AN INTEGRATED ENVIRONMENT FOR THE NUMERICAL MODELING OF COMMUNICATION ANTENNAS BASED ON RELATIONAL DATABASES

Virginia Stover
Department of Mathematics and Computer Science
University of San Diego
San Diego, CA

James C. Logan
Research and Development Division
Naval Command, Control and Ocean Surveillance Center
San Diego, CA

Abstract

As modeling systems mature, they become larger, more complex, and more difficult to maintain. Modeling tools increase in number and complexity. Frequently they are written in different languages and require data in different formats. Databases also increase in size as modeling systems are applied to new and more complex problems. Engineers spend large amounts of money trying to integrate tools and data that are basically incompatible. Unfortunately, budgets do not grow at the same rate as the complexity of our modeling systems and databases. To maintain productivity, it is necessary to design modeling environments that can handle large amounts of data in flexible ways and are simple to maintain and upgrade.

This paper describes a new environment developed by the authors for the modeling of communication antennas based on a relational database management system. This approach simplifies the task of integrating a set of heterogeneous programs with incompatible data formats. The relational database provides a common store for all modeling objects including the antenna, platform, ground, electromagnetic sources, currents, charges, and fields, and model history. The database management system provides the organization, storage, and retrieval functions and some of the data input, validation and display functions for the antenna models. The main advantages of this approach are its ability to grow as new tools and capabilities are added, its portability to other machines and operating systems, and the ability it provides engineers to easily share data among themselves and with other modeling applications.

This work was conducted for the Naval Ocean Systems Center as part of the Navy Summer Faculty Research Program, a cooperative program with the American Association for Engineering Education (ASEE).

Introduction

Engineers use both physical and numerical models to predict the performance of communication antennas. Some of the numerical analysis programs in use are based on the Method of Moments and include NEC[1], MININEC[2], and Junction[3]. These programs are intended for modeling arbitrary geometries defined by wire frames and surface patches. They typically compute currents, charges, impedances, electric and magnetic fields, and other antenna parameters as output. Other numerical analysis programs are based on the Geometric Theory of Diffraction (GTD) and Finite Difference Time Domain (FDTD). GTD programs compute electric and magnetic fields from arbitrary geometries composed of generic shapes such as plates and cylinders. FDTD programs compute fields from partitioned volumes and surfaces.

Besides an analysis program, computer antenna modeling requires programs for inputting geometric, electromagnetic, and program control data and for analyzing and displaying results. Many of these support programs are sophisticated special-purpose tools. Others are off-the-shelf products like CAD programs, spreadsheets, and database management systems for inputting data, statistical analysis programs and graphics programs for analyzing output, and word processors and desk-top publishing systems for generating reports. These programs are often written in different languages and require input and output data to be in different formats.

Antenna models typically require large data sets for both input and output. Besides the antenna, all or part of the platform (ship, tank, jeep, airplane, etc.) may be required in the model. To model a ship may require thousands of nodes, wires, and patches. Many versions of the same model are often generated when investigating various antenna configurations. Output is often computed repeatedly over a range of frequencies for one or more potential antenna sites. Furthermore, Method of Moments codes have rigid requirements for model input. For example, there are restrictions on wire radius, wire segment length, wire spacing, number and angle of wires at junctions, size of surface patches, and the total number of unknowns (nodes, wires and patches). Therefore, users often have difficulty preparing and validating input data sets, especially when these sets are large. In addition, users may have difficulty converting data to the format required by the analysis software.

An Integrated Environment

Many of these difficulties can be overcome by providing a single integrated environment for antenna modeling. A current Navy effort in this area is called the Numerical Electromagnetic Engineering Design System or NEEDS[4]. NEEDS will combine existing software tools into a single uniform environment. It will guide the user through the steps necessary to build a model, validate the model, compute currents and other EM parameters, and analyze and

display results. It will convert data as needed, store intermediate and final results, and document the history of models and their various versions. It will facilitate the reuse and sharing of model input and output. It will allow communication over networks for remote processing. In addition, the system will be portable across a variety of machine types and operating systems.

It is important to design NEEDS for maximum flexibility. It is inevitable that additional analysis and support tools will be added. It must be possible to modify the system easily in order to integrate new tools written in arbitrary languages. This requires that the major components of the system (model input, electromagnetic analysis, output display, process control, and data management) be implemented in separate independent modules. It must be possible to use the data in ways not necessarily anticipated at development time. Thus, data retrieval methods must be flexible. To support a variety of analysis codes, the model geometries should be represented by generic shapes from which wire frame or surface patch models can be automatically generated as required by the particular analysis codes used.

