Application Development with VA's FileMan: From Terminals via Client/Server to Web

by Mikko Korpela, Mauri Kaatrasalo, Hellevi Ruonamaa

The majority of the hospital information systems installed in Finland are based on the FileMan/Kernel technology of the U.S. Department of Veterans Affairs (VA). In 1995, a project was established to "modernize" the systems in a stepwise manner. As a first step, the strategic alternatives available were analyzed (Karvinen *et al.* 1996). The strategy selected was based on the client/server architecture, with Borland's Delphi as the user interface technology and the VA's Remote Procedure Call (RPC) Broker as the client–to–server communication technology.

In this paper we report on the experience in Finland thus far in developing tools and standards for modernized client/server applications based on FileMan, Broker, and Delphi. The reader is expected to have a basic knowledge of these technologies. We first study the architecture of client/server systems based on FileMan and Delphi in section 1, presenting an object-oriented breakdown into high-level functional components. In section 2 we present the generic functionality of the most fundamental part of any database application—file entry, edit, and browsing. The next section deals with the architecture and tools for producing textual and graphic reports from a FileMan database in the client/server context.

The move from centralized systems with "dumb" terminals to distributed systems with PC clients and graphic user interfaces is a major step. However, another similar technical revolution is already around the corner. In section 4, therefore, we discuss the challenge of the World Wide Web (WWW) technology to the FileMan/Broker/Delphi architecture we are just developing and identify ways of ensuring a smooth introduction of the WWW technology. At the end of the paper we draw some general conclusions of our experience.

1. The functional architecture of FileMan/Broker/Delphi systems

Three of the five university teaching hospitals in Finland (Helsinki, Turku and Kuopio), the leading vendor of laboratory information systems (Mylab Corporation) and the Computing Centre of the University of Kuopio established a project in 1995 to explore the ways of modernizing existing systems based on VA's FileMan and Kernel.

The University of Kuopio has been deeply engaged in introducing M and the VA technology in Finnish health informatics since the late 1970s and mid-1980s, respectively. During the 1990s its Computing Centre has functioned as the national support center for FileMan and Kernel, translating new versions to Finnish, adapting them to the technological and cultural requirements in Finland, and offering technical support to M software houses. Since 1992, the Computing Centre has developed in-house administrative systems with FileMan and HyperM, a Graphic User Interface (GUI) software running on InterSystems' DataTree M, originally developed by SAIC and later distributed in Europe by CDS Ltd., UK. We thus had some years of experience already in client/server technology and GUIs in the M/FileMan/Kernel environment.

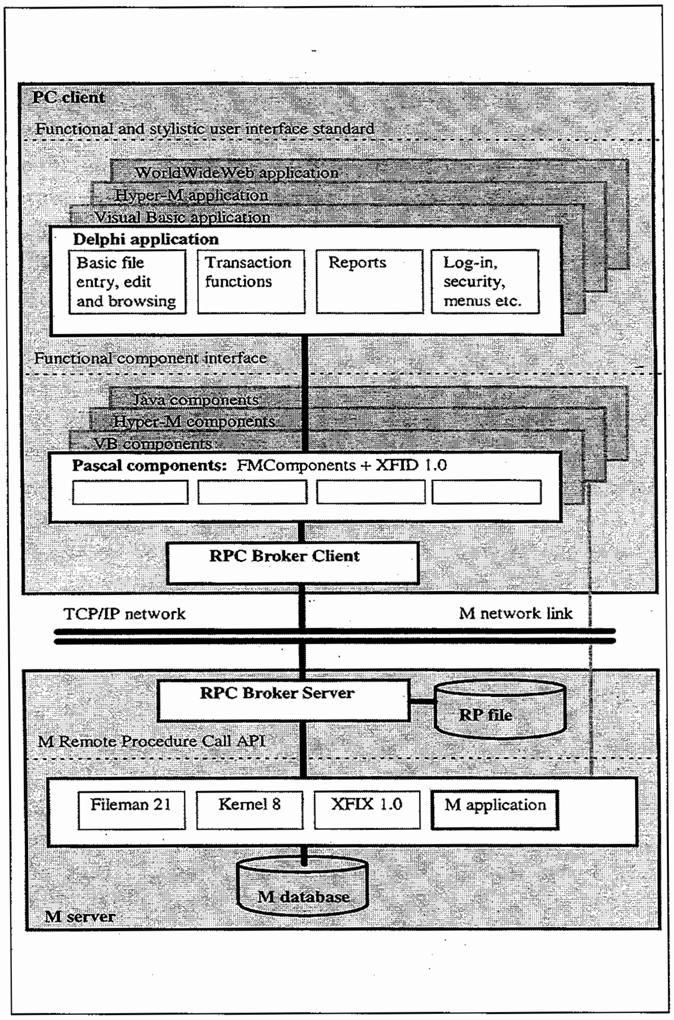


Figure 1 - The program interfaces between the M database and the user

After analyzing the pros and cons of three strategies—all-M, standard tools, and compromise—the project decided on selecting Borland's Delphi and VA's RPC Broker as the foundation of the next generation of M-based hospital information systems in Finland (Karvinen *et al.* 1996). We recognized early on, however, that the selection of the basic technologies is just a small part of the task of designing a new applications development methodology. Furthermore, the user interface should not be too dependent on any one proprietary technology. It is important to identify major *functional* components as the building blocks of any application package and specify these components in such a way which can be implemented in a variety of technologies.

