# DATA REPRESENTATION AND MODELING USING GLOBAL MAPPING

William J. Harvey, Ph.D.

Institute for Information Management and
Department of Computer and Information Systems

Robert Morris College
Coraopolis and Pittsburgh, Pennsylvania

## Abstract

M associative global arrays provide a platform for data management. Database implementors map logical structures onto global arrays. This paper presents a survey of the role of global mapping in supporting custom database designs and database systems. The paper focuses on current topics, such as using global mapping in relational database systems, in SQL access to FileMan databases, and in object data management.

## Introduction

M global arrays of *n* dimensions are named *global* because all qualified users in an M system user class can access these arrays. By comparison, *local* arrays are dependent on the execution of a single program, like variables and arrays in most programming languages. Global arrays (globals) provide persistence for applications based on M by hiding all details of file access and indexing support (such as B-tree pointer management). M is listed as one of the languages with a model of persistence in Section 2.8.1 (Areas for Standardization) of the Reference Model for Object Data Management developed by the ANSI Object-Oriented Database Task Group.[1] Arrays are traversed and checked by using standard (ANSI/MDC X11.1 1990, FIPS 125, or ISO 11756) functions. You can find early treatments of M arrays in the references cited in the notes section included with this paper.[2,3] M arrays have the following advantages:

- o the concept of the M array (subscripted variable), as a platform for data management, is relatively simple
- o the M array offers a high level of automation of persistent data storage
- o multilevel associative indexing
- o within the context of a single language design, there is no impedance mismatch between database access and procedural programming

M arrays support custom database designs and their accompanying query and retrieval systems. Each database implemented in an M environment has a system of mapping logical structures onto global arrays. Database systems based on M incorporate mapping strategies and data definition utilities. Such data definition facilities correspond to the data definition capabilities provided for standard SQL and NDL databases. Everest et al. include a set of "commonly accepted semantic data modeling constructs" in the introductory section of their comparative treatment of "data structuring concepts" underlying the ANSI NDL and SQL standards; entities, attributes, relationships, and identifiers.[4] We can use these constructs to examine global mapping capabilities and representation of information models in M database technology.

General-purpose M database systems have sophisticated global mapping facilities for accomodating information models. Examples of such general systems in M environments are:

- o VA File Manager (FileMan)
- o SQL implementations, such as InterSystems' Open/MSQL™ and KB Systems' KB_SQL™
- o Digital Equipment Corporation's DASL™ product (Digital Standard MUMPS Application Software Library)
- o BAIK[5]
- o AIDA[6]
- o NOUS (also called GNOSIS)[7]
- o Relational systems using an interface other than SQL, such as the National Computer Systems' Trustware™ Report Generator, based on the QBE model
- o MUMPS Query Language (MQL)[8]
- o Object data management prototype systems, such as EsiObjects™ of Educational Systems, Inc. or MOOPS™ of MGlobal International, Inc.
- o DBMpS[9]

These systems utilize code generation and thus reduce or eliminate ad hoc procedural coding.

The VA FileMan database management system, which is based on M, has automated support of abstract data types. VA FileMan can support higher levels of database representation. Lindner and others provide examples in the implementation of Ntelligence[10] and in Andrews' model of a "common data structure" for clinical data management.[11] In the same fashion as SQL databases, which store metadata in SQL tables, FileMan databases store metadata in FileMan files.

Global mapping achieves data definition by exploiting three properties of M arrays:

- o Hierarchical subscript paths
- o Sets of subscripts
- o Stored strings

Global variable subscripting produces a representation of sparse arrays, since only the necessary references are present in the array structure. The use of the term array here is different from the usual connotation of a dimensioned array with cells that exist whether empty or not. You can use a subscript path as an index and detect the presence or absence of paths nonprocedurally.

The lowest level of subscripting for a given reference inherits a parent pathway to the root node, a variable name, and each subscript node may have descendents. The subscripts with a particular common parent reference constitute a set of values.

Strings are stored at nodes of arrays. These strings may be the empty string (null string) and you may interpret all values in M as strings, numbers, or boolean values, depending on context. It is common practice to store data functionally dependent on a given reference as a delimited string or a set of such strings.

