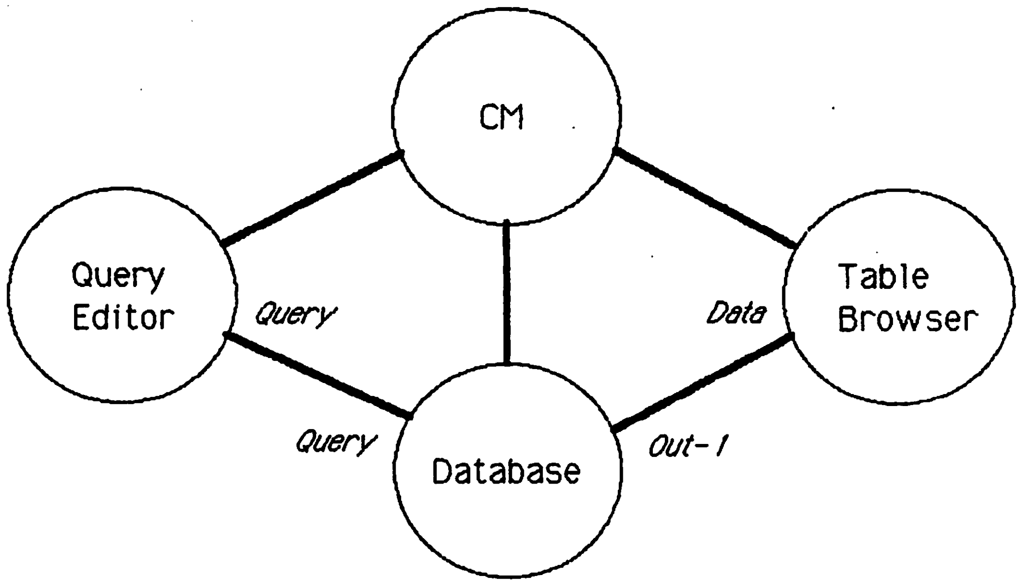


Figure 8: Process structure and IPC connections.

between each tool and the CM and for each connection between tools. Port names are shown at the ends of the IPC connections in the figure. The IPC connections are bidirectional, asynchronous message channels implemented by 4BSD internet domain streams. Consequently, the CM and tool processes may run on different hosts. The tool windows, however, must be displayed on the same screen so that connection lines can be drawn.

Tool programs are linked with a library, called *libCM*, that executes commands sent to the process by the CM and that provides the stream abstraction over which tools communicate. CM operations are initiated by buttoning a CM command or remotely by invoking a tool command (e.g., executing a menu operation to add a port). Regardless of how the operation is initiated, messages are sent between the tools and the CM to implement it. For example, suppose the user requested that port *P1* in tool *T1* be connected to port *P2* in tool *T2*. To specify this operation, the user executes the CM *Show* operation which displays the ports and connection lines. The CM implements this operation by sending a message to all tools requesting them to display their port window and then, draws the lines. The CM knows the location of all tool windows and the number and location of ports in each tool so that it can calculate where the lines should be drawn.[3]

After the ports and connections are displayed, the user buttons port *P1* in tool *T1* to select it as one end of the connection to be established. When the user selects the port, the libCM code in the tool sends a message to the CM indicating that the port has been selected. The same thing happens when the user buttons port *P2* in tool *T2* to specify it as the other end of the connection. To keep track of the selected ports, the CM maintains a "selected ports" list which is updated when it receives one of these messages.

After selecting the ports, the user executes the CM *Connect* operation to complete the connection.[4] Each tool creates a *connection establishment socket* and sends its name to the CM when the IPC connection to the CM is opened. The CM implements the *Connect* operation by sending messages to the two tools. First, the CM sends a message to *T1* directing it to execute a 4BSD *connect* call on *T2*'s connection establishment socket which is included in the message. Second, the CM sends a message to tool *T2* directing it to execute a 4BSD *accept* call on its own connection establishment socket. A *connect* and *accept* on the same socket causes the 4BSD kernel(s) to create a new IPC connection between the two processes. The tools can now

[3] It draws them through a transparent X window covering the entire screen.

[4] The current implementation only supports connections between two ports. This restriction could be relaxed to support a form of multicast connections if a multicast communication protocol existed.

communicate by calling the libCM routines *ReadPort* and *WritePort* which pass the requests on to the standard 4BSD stream calls (i.e., *read* and *write*).