Our view of the functional architecture of client/server systems based on FileMan and Broker is depicted in Figure 1. The overall system breaks down into a client part written in Delphi Pascal and the server part written in M. These parts interact over a TCP/IP network through messages assembled and interpreted by the RPC Broker.

Delphi is a development tool for Windows-based client/server applications (Borland 1995). It competes mainly with Microsoft's Visual Basic, but is considered more efficient during run time and more genuinely object-oriented. It comes with a large choice of pre-programmed visual components written in Object Pascal. In our architecture, Delphi's role is to provide the *user interface technology*.

In any client/server system, there needs to be a *data exchange mechanism* which connects the user interface and the database. In this case, VA has developed the RPC Broker to connect Delphi with FileMan (VA 1996). The Broker consists of two parts, one at the client computer (written in Object Pascal) and the other at the server computer (written in M). Pascal programs at the client can call M procedures (routines) at the server through the Broker. The Broker's client part sends the name and input parameters of the M routine to the server part, which executes the routine and sends the results back. The Broker thus provides a well-defined *Application Program Interface* (API) to the M part—only those routines which are registered in the Remote Procedure file can be called and only by users who have been granted access to them through the standard Kernel security functions.

Another standard interface has to be defined *between the end user and the GUI software*, in functional and stylistic terms. Part of this standard derives from Windows and Delphi, but there are still many decisions to be made locally. In Finnish hospitals, various applications will increasingly be purchased from different vendors using different technologies, M and non-M. From the user's point of view it is outrageous if there are conflicting practices in the packages that she or he needs daily—if a certain semi-automatic sequence of user entry performs one thing in one package and a completely different thing in another. It is not possible or even necessary to standardize everything, but we are trying to coordinate our efforts with other vendors in the health information systems arena in Finland to avoid outright conflicts in the user interface.

From the systems developer's long-term point of view, it is important that the *functionality of the user interface* be defined in such a way that it can be implemented in various present and future technologies and is not dependent on one specific proprietary tool—although in practice the implementation must always rely on *some* technology, and there is always an overhead in switching from one technology to another. In our case, HyperM is a good touchstone for the portability of GUI functionality, since it supports Windows-look-alike displays on dumb VT220-compatible terminals. If we can implement the central ideas of our user interface standard to some extent with HyperM and VT220 terminals, then we can be quite sure that the functionality can be implemented with other GUI tools as well. Of course, it is more important to provide for compatibility with future rather than passing technologies, so we are trying to keep abreast of new operating systems and Web browsers to see which type of functions can and should be used in the GUI.

According to the object-oriented paradigm, systems should be composed of reusable components in a hierarchical manner. The object class "applications" should thus be analytically decomposed to its *highest level functional subcomponents* in a top-down manner. When these components have been implemented from bottom up using lower level components, they can be used as the standard building blocks of any application package. In our mind, any on-line database application includes the following types of building blocks, among others (Figure 1):

- user log-in and security,
- function selection through commands/menus/desktop,
- basic file entry/edit/browsing,
- reports, and
- transaction functions.

For maximum productivity, the system developer should be able to quickly compose the bulk of the application package from such "pre-cooked" building blocks. The most important core functionality of the application, the transaction functions, probably need to be so carefully tuned to fit the flow of the work process in question, that no standard building blocks can be of much help in developing those parts. However, if the more routine 80% of the application software can be easily composed in 20% of the time available, then the developer can concentrate on the most demanding part.

In this case the log-in and security functions, etc., will be part of the VA's Broker tools. The basic file entry/edit/browsing and reporting parts are not covered by existing tools, so we decided to prepare as many pre-cooked building blocks as possible for them. Our approach to these will be presented in the next two sections of the paper.

When an application has been assembled from standard high-level components in a consistent manner, then it is relatively easy to re-build the same application from *functionally equivalent components* in a different technology, as need arises (see Figure 1). For instance, when analyzing the requirements for the Delphi components, we simultaneously experimented with the Publication Register and Research Project Register of the University of Kuopio. Since the VA's basic Delphi components were not yet available, we developed functionally equivalent GUI building blocks on HyperM for the most important parts. It appears now that it will be fairly easy for us to convert the HyperM forms of these applications to Delphi when the tools are ready—the structure and functionality of the visible parts of the applications remain the same; what changes are the standard components and the programming language binding them together. It will be much harder for us to convert the older HyperM applications which were not composed of standard building blocks.

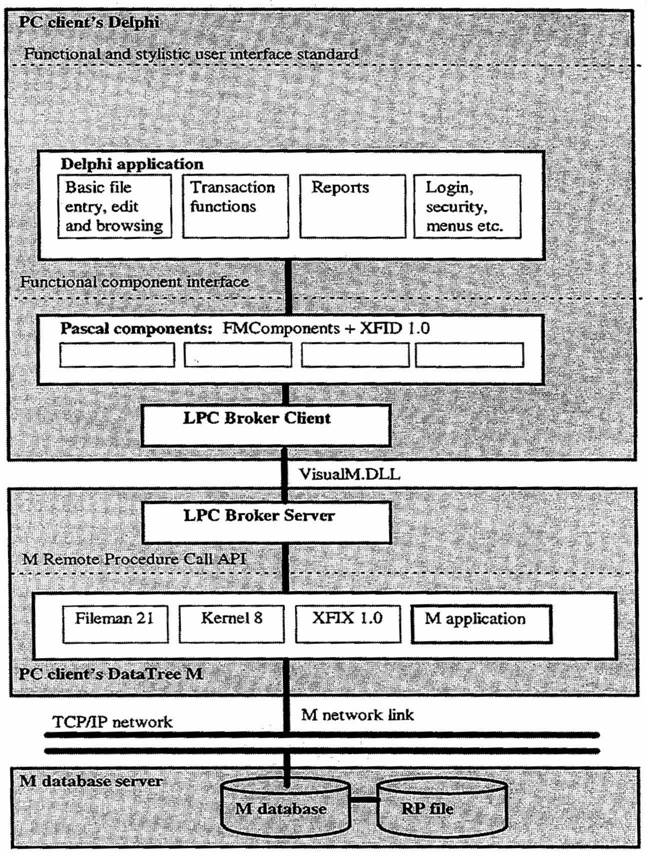


Figure 2 - The M-to-Delphi linkage implemented within the client: "Local Procedure Call Broker"