Data definition in M databases specifies global variable subscripting paths and string structures. Automated data dictionaries document the semantics of the structures as well as the structures themselves. Data definition treatment of strings provides description of delimiting techniques. The delimited elements are called *pieces*, since you use a standard M $PIECE function to manage the delimited strings. Global mapping data definition facilities in different systems have in common the specification of array reference (variable name and subscripting pattern) and, if necessary, delimiter used and piece position.

Use of M data management technology in heterogeneous database systems requires attention to equivalence of M and non-M databases and thus a global mapping logic.

## Custom Data Representation and Modeling Using Globals

Custom database design in the M environment involves mapping entities and relationships to arrays. Consider a simple use of an M array. In a department store, garments are on sale at a discounted price; discount rates are specific to the individual item and assigned at the time of markdowns. Example 1 shows a way to represent these garments, assuming more than one discount amount or method exists and a single amount or method is applied to any given garment on sale.

| Example 1 |
|---|
| `^ONSALE(GARMENT)=DISCOUNT` |

Note the following about Example 1:

- o GARMENT is a unique garment identifier; DISCOUNT is the discount percentage or possibly a code for the markdown method for a particular garment
- o ^ONSALE(GARMENT) evaluates to the applicable discount or discount method
- o GARMENT in ^ONSALE(GARMENT) is a subset of GARMENT in an inventory which could be represented by ^INVENTORY(GARMENT).
- o Array ^ONSALE is equivalent to a file with indexing, but non-procedural access is possible to determine conditions through syntax such as IF $DATA(^ONSALE(GARMENT)), where $DATA is a standard function that checks array tree structures.

Example 2 shows a different programming convention, but conveys the same information as Example 1.

| Example 2 |
|---|
| `^ONSALE(GARMENT,DISCOUNT)=<null>` |

According to the convention in Example 2, $ORDER(^ONSALE(GARMENT,"")) evaluates to the applicable discount.

In Example 3 we have a different department store model. In this model we assume that there are various established sale categories and that discounts may vary by type of sale. We add SALETYPE to identify a particular category of sale.

| Example 3 |
|---|
| `^ONSALE(GARMENT,SALETYPE)=DISCOUNT` |

The data structure in Example 3 permits us to determine which discount is used for a particular type of sale, assuming that discounts may vary by type of sale.

If we imagine a database in which we record which sets of data employees are authorized to use, we require representation of a many-to-many relationship, as shown in Example 4.

Example 4

```
^AUTHORIZE(EMPLOYEE,DATASET)=<null>
^ACCESS(DATASET,EMPLOYEE)=<null>
```

EMPLOYEE and DATASET values uniquely identify employees and sets of data, respectively. We note the following in Example 4:

- ^AUTHORIZE allows us to determine which sets of data a given employee is authorized for
- ^ACCESS is an inverted index of ^AUTHORIZE and allows the programming an easy route to displaying which employees may access a given set of data

Standard M database traversal functions permit all employees in ^AUTHORIZE or all sets of data in ^ACCESS to be checked or reported.

These examples show that rule structures and constraints can be represented conveniently and exploited in operations. Much more complex logical structures are possible. Databases designed for a particular application usually have features similar to those of a general-purpose database system, such as a data dictionary, but lack the ability to add files not predefined.

## Relational and Other Database Systems

You can develop relational database support for operations within the M environment, based on the associative arrays. Both SQL and Prolog models have been used in developing relational databases in M. McIntosh presents a model for interfacing SQL and M.[12]

Yannakoudakis states that "it is as easy to establish relations from trees as to create trees from relational tables."[13] He uses the term *binding attribute* to refer to the attribute to use as the parent subscript (and thus as the key) in storing a relation as an M tree-structured array. What Yannakoudakis refers to as *alternative binding orders* represents the alternative implemented by setting up inverted global files (or subtrees) in M databases. Although indexes are redundant structures from the strict standpoint of the relational database model, and indexing is distinct from base tables and views, in M, indexes and inverted files must be established in the same kind of data structure as the array representing a base table. Indexed data can point to a stored string or substring rather than to a subscript or array node reference as with a foreign key reference.

Several approaches are used in designing M database systems. Among the options to be evaluated are:

- Using internal numbers and pointers or external names
- Using the approach to keys for hierarchical databases as proposed by Jacobs where all subscripts are keys in the relational database sense and all nonkey fields are stored as data at nodes[14]
- Storing all values as paths (storing only null strings as values at nodes)
- Combining key references where the key consists of more than one attribute into a composite, delimited string
- Using the order of subscript references
- Using a single tree or multiple trees

Example 5 shows the model of using the combination of subscripts as a primary key. In Example 5, a system using this model stores a delimited string record.