Other CM operations and libCM functions (e.g., adding or deleting a port) are implemented in a similar fashion. The LibCM code in each tool maintains data structures describing the tool's ports and the CM maintains data structures describing the ports, connections, and connection establishment socket for each tool. Messages are sent between the tool processes and the CM process to synchronize the data structures.

The general structure of a tool program will be described by looking at a tool that copies data from its input port to its output port(s). The tool has menu operations that allow output ports to be added or removed dynamically. Figure 9 shows a skeleton program for this tool. The global data structures keep track of the input and output ports.

The program itself contains: 1) a main routine (*main*) that initializes the tool and dispatches events, 2) a menu routine (*MenuOp*) that is called when an *AddPort* or *RemovePort* menu operation is invoked by the user, and 3) a routine (*CopyData*) that copies data to the output port(s) when data is received on the input port. *Main* creates the tool window and menu, opens the CM connection, initializes the input port (*Tee-In*) and one output port (*Tee-Out-1*), and enters the event dispatch loop. LibCM routines have names with the prefix *CM* (e.g., *CMOpen* and *CMAddPort*).

The code shown in the figure handles three types of events: menu selections, data available on a port, and port selections. Menu selections and data available events are handled by the program, while port selections are passed to libCM to be handled. The routine *MenuOp* is called when a menu selection is made. The code for an *AddPort* menu operation calls the libCM routine *CMAddPort* to create the port. The port descriptor returned is saved in the global data structure. The code for *RemovePort* (not shown here) is similar except that it calls the routine *CMRemovePort* rather than *CMAddPort*.

The routine that copies data (*CopyData*) is called when a data ready event is received. It just calls the routines *CMReadPort* and *CMWritePort* to read and write the port, respectively.

This section described how the VS abstraction is implemented and illustrated how a tool is coded in C.

5. Extensions

This section describes several extensions that can be made to the existing system. First, more tools need to be implemented. The relation tool described above and other browsers are needed to build up the primitives from which user-interfaces can be constructed. Another useful tool might be an icon browser such as SDMS [Her80] or TIMBER [StK82]. The icon browser tool could display the icon representation of the data. Then, when

```
 /* TEE - copy input port to all output ports.  */

 /* Global data structures for ports. */
PORT   inport, outport[MAXPORTS];
int      nports = 0;

main()
{  WINDOW    wd;
   EVENT       ed;

    /* Create window and menu. */
   wd = CreateWindow(...window arguments...);
   AddToMenuBar("Ports", CreateMenu(...menu items...));

    /* Open CM connection. */
   if(CMOpen(wd))
      error("TEE: could not open CM connection.");

    /* Open IN port and one OUT port. */
   inport = CMAddPort("Tee-In");
   outport[nports] = CMAddPort("Tee-Out-1");
   nports = 1;

    /* Main loop. */
   while(TRUE) {
      switch((ed = GetEvent())) {
        case MENUSELECTION:     MenuOp(ed);  break;
        case PORTDATA:          CopyData();  break;
        case PORTSELECTION:    CMEvent(ed);  break;
      }
   }
}

MenuOp(ed)       /* Display menu and execute user selection. */
EVENT   ed;
{
   switch(DisplayMenu(ed)) {
     case ADDPORT:
                    outport[nports] = CMAddPort(MakeName("Tee-Out-", nports + 1));
                    nports = nports + 1;
                    break;

     case REMOVEPORT:
          . . .
   }
}

CopyData()       /* Copy data from inport to all out ports. */
{  BYTE    buffer[BUFSIZ];
   int       nbytes;

    /* Read the IN port "Tee-In" and write OUT ports "Tee-Out-i." */
   nbytes = CMReadPort(inport, buffer, BUFSIZ);
   for(i = 0; i <= nports; i++)
      CMWritePort(outport[i], buffer, nbytes);
}
```

Figure 9: Tool program structure.

the user selects an icon, the tool could execute an icon-specific tool that could be connected to other tools. Example icons might be forms, reports, graphs, or a spreadsheet.

A second extension needed by the current system is a mechanism to save an interface configuration so that the user does not have to recreate it each time he wants to use it. A visual specification language is needed that allows a user to select and configure the tools and to save the definition in a form that can be executed later. Another useful mechanism that is needed is a way to close an interface and show it on the desktop as an icon similar to the way a window can be closed to an icon in a conventional window manager. This feature could be implemented by designating one or more tools as "interface configuration tools." These tools could change their window to an icon when they were notified that some tool in the interface configuration received an *iconify* operation.

