# Using M to Navigate Multiple Step Clinical Algorithms Over Time

*by Marilyn D. Paterno, BS, B.Mus.; Rita D. Zielstorff, RN, MS; Mark Segal, MS; Jonathan M. Teich, MD, Ph.D.; Gilad J. Kuperman, MD, Ph.D.; Roberta L. Fox, MS*

*[See Discussion Session by the same title in the Strategies and Solutions program on Monday at 4:00pm.]*

## Abstract
Clinical algorithms provide step-by-step instructions for clinicians caring for patients with a specific problem. We have designed and implemented a multi-step logic processor that evaluates incoming data, makes recommendations, and manages the algorithm over time. Though designed for a health-care environment, it is not limited to medical settings. This article will focus on two aspects of the application: its platform-independent object design, and the M engine that drives it.

## Introduction
An algorithm in its simplest form evaluates data and returns a result. In our setting, the data may be provided either by a user in the form of an interactive on-screen questionnaire, or from the database. An example of this might be a questionnaire, which an emergency department clinician completes; the algorithm returns a recommendation of whether to order x-rays. The algorithm in this example is completed in a single session. This means that data is gathered, evaluated, and the recommendation returned while the clinician waits at a workstation, then the algorithm ends. In that aspect it is typical of most computerized algorithms or clinical alerting systems in place today. We were presented with the task of providing more complex algorithms: for example, a guideline for the treatment of hypercholesterolemia, which needs to evaluate decisions made by the clinician, current therapies already in place for the patient, and the results of laboratory tests performed over time. Typically, such an algorithm is represented as a flowchart, with multiple decisions that must be made at various points in the process. We did not find any implemented examples of algorithms that can run as complex a treatment guideline as we required, although work is being done on their representation. Therefore, we decided to define and write our own [1].

In 1996 when we began the project, all data was stored in M globals and processed with routines written in DTM. Since data in this environment is accessible to all applications, the development of decision support systems is a great deal easier than it would be in other, more disparate systems. A new platform was being developed for our organization, however, with a three-tiered architecture and a new user interface. That led to the partitioning of application components into three logical groups, or services: user interfaces; logic processing; and data retrieval and storage. To facilitate the development of the algorithm project we decided to remain in the integrated DTM environment in order to take advantage of the large dataset and applications available, but to write separate modules for each component. In this way, we could be prepared to move one or more components to the new platform as it became feasible.

## Object Design
Drawing on the representation work of the InterMed Collaboratory [2], and using standard logic flowchart conventions, we deduced that two types of steps are needed to complete a complex algorithm: a "decision step" to evaluate rules, and an "action step" to perform all other tasks, including running questionnaire programs, sending messages to the user, and waiting for future events to occur. Some type of branching information is also necessary in order for the algorithm to know where to go when each step is completed.

At the time we designed the project, we did not have available to us any object modeling tools within M, and used a commercial diagramming tool to produce three models. Following the guidelines Rumbaugh [3] provides, we developed a problem statement (see Fig. 1), from which we expected to derive a list of objects for the algorithms. Using the problem statement, we isolated potential objects from nouns within it, associated verb phrases with each object, and proceeded to develop the model. The object model (Fig. 2) contains classes with attributes (properties) and operations (methods).

**Problem Statement:**

*In the course of treating patients for a given condition, there are preferred courses of action which have been shown to be effective. When a patient presents with a condition for which such a guideline is available, the computer will present it as an option to be followed. Upon acceptance by the clinician, the guideline is started by updating an algorithm activity log for the patient, then calling the first of n steps, each of which performs one or more tasks, updates the algorithm activity log, defines the current state of the algorithm, and points to the next step. Tasks may take the form of (a) actions, possibly requiring a response, and (b) decisions, which evaluate rules in order to determine the next step. A rule includes one or more conditions to evaluate and is processed by an event engine. Actions may include (1) programs to be run, such as a questionnaire presented to the clinician, (2) messages to the clinician, such as a recommendation with suggested order sets, and/or (3) events which must take place before the guideline may continue, such as the availability of required data or the passage of time. A dispatcher sends rules to the event engine to evaluate the events and returns results to the guideline. The presence of such an event in a step places the guideline in a 'wait' state until the result returns. Notification of a message's presence may be via direct screen intervention, email, pager, or update to a general message handler. What steps are to be called next may be constrained by the result of a rule evaluation, whether or not a user is present, and/or the necessity for synchronization with other steps. When the steps followed reach the end, the guideline is completed, and the algorithm activity log is updated with a 'closed' state.*

