Street Lamps and Lost Keys

by Don Gall

Preface

There is an old story about a man seen crawling around on his hands and knees under a street lamp. A crowd began to gather, and finally someone asked him what he was doing. He said he had lost his car keys, which caused several of the more compassionate bystanders to join in the search. After all, it shouldn't be difficult to find a set of car keys in a small well-lighted area. After a thorough search, someone finally asked the man if he was sure he had dropped his keys here. The man then sheepishly admitted that he had lost the keys down a nearby dark alley. Then why was he looking here under the street light? Well, there was more light here!

The story may not be very funny, but it is allegorical. Over the centuries, we have built entire areas of mathematics and whole industries to help us find things under a street lamp which were lost somewhere else.

Linear Ordinary Differential Equations

In my earlier years, I spent a lot of time studying and then teaching students how to solve linear ordinary differential equations. There was a period when I believed that the LaPlace Transform would solve most of the problems here on planet Earth. I subscribed to the John von Neumann thesis that, given a sufficiently big and fast computer, we could determine the effect that a butterfly flapping its wings in California would have on the weather in Boston.

In graduate school, one of my first real world computer problems was helping to design a device which would reproduce one half of a cycle of an internal combustion engine at the Sloan Automotive Laboratory at MIT[1]. The device would take a piston from the bottom of its cycle to the top with its velocity closely approximating the first half of a sine wave. The device used a compressed air cylinder as its power source. For obvious functional and safety reasons, it had to come to a complete halt at precisely the right distance from the tempered glass cylinder head. The device was to be used with a high speed movie camera to film the combustion process. The only thing that was linear about the equations which described the dynamics of this process was that the mass of the piston was constant. The equations were (gasp) horribly nonlinear! So much for LaPlace.

My engineering career went from there to the analysis of instability in hydraulic servomechanisms caused by stick-slip friction (horribly nonlinear) to the six-degree-of-freedom simulation of the maneuvering control of a submarine in the ocean (also horribly nonlinear and random). Very few of the real world problems that I worked on fit the linear model. Was I just unlucky or is the world inherently nonlinear? I could never manage to lose my keys under a street lamp.

What good is linear theory? If the problem that you are working on fits the linear model, you are home free. The more the problem diverges from linear theory, the less valuable it is to you. The linear model allows us to get a firm understanding of one part of the world. This, in turn, enables us to extrapolate towards the quasi-linear and make approximations to the actual solutions which we otherwise would not be able to do. The major message is that if it is a linear system, use the theory and the tools; it will make life simpler and easier for you. If the equations are not linear, find other tools or other ways to solve them checking your results with the linear theory whenever possible.

The Development and Evolution of M

By now, you are probably asking yourself, What has this got to do with M?

For those of us who were attempting to build medical databases in the mid-60s, there were no street lamps. We had FORTRAN, ALGOL, BASIC and (God forbid) COBOL. When I joined the Department of Surgery at the University of Pittsburgh Medical School, one of my first tasks was to create a database of all of the open heart surgery cases. We ended up with a system written in BASIC on a DEC computer. Each patient data were stored as a large ASCII string in his or her own separate disk file. There was a master file which contained the file names of the individual patient files.

The good news was that the department was doing only 4 to 6 open heart cases per week. The bad news, well, with the tools we had at hand, almost everything was bad news.

By the early 1970s the word about a language called [M]UMPS had gotten as far west as Pittsburgh. (As a his-

torical note for all of you young programmers, my understanding was that an itinerant pots and pans salesman had made it across the Alleghenies in a blinding blizzard with the message.)

The first version of M[UMPS] which I was able to use allowed a choice of 26 global names (^A through ^Z). It ran on a PDP-11/something-or-other computer with 12K of memory and somewhere around 160K of something that resembled a disk. Everything but the amount of memory for that computer has disappeared from my memory. I recall the amount of memory, because I know it cost us $4,000 to upgrade it from 12K to 16K. (It was that cheap because it was previously used memory! New memory obtained directly from DEC was over $7,000.)

Even this early version of M[UMPS] represented an incredible advance over the other computer languages that were available for database management at that time. As M[UMPS] evolved, it added increased functionality including the major step of the current B-tree global management structure.