A third extension to the database tools described above would be to abstract the output destination from the displayers. For example, all of the current displayer tools (e.g., the table browser, the graph viewer, etc.) have a fixed output destination. It should be possible to redirect the output of these tools to a printer, a file, a relation, or another tool.

Finally, the structure of the tools we have created thus far suggests that there might be a *metatool* structure. For example, all displayers have a similar structure. They fetch data, transform it, and display it. It may be possible to define a "display metatool" that is a parameterized tool which can be instantiated with different transformation modules.

6. Conclusions

This paper described the design and implementation of a visual shell that was used to implement a flexible interface to a relational database system. Our experience with the system has been very good. The user-interfaces are easy to use and perform acceptably well.

Acknowledgements

We want to thank Deborah Ohlsen for the work she did on an earlier version of the system and Joe Cortopassi and Peter Schmitz for their suggestions on improving the database tool interfaces.

References

[Bou78] S. Bourne, "The UNIX Shell", *The Bell System Technical Journal 57*, 6 (July 1978).

[BuC79] O. P. Buneman and E. K. Clemons, "Efficiently Monitoring Relational Databases", *ACM Trans. Database Systems* , Sep. 1979, 368-382.

[Cat84] R. G. G. Cattell, A Paradigm for Database Editing, Browsing, and Query, Unpublished manuscript, Xerox PARC, 1984.

[FrE86] C. Frasson and M. Er-radi, "Principles of an Icons-Based Command Language", *Proc. 1986 ACM SIGMOD Conf. on Mgt. of Data,* Washington, DC, May 1986.

[Get86] J. Gettys, "Problems Implementing Window Systems in UNIX", *Proc. Winter USENIX Technical Conf.*, Jan. 1986, 89-97.

[Goe85] K. J. Goldman and et. al., "ISIS: Interface for a Semantic Information System", *Proc. 1985 ACM SIGMOD Conf. on Mgt. of Data,* May 1985.

[Hae86] P. E. Haeberli, "A Data-Flow Manager for an Interactive Programming Environment", *Proc. 1986 Summer USENIX Tech. Conf.*, June 1986.

[Her80] C. Herot, "SDMS: A Spatial Data Base System", *ACM Trans. on Database Systems*, Dec. 1980.

[IBM83] *Query Management Facility: General Information.*, IBM Form No. GC26-4071, Armonk, NY, June 1983.

[LJF83] S. Leffler, W. Joy and R. Fabry, *A 4.2BSD Interprocess Communication Primer*, Computer Science Division - EECS, U.C. Berkeley, July 1983.

[McS75] N. McDonald and M. R. Stonebraker, "CUPID - The Friendly Query Language", *Proc. of the ACM Pacific Conf.*, San Francisco, CA, Apr. 1975.

[MET8?] *Some reference manual, Metaphor Computers, Inc., Meno Park, CA, 198?.*

[Shu83] J. Shultis, "A Functional Shell", *SIGPLAN Notices Notices 18*, 6 (June 1983), 202-211.

[StK82] M. Stonebraker and J. Kalash, "TIMBER: A Sophisticated Relation Browser", *Proc. 8th Very Large Data Base Conference,* Sep. 1982.

[StR84] M. R. Stonebraker and L. A. Rowe, "Database Portals: A New Application Program Interface ", *Proc. 10th Int. Conf. on Very Large Data Bases*, Aug. 1984.

[StR86] M. R. Stonebraker and L. A. Rowe, "The Design of POSTGRES", *Proc. 1986 ACM-SIGMOD Int. Conf. on the Mgt. of Data*, June 1986.

[The83] C. W. Thompson and et. al., "Building Usable Menu-Based Natural Language Interfaces to Databases", *Proc. of the 9th Int. Conf. on Very Large Databases*, Florence, Italy, Oct. 1983, 43-55.

[RTI84] *INGRES QBF (Query By Forms) User's Guide*, Version 3.0, VAX/VMS, Relational Technology, Inc., Berkeley, CA, May 1984.

[Zlo75] M. M. Zloof, "Query by Example", *Proc. NCC 44* (1975).