Figure 1. Problem Statement

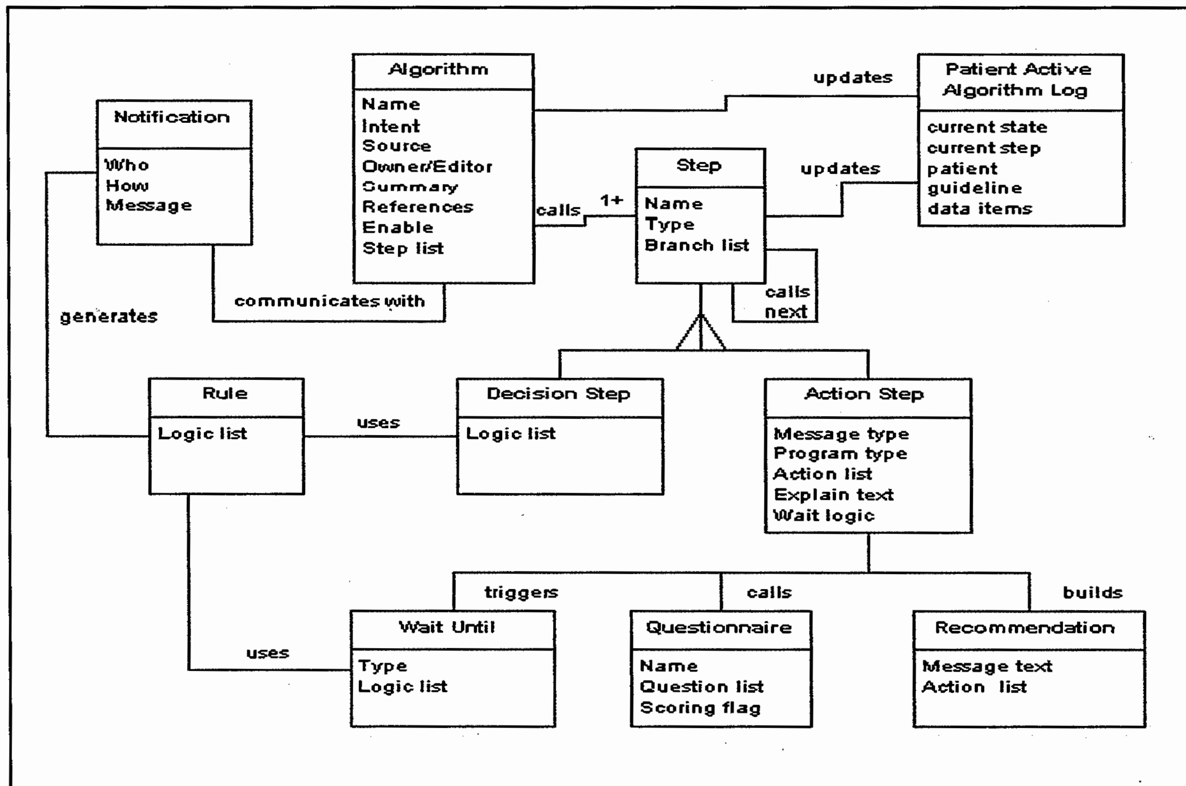



Figure 2. Object Model (some details are omitted for clarity)

The functional model, or data flow diagram (Fig. 3), displays the project's data stores, actors, data flows, and processes. Although this model shows the functions that move data, it does not place them into any time frame or relate the flow to temporal events. This model allowed us to understand the flow of data first, which in turn determines what needs to be written to produce each movement. Having completed it, we were able to go back and revise the object model, inserting operations that would be needed into the proper class descriptions, and adding attributes as they were discovered.

Time is shown by the dynamic model. We developed this model in three parts: first, we wrote scenarios describing events as they occur; next, we traced the events in time, putting each in its proper chronological order; last, we determined the state of the class at each event. For example, when data is outstanding, such as a lab result not yet filed, the step may be said to be "waiting". From this we created a state diagram for the step class, and one for the algorithm class. This proved to be quite helpful when we designed the patient-centered algorithm activity log that would contain all information on the running of an individual algorithm. We store with each step a flag indicating the current state of that step, which is used to determine the state of the algorithm.