Example 5

```
^DATABASE(TNAME,KEY1,KEY2)=FIELD1^FIELD2
```

In example 5 the table TNAME has a two-subscript key. KEY1 and KEY2 represent the two key fields. Two nonkey fields, delimited by the ^ character, are stored for each record. No duplicate key values are possible in this model. It is necessary to store more than one delimited string for a given record if the physical length of a stored string might exceed the maximum number of characters supported for a string.

Database systems can add new items to an array by using a subscript representing the order of arrival, such as a sequence counter or timestamp, to identify each new record. The system then uses index arrays to find records.

Indexing is essential to gain optimal performance. Volkstorf identifies design strategies for globals and algorithms which reduce disk access.[15] Walters presents a survey of database optimization techniques with attention to M algorithms[16] and Middleton provides a focus on query optimization of databases based on M.[17]

Digital Equipment Corporation's DASL (DSM Application Software Library) has an SQL query driver. At a general level, the DASL user maps *data names* to M globals and creates a data dictionary. The user can then define tables for SQL access.

InterSystems Corporation's Open M/SQL product integrates the SQL and M standards and makes available embedded SQL programming in M with enhancements. Pantaleo outlines the way in which relational structures are supported through global mapping and a data dictionary[18] and points out that base tables in a relational system based on M are globals at the logical level accompanied by *index maps*. Open M/SQL links tables by using designative and characteristic references.[19]

KB_SQL (KB Systems, Inc.) is an example of an SQL product designed to run in standard M environments. According to KB_SQL documentation, the "data dictionary provides a relational view of your MUMPS globals."[20] The KB_SQL user has the option of standard SQL data manipulation commands such as CREATE and INSERT.

NOUS, developed at the MUMPS System Laboratory in Nagoya, Japan, provides Prolog programming embedded in M. The structure of globals in M facilitates the representation of nested relations, but NOUS requires an enhancement of the M global structure to support nested relational databases according to the Prolog model.[21] O'Kane proposed an alternative approach relations based on M.[22] O'Kane's "mapping of the relational database tables uses the MUMPS global array name as the name of the relation and the indices (in string valued form) as the column values." According to O'Kane, in "this approach the MUMPS Global Array data base is envisioned as collections of facts concerning a subject area. ...no data is stored at the terminal node of a path description."[23]

Information Builders, Inc. provides a method for describing DSM globals to the FOCUS™ database system.[24] The FOCUS file descriptions amount to a view determined according to user requirements.

Using a form of entity-relationship (ER) modeling, Milan and Major demonstrate how "a logical data model may be mapped onto M globals."[25] Milan and Major explicitly address the concepts of entities, relations, and attributes.

## FileMan and SQL

Dealing with SQL access to FileMan and other M databases requires treatment of the following:

- o Global mapping to support SQL access to FileMan and to other M databases
- o M global mapping in general
- o Automation of SQL global mapping for FileMan data dictionaries
- o SQL and FileMan naming and data types
- o Comparison of SQL and FileMan query capabilities
- o Referential integrity support

Among the problems encountered with FileMan global mapping is the nature of identifiers, such as file and column names in FileMan and table and attribute names in SQL. The following rules apply to an SQL standard identifier; it:

- o Must begin with a letter
- o Can continue with letters or digits, and may include embedded underscores
- o Can not be identical to an SQL key word
- o Can not exceed 18 characters.

Note that this paper uses the term *identifier* according to SQL usage.

The following rules apply to a FileMan file name:

- o Uses a free text data type
- o Must be 3 to 30 characters in length
- o Can not have a punctuation mark as the initial character
- o Can include embedded blanks

In some FileMan and SQL interface models, all FileMan embedded blanks are translated into embedded underscores in SQL and vice versa, although this practice is vulnerable to name conflict problems in the event that a translated name should be the same as a table name already created using the underscore character. Other special characters and problems involving the length of the identifier require special handling.

Duplicate .01 field values and duplicate records are permitted in FileMan, so an SQL base table derived from a FileMan file could not merely consist of the .01 column and all single-valued columns. FileMan records are unique, but that uniqueness is not dependent on .01 field values.