The benefits of the modular structure have manifested themselves in one more way. The VA's Broker is specifically intended for *Remote* Procedure Calls across a TCP/IP network. In our program development and demonstration environments, however, it was highly useful to have the Delphi and M parts of the system on the same PC client, which might contain the entire database as well or map the M globals across the network from an M database server (see Figure 2). We therefore needed a "*Local* Procedure Call Broker" between Delphi and M.

Within a couple of weeks we were able to modify the RPC Broker code by removing the calls to TCP/IP ports and replacing them with calls to the InterSystems' Visual M Dynamic Linkage Library. All the code above and below the Broker components remained the same. Simply by re-compiling the Delphi parts with the RPC or LPC library we get networked or stand-alone versions of the applications, respectively. In our mind this is an encouraging argument regarding the power and flexibility of the VA's Broker architecture.

2. The basic database entry, edit, and browsing functionality

When designing the functionality and the basic building blocks for the routine file entry, edit, and browsing components, the task ahead is to design how all the basic and advanced features of existing FileMan databases can be *visualized and made controllable to the end user*, with the minimum amount of application programming. Since FileMan is a network database, not a relational one, standard techniques from commercial database front-ends cannot always be adopted.

The VA has developed a set of Delphi components, the FMComponents, which implement the linkage between visual display elements (e.g., radio buttons) and FileMan elements (e.g., a 'set of codes' type of field in a file) through the Broker (remote or local). We have designed *a higher level of abstraction* above the FMComponents, namely a standard way of visually presenting the network database structures of FileMan.

We first defined a small FileMan database which contained all the various aspects that could be found in real life cases in a minimal setup (Figure 3). It is indeed intended to be a testbed only, not applicable to any real use, although we used familiar terms from the hospital laboratory environment.

FileMan files are depicted in Figure 3 as card decks, each card representing a file entry with a few fields. One of the files (*Lab Result* file) contains a hierarchical structure of "multiple-valued fields" (subfiles), although we nowadays try to avoid this feature of FileMan's. Logically equivalent but more flexible subfile structures are created by pointer fields, depicted by arrows—e.g., the *Departmental Lab Test* file can be seen as a subfile to both the *Lab Test* file (list of wards/clinics which can order this test) and the *Ward/Clinic* file (list of tests which can be ordered by this ward or clinic). The hierarchical subfiles are easier to manage but less suitable for various data retrieval purposes, compared with the "flat" logical subfiles created by pointer fields. The standard user interface must represent both types of subfiles in the same way.

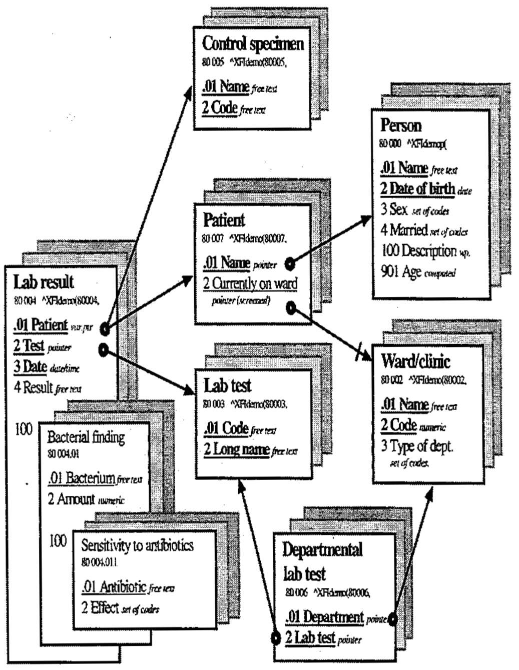


Figure 3 - The demonstration database

The demonstration database contains examples of all FileMan data types including yes/no (*Person* file's *Married* field), word-processing, computed, screened pointer (*Patient* file's *Currently on Ward* field), pointer as a name field (in *Patient* file), and even a variable pointer (*Lab Result* file's *Patient* field). There are also cross-referenced (underlined) and identifier (bold) fields.

We developed a visual presentation for all the features included in Figure 3. The examples which follow are intended to demonstrate the techniques, not to be artistically polished.

Figure 4 presents the basic layout of a form dealing with one FileMan file, the *Ward/Clinic* file in this case. The fields of the file are displayed on one or more "pages" of the Delphi's standard "tabbed notebook" structure. The tabbed notebook component is not available in all user interface technologies, but at least in HyperM the same functionality can be satisfactorily simulated. The *Save, Save As* and *Delete* push-buttons refer to the file entry currently on the form; *Save As* and *Delete* are used only on the first page if the fields of a file entry span more than one page of the form. All functionality of the form is accessible from the keyboard also, without the mouse which is often impractical in routine tasks.