The models went through several iterations, as our understanding both of our application and of object methodology developed. Although each model can change over time, the overall design is stable. When enhancements are proposed, we revisit the models to

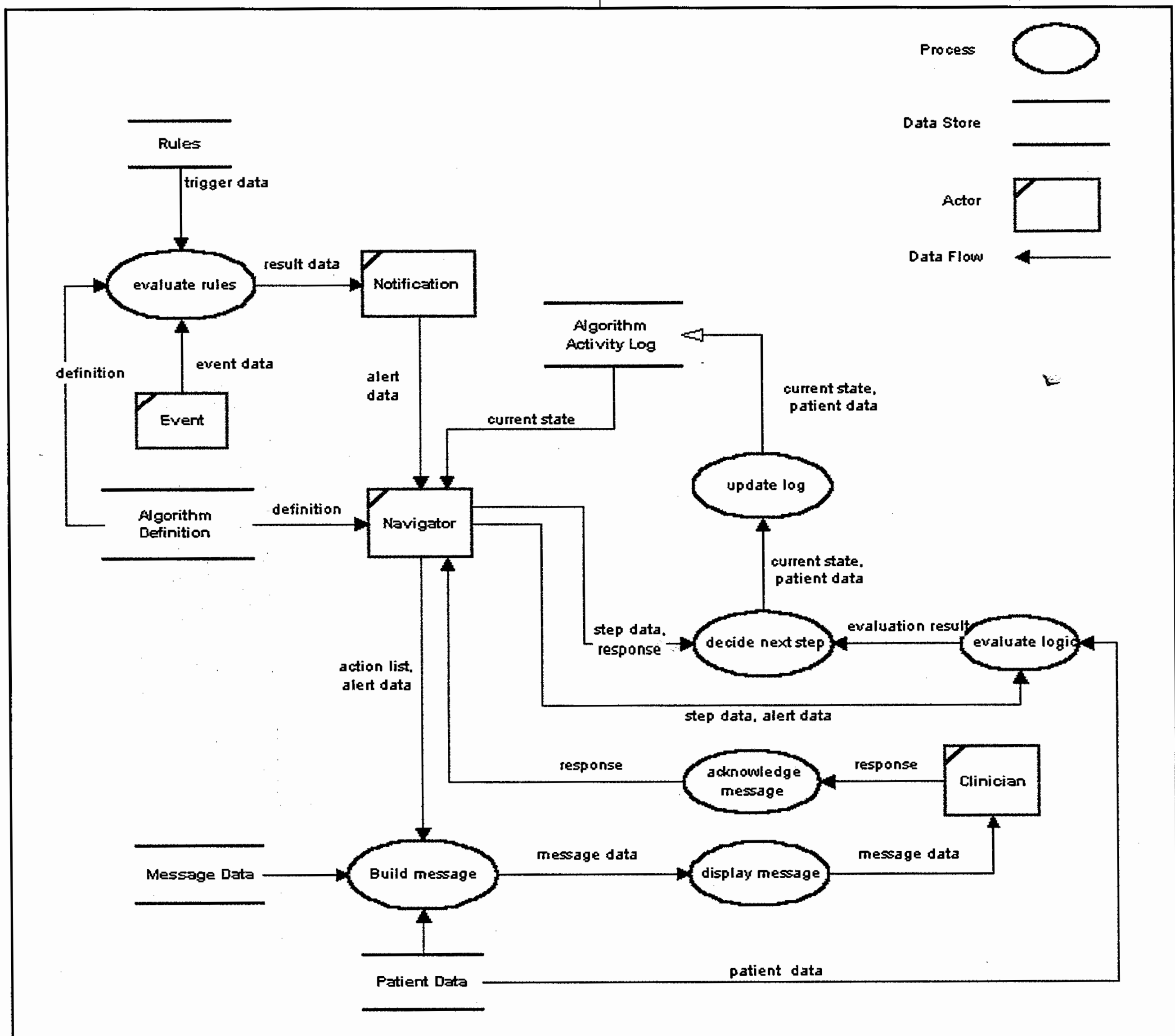



Figure 3. Data Flow Diagram

determine the impact of the enhancement on the entire project.