SQL tables can have primary keys and only a single row with a given value (or set of values) in a table. In contrast, in any attempt to identify FileMan rows without using the internal entry number, four elements come into consideration:

The .01 field of the file

The unique number or row id of the record (.001 field), called *internal entry number* (IEN) in FileMan terminology

*Identifiers*, according to FileMan terminology, declared for a file and used in lookup

Fields declared as *mandatory*, some of which may be identifiers

In fact, the true unique primary key of a FileMan file is the internal entry or record number (.001 field). Open M/SQL and KB_SQL, which are based on M, support accessible record numbers.

FileMan explicitly supports only a single attribute, the internal entry number, to serve as a key for a file. There is no restriction on storing composite structures in fields. By convention, some FileMan .01 columns contain values that are composite structures, such as names (Last,First). Use of numeric keys requires special handling (.001 field).

You can represent subfiles as tables, at least conceptually, using appropriate mapping. Project out all multiple-valued FileMan columns to become base tables. Subfiles thus become tables. Subfiles scope naming so naming conflicts can arise when files are decomposed into sets of tables. A subfile could have the same name as a file at the same site. Concatenation conventions are required for automated table and field naming in creating tables based on subfiles. Internal entry numbers for the file and each subfile level are required to identify records developed from subfiles.

Intersystems Corporation and KB Systems, Inc., have both implemented automatic global mapping facilities to support SQL interfaces to FileMan databases.

## Supporting Object Data Management

Implementors of object data management systems based on M treat persistent objects as arrays (syntactically as subscripted variables). The "general approach" of implementers to supporting the object programming paradigm object programming based on M has been "to use Globals as the storage area for object data."[26] Garcia and Supakkul[27] assert that in M "any global (in conjunction with the global module) is an example of an object with a high level of functionality (service) that can be demanded of an M system relative to that global."

You can design a subtree of an M array to represent state, properties, rules, and conditions, and to store components of algorithms or entire programs of executable code.[28] You can easily use such arrays to represent state transition diagrams. You can treat subtree pathways as relations for the purpose of testing non-procedurally for existence (as in Prolog). Thus you can store specification, current state, and operations for a given object according to an arbitrary indexing strategy for a classification or identification system. You can replicate structures of this type in any standard M implementation, because all "pointers" are symbolic.

Although you can store executable code in the arrays, designers of object system prototypes usually implement methods in M routines. In an M system, the routine is the unit of stored programming. You may store portions of code in arrays and then specify run-time selection and retrieval of specific portions of a program from an array. The M standard supports syntax for six varieties of indirection. Partial indirection (indirection in specifying global references) is particularly valuable to implementors and application designers in supporting object-oriented environments.

Three examples of implementations of object-oriented environments based on M are EsiObjects (Educational Systems, Inc.), the object approach of Omega Computer Systems, Inc., and MOOPS (MGlobal International, Inc.). EsiObjects and MOOPS use extensions of the standard M language in order to include object-oriented constructs. Some of these extensions are formally proposed extensions to the ANSI/MDC X11.1 standard and some are vendor specific. Omega uses extrinsic functions and standard M.

EsiObjects[29] uses the following object hierarchy: (1) universal, (2) class, (3) instance, and (4) temporary, of which the first three elements of this hierarchy are based on M globals. EsiObjects implements a preprocessor that verifies code, expands extensions, controls programming conventions, and enforces encapsulation. The preprocessor has optimizing capabilities that deal with such needs such as performance in single and multiple inheritance searching. Browsers allow exploration and modification of the structure of objects.

In looking forward to the next X11 standard, EsiObjects incorporates the use of standard structured system variable names (ssvns) which permit standard structure definition that is independent of vendor implementation.

Omega Computer Systems[30] uses M globals directly in its object data management approach. This implementation evolved from an effort to develop tools and achieve reusability and is used with a specific group of applications. In this implementation the creation of a metaclass enables you to write a generalized M routine to "be applicable to all classes which are instances of the metaclass." This approach ties class representation directly to global array structure, as Example 6 shows.

Example 6

```
^VENDOR(ID) = data fields
^GLCHART(ACCT) = data fields
```

Example 6 uses ^VENDOR for the Vendor Demographics Class and ^GLCHART for the General Ledger Chart of Accounts Class. Omega Computer Systems, which markets a law firm package, looks forward to being able to modify components of its package without extensive rewriting of major parts.