For the past 15 or so years, we in the M[UMPS] community have had at our disposal a very powerful and efficient language and database management system which allows us to model real world systems storing the data in any highly intelligent or ridiculously stupid structure that we can imagine. What have we done with all this power? If there is one common thread running through the M community, it is probably a tendency to create data structures which best fit the problem at hand. Why not? It is easy to do. It gives good storage and retrieval efficiency. If the structure doesn't lend itself to a report writer, so what; it is easy to write hard-wired reports in M. The three most common comments about M software packages are:

1. They were developed faster than it was thought possible.
2. The input and review screens are not as jazzy as they should be.
3. The reporting capability is not all that good.

As in all communities, the M community feels better when surrounded by others who think as they do. It amazes me to hear a group of COBOL programmers postulating that since there are more lines of COBOL code in existence than any other single language, COBOL should be made some sort of required standard language. (I must admit that if I were as prejudiced against minority groups as I am against the COBOL language, I would be in jail now!) On the other hand, I hear people in the M community saying that the Window technology is wasteful and not important, that M has little to learn from the inferior relational database technology and that SQL isn't efficient enough to be of use.

Relational Database Technology

In many ways, the relational database model is analogous to linear system theory. Both have an abundance of applicable mathematics for support. Neither always represents the real world. Although M can easily model structures which the relational structure is poorly equipped to model, the relational technology has one major advantage over M—the extensive and robust technology that has been developed around it. There is no way the much smaller M community will overwhelm this extensive and widely-used technology. We need to take advantage of this technology in areas where we can and circumvent it in areas where it is not beneficial.

James Martin, in a recent book[2], makes some very interesting comments on the relational technology model. In Appendix C of this book, Martin emphasizes the need to avoid the relational model for complex data structures or applications which have much data connectivity, in order to improve performance. He cites examples of one and two orders of magnitude increase in performance of non-relational over relational databases. He also points out the advantages of physical data clustering which can be obtained by not using relational database structures. Martin stresses that more sophisticated database structures are needed and that they will coexist with relational database structures rather than replace them.

About seven years ago, our M development efforts were shifted towards the use of object-oriented programming and relational database techniques and tools[3]. The methodology we developed used data-typed attributes to define relational views of an M data dictionary. We have found that about 98% of our data could be defined as normalized relational database structures and simultaneously stored efficiently as clustered hierarchical M structures.

Because law firms have been slow to adopt Windows methodology, we decided to develop our initial object oriented programming version with a character user interface. With less than 2 man-years of programming effort using Borland Delphi development software, we were able to also create a Windows interface which used the original database engine. This gives us the capability of running character and graphical user interfaces simultaneously on separate client computers with both accessing the same data.

If we had been able to use the Delphi data-aware controls which would allow Delphi to communicate directly with our M database, our efforts would have been significantly reduced. Instead, we had to write our own database interfaces, many of which relied heavily on existing M routines, to populate Pascal data fields from the M database. If we had had an ODDBALL driver which could link to the Delphi data-aware controls and which could either use M code to efficiently access the M data or be able to choose from mul-

tiple views to efficiently access the M data, our Windows development would have been much faster and easier.

In a similar fashion, we need improved back end tools which will enable us to provide relational views of our not necessarily relational data so that these data can be utilized by many of the increasingly sophisticated reporting packages such as Report Smith and Crystal Reports. These back-end tools need to rise to the level of sophistication of the report writers they interface with. This reporting capability should be viewed as a fast and efficient standard rather than as something to try if you don't have the time to create a hard-wired report.

Summary

The M community needs to develop new and efficient ways to interface to the much larger world of relational database technology and the tools and methodology surrounding that technology. High on our list should be the development of products to enable us to more efficiently link M databases to products such as the Delphi data-aware controls. We need to improve our ability to link to the many SQL products to enhance the general reporting capabilities in our systems.

We in the M community have in front of us a rare opportunity. Many of the things that the leaders of the object-oriented database movement would like to do with databases can be done readily by M. We need to start viewing M as an addition to and an improvement on the existing relational technology. There is no reason to try to turn M into a relational database nor is there a reason to expect M to ever replace relational databases. If we work very hard, M as a language and an object-oriented database will have an opportunity to coexist with, improve the performance of and expand the horizons of relational databases. **M**

NOTES

1. White, Pepper. 1991. *The Idea Factory.* New York, NY: Dutton-Penguin Books USA. (see for example page 178.)

2. Martin, James and Joe Leben. 1995. *Client /Server Databases - Enterprise Computing.* Upper Saddle River, NJ: Prentice Hall PTR.

3. Gall, Don. "An M Implementation of Object-Oriented Programming." *M Computing* 3, no. 1 (1995): 13-19

Don Gall is CEO of Omega Legal Systems in Phoenix, AZ and a member of the MTA Board of Directors.