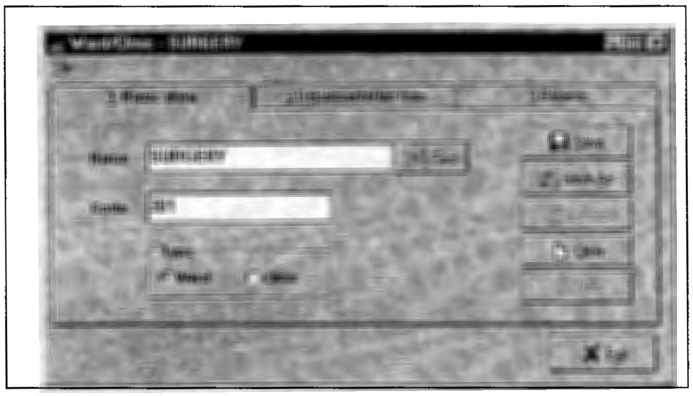


Figure 4 - A sample form for basic file entry, edit, and browsing

In the Windows standards, the object to be displayed (file entry in this case) is retrieved in a cumbersome way through a File pull-down menu. We wanted to retain a more FileMan look-alike way of selecting the file entry. The user can type a few characters from the beginning of the name of the entry in the corresponding field and then hit the Enter key or push the Find button with the mouse. If the string is not sufficient to uniquely identify the entry, a VA-supplied selection component will show the choices very much in the same way the "old" FileMan does.

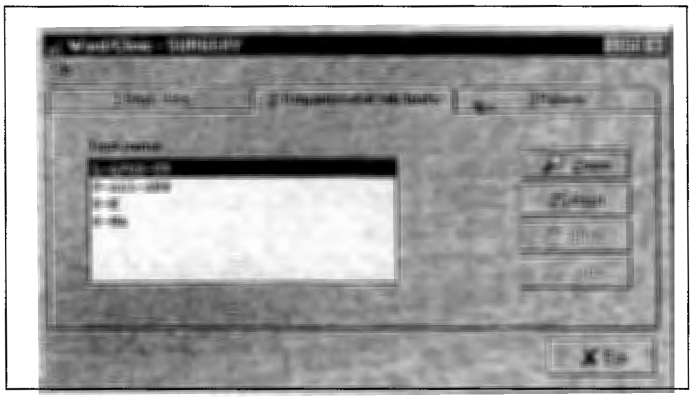


Figure 5 - A sample "subfile list page" within a "tabbed notebook."

If there are subfiles, either as multiple-valued fields or as other files pointing to the current one, each of them requires a page of its own in the tabbed notebook. In Figure 5, such a *subfile list page* is presented, displaying a list of all the *Departmental Lab Test* entries which point to the current *Ward/Clinic* (Surgery in this case). A third page will display a list of all the patients currently on this ward, since there is a pointer from the *Currently on Ward* field of the *Patient* file to the present file (refer to Figure 3). The lists can in real life display more than one piece of information about each item in a columnar way; in Figure 5 there is nothing more to display in the demonstration database besides the code of the laboratory test.

If the user has sufficient rights to "navigate" further in the database structure, she or he can select an item from the list and "zoom in" to it either by double-clicking or by pushing the *Zoom* button. This will open up a new form dealing with the subfile (i.e., with the *Departmental Lab Test* file in this case Figure 6). The new form is opened slightly to the right and down over the previous one, so that the titles of both forms will be visible. The new form can again be a tabbed notebook containing subfile list pages which can again be used to zoom in further to the database structure along the paths provided by the information in file.

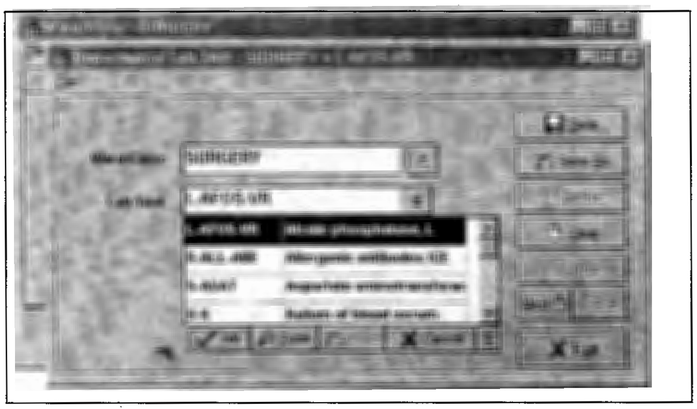


Figure 6 - Another file opened up by zooming from the "subfile list." A "special look-up component," with a further zoom button, is opened at a pointer field.

The *Next* and *Previous* buttons browse through the file in an order which depends on the way this form has been arrived at. In our example, the buttons would move forward and backward on the list displayed in Figure 5 (i.e., browse through the departmental lab tests (logical subfile) attached to the Surgery Ward). However, the same *Departmental Lab Test* form can be reached from the *Lab Test* file, through a subfile list page displaying all the entries pointing to a selected laboratory test (i.e., a list of all the wards/clinics which can order this test). In that case the *Next/Previous* buttons would move within that subfile. The *Departmental Lab Test* form might also be directly accessible from the main menu of the application; in that case, the *Next/Previous* buttons would work in the alphabetical order of the name field ("B" cross-reference in the FileMan file).