MGlobal presents object and shared variables in the MOOPS product[31] syntactically as M local variables but the system implements these kinds of variables as global variables of restricted classes. The complete hierarchy of variables in MOOPS is as follows: (1) bound, (2) method, (3) object, (4) shared, and (5) global. In MOOPS, ordinary M globals are accessible in object routines. A class is represented as an M routine incorporating all the methods associated with the class. Implemention-specific commands permit establishment of a hierarchy of classes and of variables shared by objects instantiated from a class.

The MUMPS Development Committee is consulting the experience of these and other implementations in the process of M standardization.

## Conclusion

This paper has described the general practice of global mapping in M computing environments, with a focus on two topics of current interest: (1) support for relational database systems, (2) support for heterogeneous environments involving FileMan and SQL, and (3) object data management. Global mapping constitutes a data definition strategy for a given database. Database systems use global mapping to relate complex logical structures required for applications to the relatively simple platform offered by M arrays.

## Trademarks

DASL, DSM, and VAX/VMS are trademarks of Digital Equipment Corp.
FOCUS for VAX/VMS is a trademark of Information Builders, Inc.
KB_SQL is a trademark of KB Systems, Inc.
Open M/SQL is a trademark of InterSystems Corp.
MUMPS is a trademark of Massachusetts General Hospital
Trustware is a trademark of National Computer Systems

## Notes

1 *Object Data Management Reference Model.* (ANSI Accredited Standards Committee. X3, Information Processing Systems.) Document Number OODB 89-01R8. 17 September 1991.

2 R. A. Greenes, A. N. Pappalardo, C. W. Marble, and G.O. Barnett, "A System for Clinical Data Management," *Proc. Fall Joint Computer Conference* (Montvale, NJ: AFIPS Press, 1969), p. 301.

3 J. Bowie and G. Octo Barnett, "MUMPS -- An Economical and Efficient Time-Sharing System for Information Management," *Computer Programs in Biomedicine* 6 (1976): 17.

4 Gordon C. Everest, Salvatore T. March, Magdy S. Hanna, and Mark N. Sastry, "An Analysis of the Data Representation Constructs of the ANSI NDL and SQL Standards," *Computer Standards and Interfaces* 10 (1990): 3-27.

5 Wolfgang Giere, *BAIK: Befunddokumentation und Arztbriefschreibung im Krankenhaus* (Taunusstein, Germany: Media, 1986).

6 Joop Duisterhout, Berend Franken, and Frans Witte, "AIDA: A Set of Software Development Tools for Building Information Systems," *MUG Quarterly* 15, 2 (1985): 29-32.

7 Tatsuhiro Uchida and Donald A. Smith, "Set GNOSIS = MUMPS + Prolog," *MUG Quarterly* 15, 3 (1985-86): 14-19.

8 Sally Webster, Mary Morgan, and G. Octo Barnett, "Medical Query Language: Improved Access to MUMPS Databases," *MUG Quarterly* 16, 4 (1987): 9-11. Sally Webster, Mary Morgan, and G. Octo Barnett, "Medical Query Language: Improved Access to MUMPS Databases," *Proceedings of the Eleventh Annual Symposium on Computer Applications in Medical Care*, Washington, DC (IEEE, 1987), pp. 306-309.

9 Darrell W. Simmerman, "The Data Base MUMPS System - DBMpS," *MUG Quarterly* 15, 4 (1986): 37-38.

10 Kyle A. Lindner, David J. Whitten, Linda Elting, and Gerald P. Bodey, "Ntelligence: A FileMan-based Expert System," *MUG Quarterly* 20, 1 (1990): 64-67.

11 Robert D. Andrews, "A Common Data Structure for Complex Clinical Data," *MUG Quarterly* 20, 1 (1990): 49-53.

12 Edward J. McIntosh, "A Model for Interfacing MUMPS and SQL," *MUG Quarterly* XX, 1 (1990): 14-20.

13 E. J. Yannakoudakis, *The Architectural Logic of Database Systems* (London: Springer, 1988), pp. 205-210.

14 Barry E. Jacobs, *Applied Database Logic I: Fundamental Database Issues* (Englewood Cliffs, NJ: Prentice-Hall, 1985), pp. 59-84.