## Processing Engine

Algorithms are processed by a set of routines that we call the Navigator. An algorithm session may be initiated with a request from the user, or when an event occurs that triggers the creation of a new or resumption of an existing algorithm. For example, the hypercholesterolemia algorithm may be triggered when a new cholesterol result is filed for the patient. If an event initiates the session, the Navigator may notify the clinician that the algorithm is available for initiation or that a session has been initiated, using one of the means described in the problem statement.

The Navigator operates in two modes: interactive, where the user is present at the computer; and background, or non-interactive, where the user is not present and must be notified when changes occur in the algorithm state. Instances of algorithm logic which need to be evaluated, including decision step logic and time-triggered events, are stored as rules and evaluated by an inference engine which is already in place in our organization [4]. This engine returns result data in a message to the Navigator. As it processes steps, the Navigator logs changes in state along with the data affecting or affected by such change, sends messages to the user, receives and acts on user responses, requests logic evaluation from the inference engine, and determines the next step. A session ends when the last step has been processed, the user elects to remove the patient from the algorithm, or a wait state is reached. Wait states occur when the current step needs additional data not presently available in order to complete its task. A wait state may end automatically after the passage of a specified period of time, when an event occurs which sends a message to the algorithm, or when the user returns to the computer and invokes the algorithm program again. Figure 4 illustrates an algorithm that includes wait states, triggering events, decision and action steps.