Figure 6 presents still another mechanism for navigating in the database. Both the *Department* and the *Lab Test* fields of the *Departmental Lab Test* file are pointers to other files. If the user has sufficient access rights, she or he can again zoom in along a pointer and open up a new form displaying an entry in the file pointed to. This is equivalent to the "Learn As You Go" (LAYGO) functionality in traditional FileMan. For instance, if the user wants to have a closer look at the *Lab Test* in question, she or he will click at the "drop-down list button" at the end of the field. A list of the choices (i.e., entries in the *Lab Test file*) will be displayed. If there are more choices than it is reasonable to transfer across the network at one time, a *More* button will also appear (depicted by a double down-arrow). The next batch of choices will be retrieved from the server by pushing this button.

If the user has the FileMan LAYGO rights, a *Zoom* button will also appear. By selecting an item on the list and clicking this button, or double-clicking the item, the user can open a new form (Figure 7) displaying all the information available about the *Lab Test* in question. If the user has sufficient rights, she or he can even edit the lab test's basic data and navigate further across the database to the test results and so forth.

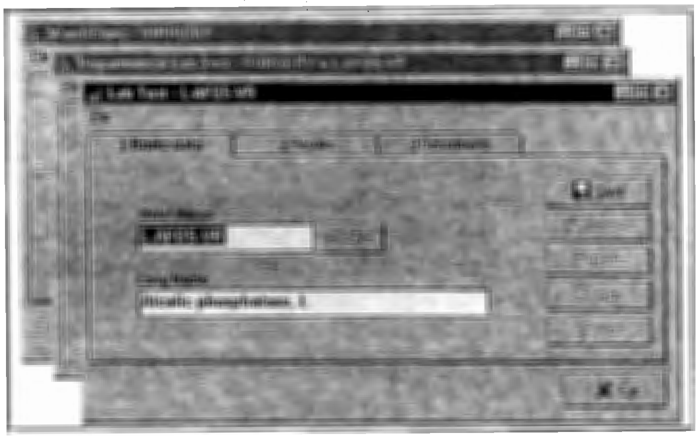


Figure 7 - Hitting the "zoom" button opens a further form via the pointer.

In summary, the tabbed notebooks, subfile list pages, and the two types of zoom functions taken together are capable of providing a visual presentation of the entire database structure of any complexity. Provided that the user has sufficient access rights, she or he can move around the "hyperdata" as far as the PC's memory and other resources permit. The basic aspects of the user interface are incorporated in Delphi form templates which systems developers can use as the basis. For each file of the database, one form needs to be developed, possibly with a number of pages. The same form can then be used as the "data entry method" of the file in question throughout the application, irrespective of whether it is accessed directly from a menu or through a linkage from another file, in accordance with the object-oriented paradigm.

3. Report generation functionality in the client/server environment

Besides the interactive browsing functions discussed in the previous section, the users will also need more voluminous reports on paper or screen. In the client/server environment, however, it is not self-evident how voluminous data should be transferred across the net from the server to the client. In the Broker architecture in particular, the client and the server can communicate only through request messages sent by the client and response messages sent by the server. A program running on the server cannot directly write to the client's screen or any other device at the client end.

Let us use a simple practical example to study the requirements for the architecture and tools for producing textual and graphic reports from a FileMan database in the client/server context. Figure 8 presents an example of the types of report generation parameters required, assuming that the user wants a report from the *Lab Result* file. Typically, a report will contain a *selection* of the file entries (in this case the results of a few selected laboratory tests for a given patient for the last month) *sorted* in a given order (in this case first according to time, then according to the test code), *formatted* in a given way (in this case as a graph instead of a print) and output on a given *device* (in this case a window on the client PC's screen). Some reports should work with fixed parameters—"click this button to produce the patient's standard cumulative lab test report"—while it should also be possible for the user to specify more or less *ad hoc* reports. The user should be able to save the specifications of an ad hoc report for easy repetition.

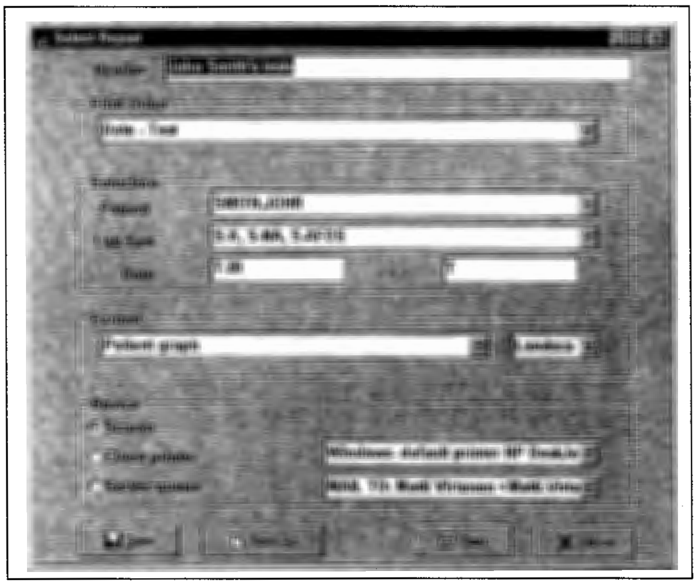


Figure 8 - A sample report start-up window.

How do we implement such reporting functionality? A client programmer can use VA's FMComponents to request for a list of raw data (e.g., the laboratory results of a given patient and then write a Pascal program to format and display it). This is reasonable when relatively small amounts of data are transferred in an interactive setting or displayed in a graphic way. However, most reports will remain textual, and there may already be an existing M routine to produce such a report on the server side. Since our main interest is in modernizing legacy systems based on FileMan, we wish to reuse existing software whenever possible.