15 Charles S. Volkstorf, *The MUMPS Handbook of Efficiency Techniques* (College Park, MD: MUG, 1985).

16 Richard F. Walters, "Database Optimization: A Overview," *MUMPS Computing* 22, 5 (November 1992): 52-59.

17 Dave Middleton, "Query Optimization in MUMPS," *MUG Quarterly* 21, 3 (June 1991): 32-39.

18 Michael R. Pantaleo, *Relational MUMPS: A practical approach for designing relational database systems* (Cambridge, MA: Intersystems, 1991).

19 *Open M/SQL: A Gentle Introduction* (Cambridge, MA: InterSystems, 1991).

20 *KB_SQL Database Administrator's Guide* (Herndon, VA: KB Systems, 1991).

21 William J. Harvey, "Implications of Non-1NF Extensions to the Relational Database Model for the MUMPS Standard and MUMPS Databases," *Proceedings of the 15th MUMPS Users' Group of Japan*, Supplement (Nagoya: MUG-Japan, 1988), pp. 2-22.

22 K. O'Kane, "Design for a Relational Database System in MUMPS," *MUG Quarterly* 15, 2 (1985): 33-37.

23 K. O'Kane, "A Portable Hybrid MUMPS Development System Host," *Proceedings of the Seventh International Computer Software and Applications Conference* (Silver Spring, MD: IEEE Computer Society, 1983).

24 *Focus for VAX/VMS$^{TM}$: Interface to Digital Standard MUMPS, Release 6.1* (New York, NY: Information Builders, Inc., 1991).

25 J. Milan and D. G. Major, "MUMPS and Data Modelling," *MUG Quarterly* 15, 4 (1986): 27, 29-31, 33-34.

26 ANSI/MDC X11/SC15/TG2/91-2, "Object Oriented MUMPS Program Enhancement." 26 December 1991.

27 Alfredo Garcia and Somboon Supakkul, "Object-oriented Programming in MUMPS," *Proceedings of the Fifteenth Annual Meeting of the MUMPS Users' Group* - Europe, pp. 21-27 (Rotterdam: MUG-E, 1990), pp. 21-27.

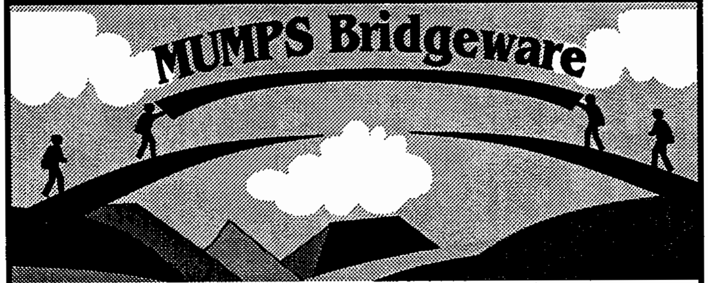
28 Bill Harvey and Bob Skovira, "Information Modeling Using Associative Global Arrays in MUMPS," *Object Oriented Reasoning in Information Modeling* (Proceedings of OOPSLA Workshop), Haim Kilov and Bill Harvey, eds. (Coraopolis and Pittsburgh, PA: Robert Morris College, 1992), 17-19.

29 Terry L. Wiechmann and Jerry Goodnough, "EsiObjects: An Object Oriented Application Development Environment" (Bolton, MA: Educational Systems, Inc., 1992).

30 Don Gall, "Object Oriented Programming, MUMPS and the Real World" (Phoenix, AZ: Omega Computer Systems, Inc., 1992).

31 Frank M. Brown, "MOOPS 0.2 - MUMPS Object Oriented Programming System, Programmer's Reference Manual" (Houston, TX: MGlobal International, Inc., 1991).

**The fastest way to migrate from MIIS to MSM or MUMPS/VM without disturbing the users**



MUMPS Bridgeware is the fastest way to convert the source code of your original MIIS programs with 100% accuracy.
MUMPS Bridgeware also transfers your Database at 20 Mbytes/h, decoding the original disk blocks into MSM or MUMPS/VM disk blocks.
Let MUMPS Bridgeware protect your software investment. We guarantee your satisfaction and minimum down-time.
The only thing your users will notice is increased system performance.

**CompScientia**

Micronetics VAR

Rua Barão do Flamengo, 32 - 6° andar
22220-080 - Rio de Janeiro - RJ - Brazil
Phone 55 21 205-4423 • Fax 55 21 285-7852

**More than 100.000 programs converted**