The Navigator is comprised of six modules. These con-

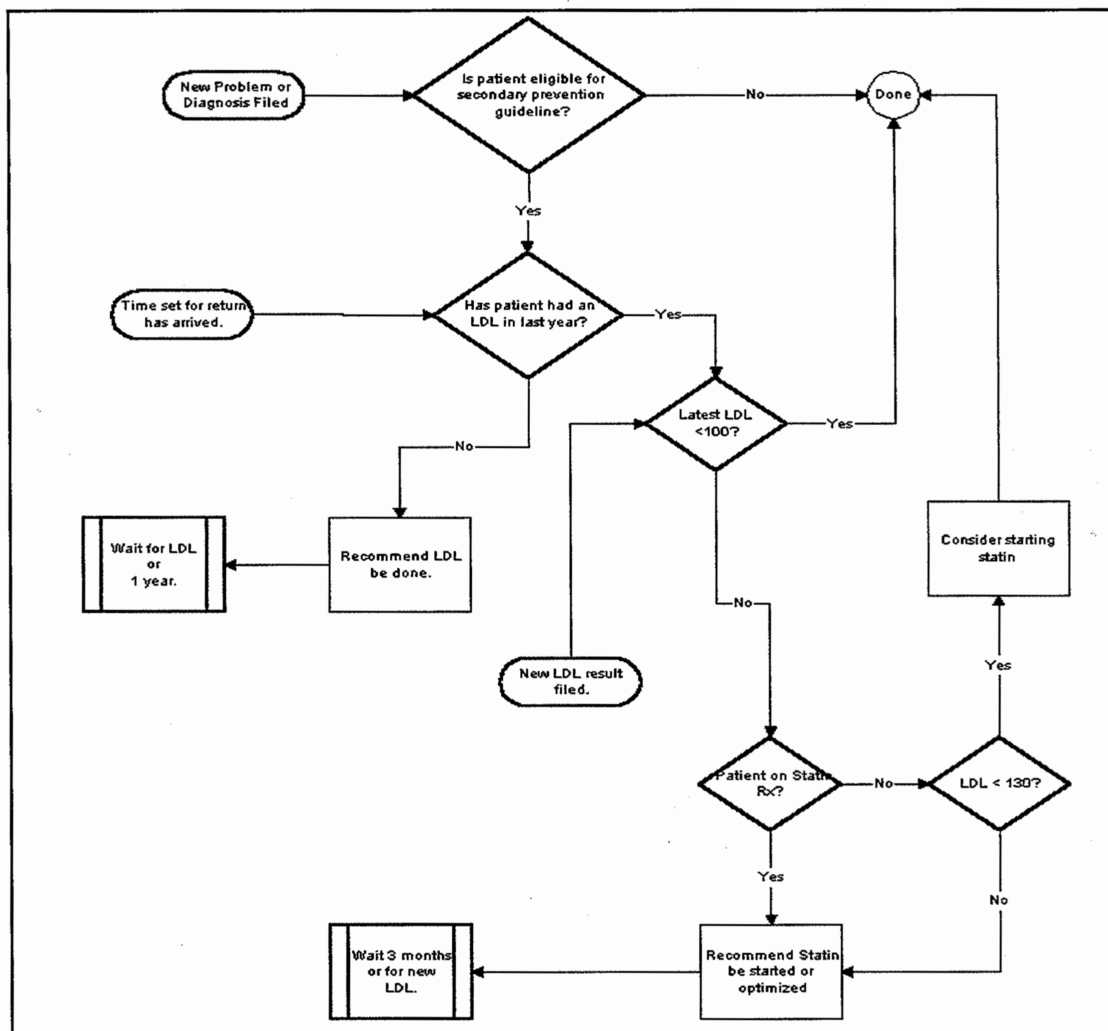



Figure 4. Glowchart Example: Guideline for Secondary Prevention of Cholesterol (reduced for space).

trol its flow, load data, process tasks, call messaging functions, run programs, and file the data to a patient-specific algorithm activity log. The controlling routine sets up the environment, determines the current step to be processed, and calls the load module to retrieve the algorithm definition and log data for that step. It then calls a recursive subroutine, NAVIGATE, which receives the current step identifier as its only parameter.

NAVIGATE calls the task module, which creates a list of tasks, performs them, and evaluates the results. There are four types of tasks: messages, programs, rules, and wait actions. When creating the task list, the task module determines if a user is present or needed, bypassing tasks that require user intervention when none is present. As it $Orders through the task list, it calls messaging functions and/or programs as they are needed, passing in a data array and storing results returned locally for later filing. It passes rules to the inference engine for evaluation; last of all it looks for wait actions which need to be triggered. Before returning to NAVIGATE, the task module determines the current state of the step it has processed, and also that of the algorithm.

Once the task module is finished, NAVIGATE determines the next step, calling the load module to retrieve its definition data and set up a log record. When finished, it calls itself again at the top, passing the step identifier as its parameter (Fig. 5). If there is no step to be run, either because the algorithm has ended or has entered a wait state, the parameter is an empty string, and the NAVIGATE subroutine ends. The controller module then calls the filer to update the algorithm activity log, cleans up the environment, and quits.

```
NAVIGATE(CURSTEP)     ;
     S LOGN=$S(STR]"":"STR",1:"ALR")
     K @TASK
     D LOADTASK^ALGNAVTASK(LOGN,.TASK,.MESG)
     I '$$RUNTASK^ALGNAVTASK(.TASK) S STEPN=""
     E  S STEPN=$$WHERETO(LOGN,@LOG@(@LOGN,"STATE"))
     I STEPN D NAVIGATE(STEPN)
     Q
```

Figure 5. NAVIGATE subroutine

## Lessons Learned

The algorithm project has been successful, and several clinical practice guidelines have been created and activated. Version 1 of the application did not include time delay processing or non-interactive notification. In version 2 we have added eligibility triggering of algorithms by outside events as well as time delays which wait for events to occur.

To use object design within a language that is not object-oriented takes a great deal of discipline. One cannot simply begin writing program code. While we cannot reference objects with a specific object syntax, we can define them concretely, provide a standard method of access, and see clearly what pieces need to be isolated. As an example, instead of including in the same M routine data access code, data manipulation code, and data entry code, we need to separate those functions, so that each can be called by an independent process, passing along needed parameters.

## Conclusions

Looking back over three years spent in system design, object modeling, programming, testing, and running clinical algorithms, we have reached two conclusions: first, that the time spent on object design was well worth the effort. It increased our understanding of what we were doing, forced both programming and non-programming designers to communicate clearly, and clarified the issues so that programming could be accomplished that would support future change as well as present use. Although it would have been nice to have technology available within our M environment to assist in the design, we expect to be able to redefine the objects from our models using such technology now that it is beginning to be available to us.