Figure 9 presents our view of the general architecture of report generation by making use of existing M software. Starting from the upper left corner, the user will select a report from a menu. A few most-recent user-specified reports should be available in addition to the pre-programmed

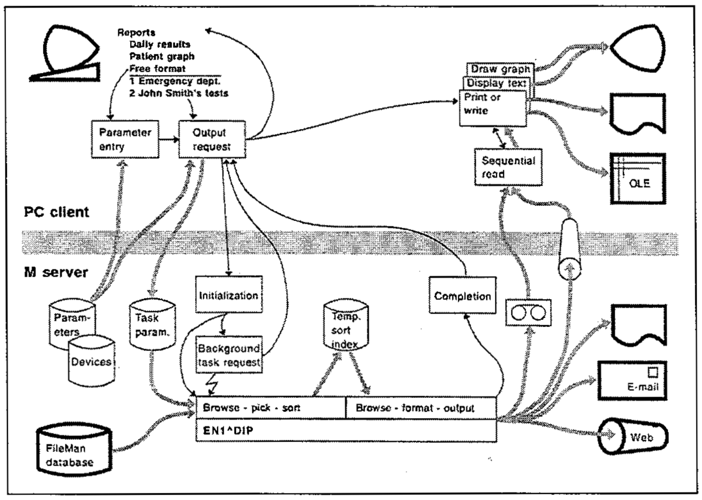


Figure 9 - The report generation architecture

reports. The parameters of each report are stored in a set of parameter files on the server side. The parameters include:

• Which FileMan file is the basis for the report?
• Which fields are used for selecting the subset of file entries for the report? What are the default selection criteria for each field? Can or must the user modify the selection criteria when starting the report?
• In which order (by which cross-reference) will the FileMan file be browsed through, to search for the entries to be selected? Is the browsing order dependent on the actual selection criteria the user entered?
• What is the default sorting order in which the entries will be output (sorted by which fields)? Which FileMan sort template or M routine will implement the sort? Can the user select from other sorting orders when starting the report?
• What is the default formatting of the output? Which FileMan print template or M routine will implement the formatting? Can the user select from other formats? Is the selection of formats dependent on the actual sorting order the user selected?
• What is the default output device? Can the user select from other devices? Is the selection dependent on the actual format the user selected?

If the parameters of the report are not completely fixed, a window like the one in Figure 8 will pop up and the user can modify the default parameters. An output request component will then be called (see Figure 9). It will send the name of the M procedure and the actual start-up parameters through the Broker to the server side. If the user selected to print the report on one of the server's devices, the report generation procedure can be started as a TaskMan task in the background, and the server procedure just needs to send a message back to the client to inform it that the request has been completed. Otherwise, the server procedure must call the given M report generation subroutine and wait for its completion before responding to the client.

All report generation procedures are comprised of the same steps. First, the database will be browsed through according to some index (a cross-reference or the physical order of entries), trying to keep to the minimum the number of entries traversed. At each entry, the selection criteria are studied to see if the entry must be picked for output or not. If the browsing order is not the same as the output order, a temporary sort index must be generated. In that case, the entire procedure of "browse–pick–store in temporary index" must be completed first and then the index browsed through again. At each printable file entry, information will be retrieved from the database, formatted, and written to the output device.

The entire report generation procedure can be implemented by a call to the FileMan's print routine EN1^DIP, if the selection, sorting, and formatting criteria are not too complex. The other alternative is to write a hard-coded routine. Unfortunately, it is not easy to combine the two approaches (e.g., to use a FileMan sort template to pick the entries for printing and a hard-coded subroutine for advanced formatting).

Whether it is the DIP routine or a hard-coded one, the output procedure will ultimately issue *M WRITE* commands to produce the output on a device accessible by the server. The standard VA Kernel's Device Handler is capable of handling various printers and files of the host operating system. The latter can, for instance, function as a "pipe" to a World Wide Web service, in case the report is formatted as an *html* (Hypertext Mark-up Language) document. The Device Handler also allows the output to be automatically sent to a given email address as a MailMan message. For instance, our university's Publications Register produces departmental publication lists to the Web (see *http://www.uku.fi/english/* -->Research-->Publications), and our Student Administration system makes use of the MailMan output feature by automatically sending exam results to students by email.

In order to get the output to the client side, the easiest way is to first direct the output to a temporary sequential file on the server side (presented by a "tape cassette" in Figure 9; the VA Kernel's *Browser* device can also be used). When the output procedure is completed, a response message is sent to the client to acknowledge the completion of the output request and to inform of the name of the sequential file. The client software then invokes an appropriate display or printing component, which makes use of a sequential read component. The latter issues requests to the server through the Broker to send in the contents of the sequential file line by line.

If the report was readily formatted on the M side, it can be simply displayed in a memo window or directed to a local printer. Otherwise some Delphi Pascal code can be written, or an existing Delphi component used to present the data as a graph, for instance. The output can also be directed to a spreadsheet or some other OLE (Object Linking and Embedding) object.

The main problem with the above procedure is that the report generation on the server side takes place as a single, non-breakable step from initialization to completion. If the user, for instance, accidentally starts a procedure which produces a huge report, there is no easy way to cancel it while the client's output request component is waiting for a response from the server. A more interactive way is to use a pipe device between the server and the client (see Figure 9).

A pipe is a TCP/IP sequential output port on the server side and a sequential input port on the client side. When this alternative is used, the client side first allocates a free input port and sends the port number to the server side along with the output request. The report generation procedure can now be started as a background task and the client side be immediately informed of the completion of the request. The client then invokes the display or printing component, and the sequential read component starts to wait at the end of the pipe. When the report generation procedure on the server side proceeds to the output phase, it will start writing to the pipe device, and the data starts dropping through to the client. If the user now realizes that the data is not what she or he intended, s/he can push the *Cancel* button. The client software will close the pipe, the server process will get an error condition and terminate the task.

In summary, report generation is a complex procedure in a Broker-based client/server environment. The architecture presented above makes it possible to benefit from existing M software, including FileMan's standard sort and print templates. Delphi Pascal coding is needed for graphic and other special outputs which can be produced on the client side only. All other reports can be freely directed to either a server device or a client device.

We are just beginning the implementation of the architecture. Reports can already be generated by a Broker call to EN1^DIP or to a hard-coded M routine and transferred through a sequential file to the client for display. However, the start-up parameters and the pipe device will need more work.

4. The challenge of the World Wide Web technology

Hospitals in Finland still have more dumb terminals than PCs. Each university hospital has from 1,000 to a few thousand terminals in use. It is estimated that it will take at least five years before client/server applications can be installed in all parts of the university hospitals. Besides the new hardware infrastructure required, it will be an enormous task to develop or purchase new applications packages which make use of the client/server architecture. The tools and techniques presented in the previous sections of this paper are intended to benefit from existing databases and report generating software, but we readily admit that it is still not an easy task to modernize the legacy systems or develop brand new ones.

Yet experience indicates that the burden of hardware infrastructure and software development may be a small part of what it takes to train the users, manage the configurations and software versions of thousands of users, organize a help desk, etc. Client/server systems are known to require much more systems management work than traditional terminal-based systems.

During the last year, an alternative to PC clients has emerged under the slogan of *Network Computer* (NC). The new paradigm has its roots in the World Wide Web technology and the Java language. The idea is that instead of distributing specialized client software to all nooks and corners of a university hospital for instance, the clients should just run a standard Web browser capable of loading the specialized functionality from the network in small pieces ("applets") as and when needed. Thus, the ever more complex and "fat" PCs could be replaced by simple NCs, and the software could be centrally managed.

It may well be that by the time our hospitals have replaced all the terminals by PCs, it appears that they could have saved a lot of time and work by going straight to NCs or at least "Network PCs." Is it thus better to freeze the modernization of legacy systems along the client/server model and wait until the NC alternative has come true?

The NC model is still based on the client/server architecture and graphical user interface. Ideally, there needs to be no difference from an end user's point of view between a PC client application and an NC client application—only the software architecture is different. In our view, it is important to start modernizing the legacy systems with the kind of technology that is readily available, but be prepared for a change of technology within the next few years. The new technology can be based either on Java NCs or more probably on reduced-Windows Network PCs, but it will, in any case, be based on Web browsers as the user interface engine.

The main challenge to the FileMan world, in our mind, is to develop another functionally equivalent set of systems development building blocks in the Web technology in parallel with the Delphi-based ones (refer to Figure 1). As we now have FMComponents, output request components etc. in Delphi Pascal, we should develop functionally equivalent components for the Web technology. As we now have the standard building blocks for the basic file entry/edit/browsing functions in Delphi, we should develop similar building blocks for implementing the same user interface (Figures 4–7) on Web servers.

All the required features do not yet exist in the Web technology for the easy development of secured and efficient data entry functions. However, we are embarking on developing the Web components in parallel with the Delphi components. When the Web/NC alternative becomes mature, it should be a reasonably limited task to convert the applications to the new technology; in the same way as we now plan to convert our experimental HyperM applications to Delphi. All the server software behind the Broker interface will remain intact.

The Web/NC challenge does not need to be another enormous re-development task to the FileMan world. However, an easy transformation requires that the client/server applications are now composed from as high-level building blocks as possible in a consistent manner, keeping the forthcoming alternative technology in mind.

5. Conclusion: Whither applications development with FileMan?

VA's RPC Broker and FMComponents provide completely new prospects to modernize legacy systems based on FileMan databases. Our experience thus far is that impressive and efficient systems can be developed with this technology. For maximum productivity of systems development, standardized functionality and higher-level building blocks are also needed. Applications developed in such a way for one GUI technology can be fairly easily converted to another, retaining the touch and feel of the user interface.

We are developing a set of tools and standards for the Finnish FileMan users based on Delphi, Broker, and FM-Components. We call the tool kit *FixIT* ("Is your hospital's Information Technology worn out? Then FixIT!"). We have also experimented with developing functionally equivalent building blocks in HyperM, successfully. If necessary, it is possible to complete both the Delphi and HyperM building blocks and thus provide PC users with a fully modern interface and the terminal users with a somewhat more restricted interface, without duplicating all the programming efforts. The same approach is followed in developing another set of building blocks for the Web technology in a couple of years. The RPC Broker and FMComponents were officially released in November 1996. For about a year before that, our development work was based on preliminary information and later on an early evaluation kit. It is a key issue to the Finnish hospitals and software houses using FileMan that the development efforts in Finland and the USA will be more closely coordinated in the future. Much of the functionality developed and to be developed in Finland, the "special lookup component" and the report generation components for instance, might be relevant to the users of FileMan applications in other countries as well. We are, therefore, very happy that the contacts between the American and Finnish development teams have recently been brought to an official standing.

There is a lot of work to be done before systems developers have a complete tool kit in Delphi and still a lot more before the same is available in a Web browser technology. By coordinated action and division of labor between various development centers, the duplication of efforts can be avoided, and the future appears bright for client/server applications development with VA's FileMan, even on the Web technology.