Our other conclusion is that the isolation of M functions is the single most important programming step we took. In the past, when M was operating system, user interface, data storage, and programming language, it did not matter very much that all its functions were in the same routines, even within the same code lines. By keeping data access separate from user interface, and logic processing separate from both, we can continue to use M where the environment supports it, and can call the other functions where it is necessary. At the present time we are still using M for all parts of the application; data access, user interface, and logic processing. Changes are expected to take place, the first most likely to be the user interface. Isolating calls to the screen within the processor will make this task easier, and by standardizing the data formats we will be able to use more than one interface during a transition process if needed. Although the current database is not planned to move from M globals, it is possible that over time there will be additional data sources in use. We think we have allowed for that possibility by

*(continued on page 18)*

*(from page 16)*
separating the data access functions into a separate load routine, into which we can insert external calls to load data from another source. By isolating messages and programs as functions, the Navigator does not need to concern itself with the user interface being used. By isolating the load and setup of data arrays into separate routines, they may be modified to make calls to other data sources without the Navigator being concerned.

## Endnotes

[1] Zielstorff, R.D., Teich, J.M., Paterno, M.D. et al., "P-CAPE: A high-level tool for entering and processing clinical practice guidelines", *Proceedings of AMIA '98 Annual Symposium*, Chute, C.G., ed. (Hanley & Belfus, Inc., Philadelphia 1998) 478-82.

[2] Ohno-Machado, L., Gennari, J.H., Murphy, S.N., Jain, N.L., Samson, W.T., Oliver, D.E., Pattison-Gordon, E., Greenes, R.A., Shortliffe, E.H., and Barnett, G.O. "The GuideLine Interchange Format: A Model for Representing Guidelines", J. American Medical Informatics Assoc. 5(4):357-372, 1998.

[3] Rumbaugh, J., Blaha, M., Premerlani,W., Eddy,F., and Lorensen,W. *Object-Oriented Modeling and Design* (Englewood Cliffs: Prentice-Hall Inc., 1991).

[4] Kuperman, G.J., Teich, J.M., Bates, D.W. et al. "Detecting alerts, notifying the physician, and offering action items: a comprehensive alerting system", *Proceedings 1996 AMIA Annual Fall Symposium*, Cimino, J.J., ed. (Hanley & Belfus, Inc., Philadelphia 1996) 704-708.

---

Email (principal author): mdpaterno@partners.org

*The Clinical Algorithm Project was written using the Brigham Integrated Computer System (BICS) and runs at Brigham and Women's Hospital and its affiliated ambulatory practices, in Boston, MA. Marilyn Paterno is Team Leader for the Clinical Systems Research and Development Group at Partners HealthCare System, Inc., Boston, MA, and serves as Technical Lead on the project.*

*Other members of the development team are Rita Zielstorff, Manager, Mark Segal and Roberta Fox, Systems Designers, and Saverio Maviglia, Clinical Research Fellow. Jonathan Teich is the Director of Clinical Systems Research & Development. Gil Kuperman is the Manager and principal architect of the BICS Clinical Alerting System.*

---

## Advertiser Index

We appreciate these sponsors of the September issue and all the companies who support the M community through their commitment to excellence.


| Advertiser | Page |
|---|---|
| Career Professionals Unlimited | 25 |
| CyberTools, Inc. | 25 |
| ESI Technology Corporation | 35 |
| George James Software, Ltd. | 45 |
| Henry Elliott & Company, Inc. | 1<br>Back Cover |
| InfiniMed, Inc. | Inside Back Cover |
| InterSystems Corporation | 47 |
| Jacquard Systems Research | Inside Back Cover |
| KB Systems, Inc. | 18 |

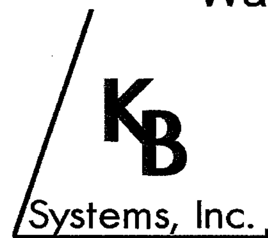

*This index appears as a service to our readers. The publisher does not assume any liability for errors or omissions.*


# KB_SQL Version 4.0!

The proven SQL/ODBC solution for all M types, including Caché

- Full-featured reporting environment
- Works with Crystal Reports, Microsoft Office, and more
- Supports updates of your M data via SQL commands
- Seemless integration with VA Fileman databases

Want to know more?




www.kbsystems.com
**KB Systems, Inc.**
Voice (703) 318-0405

©1999 KB Systems, Inc.
All products are registered trademarks of their respective companies.