M

References

Borland. 1995. *Delphi User's Guide*. Scotts Valley, CA: Borland International.

Karvinen, K., Korpela, M., Ruonamaa, H. "Rejuvenation of legacy systems: The case of M/Kernel-based hospital information systems in Finland." *M Computing* 4, 1, (1996): 9-14.

VA. 1996. *RPC Broker User Manual*. Version 1.0. San Francisco, CA: Department of Veterans Affairs.

The University of Kuopio Computing Center introduced M Technology in health informatics in Finland in the early 1980s and the FileMan/Kernel technology a few years later. Dr. Mikko Korpela is head of Systems Development and a researcher on health informatics at the university.
Email: Mikko.Korpela@Uku.fi
Http://www.uku.fi/~korpela

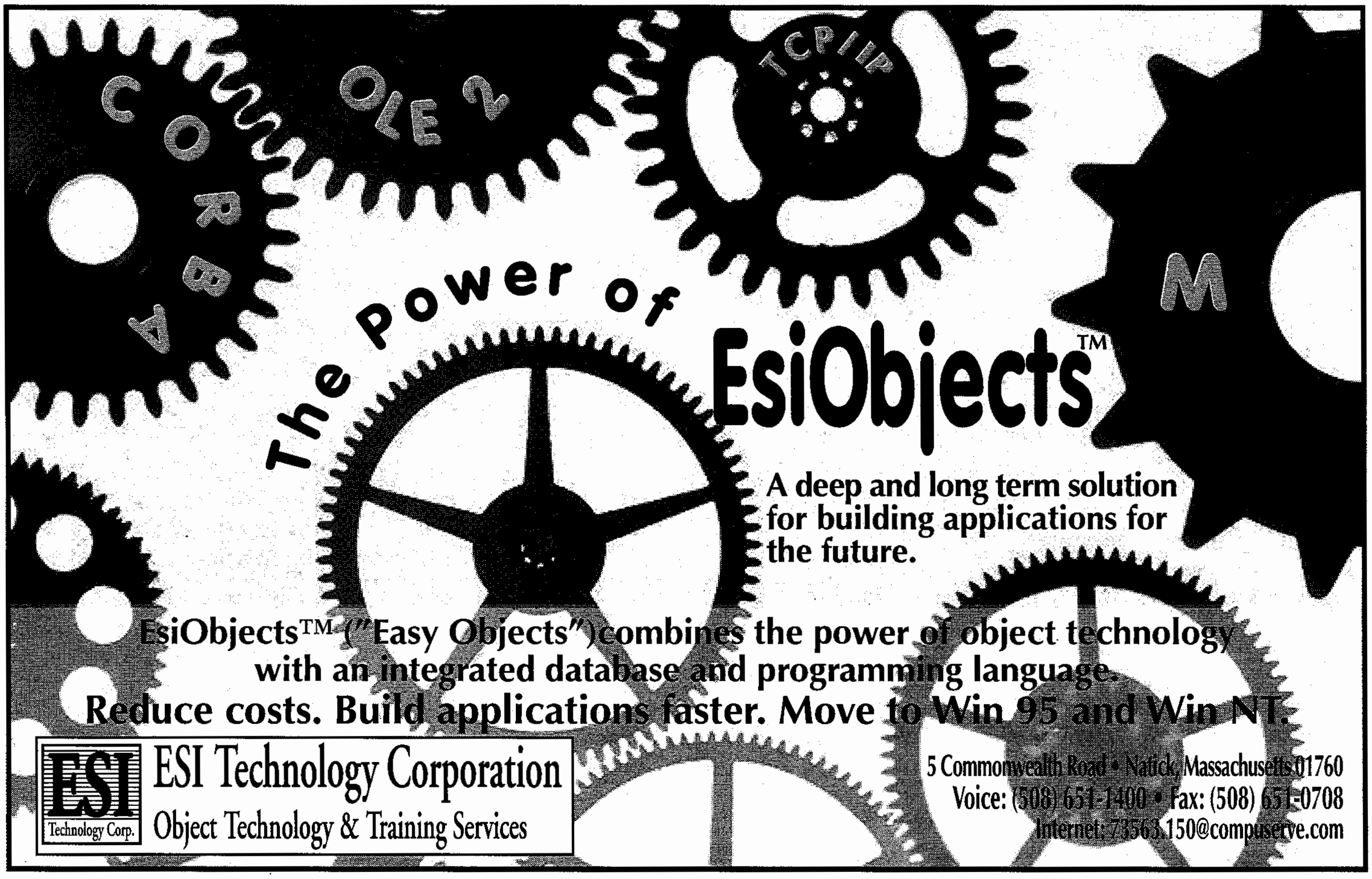
Mr. Mauri Kaatrasalo implemented the FixIT components as his M.S. work.

Ms. Hellevi Ruonamaa, M.S., has translated FileMan and Kernel into Finnish and is now in charge of FixIT support.

M Professionals Wanted

Contact MTA's

Job Referral Service



CORBA
OLE 2
TCP/IP
M
The Power of
EsiObjects™
A deep and long term solution for building applications for the future.
EsiObjects™ ("Easy Objects") combines the power of object technology with an integrated database and programming language.
Reduce costs. Build applications faster. Move to Win 95 and Win NT.
ESI Technology Corp.
ESI Technology Corporation
Object Technology & Training Services
5 Commonwealth Road • Natick, Massachusetts 01760
Voice: (508) 651-1400 • Fax: (508) 651-0708
Internet: 73563.150@compuserve.com