There are many issues to be addressed in building an integrated system such as NEEDS. These issues include conceptual model representation, partitioning algorithms for generating wire segments and patches, model evaluation and validation, and user interface designs. This paper, however, focuses on the issue of data management. This is an issue that becomes more important as databases increase in size and complexity. A significant amount of time and money is now being spent maintaining large databases that are basically incompatible. Methods are needed for handling large databases in ways that will allow for the efficient sharing of data among engineers and across tools and applications.

Data Management

A major problem with integrating heterogeneous programs is the management of incompatible data sets. Each program in the system typically uses its own internal files and has its own unique file formats. Thus, there tends to be significant duplication of both data and data-accessing routines. This duplication is inefficient and can lead to serious inconsistencies in the data. In addition, conversion programs are needed to translate files from one format to another. One or more conversion programs may be needed between each pair of programs in the system. The number of conversion programs needed, therefore, generally increases as the **square** of the number of programs in the system. Finally, data files tend to be difficult to modify. Any change in the format of a file requires changes to all programs accessing that file.

One way to deal with these problems is to use a common database. By using a common database, each item of data needs to be

stored in only one place, and common data access routines can be provided. Furthermore, conversion programs are needed only between each program file and the database. Thus, the number of conversion programs needed will grow **linearly** with, rather than as the **square** of, the number of programs. This savings becomes more important as the system grows. Finally, as we shall see, database languages exist that make application codes less dependent on data file formats.

Relational Databases

The leading database technology today is the relational database[5]. Relational databases are known for their ability to minimize data redundancy, provide flexible data representation, and allow for efficient data access. A relational database organizes data as a collection of tables. A separate table is used to represent each type of object or "entity" in the database. Each table has a fixed set of columns representing the characteristics or "attributes" of each object type. And each row of the table represents a single object or "instance" of that type.

For example, each node in a wire frame can be described by its x-, y-, and z-coordinates (Table 1). An additional attribute, called Node_Number, is used as a key to identify each node uniquely.

NODES

Node_ Number	X	Y	Z
1	0.0	0.0	0.0
2	0.0	0.0	1.5
3	2.8	0.0	1.5
4	0.0	2.8	1.5

A List of Nodes
Table 1

Tables also can be used to represent relationships among two or more entities. Relationship tables combine the key attributes (such as names or ID numbers) from two or more entities. For example, wires can be characterized by the indices of the nodes at each end of the wire (Table 2). Additional attributes, such as Radius and Number_of_Segments, can be added to describe the relationship further.

WIRES

Wire_ Number	Node1	Node2	Radius	Number_Of_ Segments
1	1	2	0.001	4
2	2	3	0.01	2
3	4	2	0.002	4

A List of Wires
Table 2

Note that the columns labeled "Node1" and "Node2" in the Wires Table contain values of the key attribute (Node_Number) appearing in the Nodes Table. These attributes are called "foreign keys" in the Wires Table. A foreign key points to a key in another table.

It is important to design database tables carefully in order to reduce the number of blank spaces and the amount of redundant information. Redundant information wastes space and can lead to inconsistencies. A well-known process for designing relational database tables is called "normalization". Normalization breaks large tables into smaller tables so that each table describes a single atomic entity or relationship. Relationships among tables are preserved in the foreign keys. Smaller tables can later be combined into larger tables, as needed, by using an operation called a "join".

Both the Nodes Table and the Wires Table above are normalized. Notice that when multiple wires end at the same node, there is no need to repeat the x-, y-, and z-coordinates of that node. The coordinates are stored only once in the Nodes Table and a reference made to them in the Wires Table through the foreign keys. This helps to maintain the consistency of the data if changes are made to the node's coordinates.

Besides the database itself, a query language is needed for organizing, storing, and retrieving values from a database. The Structured Query Language (SQL) is an ANSI standard for relational databases. SQL commands can be executed interactively or embedded within a general-purpose programming language like C or FORTRAN. SQL queries allow the user to retrieve any subset of the rows of any table from any subset of its columns, as well as to combine tables. SQL provides flexibility in allowing the user to retrieve precisely the subset of data required. SQL is a non-procedural language. That is, it describes **what** subset of data to retrieve without describing **how** to retrieve it. This minimizes the amount of programming required. It also ensures that the code that accesses the data is independent of how the data is actually stored. This "physical data independence" ensures that changes to the physical structure of the database can be made without affecting the code

that accesses it. Some examples of SQL queries are as follows:

EXAMPLE 1. The query

```
SELECT SUM (Number_of_Segments)
FROM Nodes, Wires
WHERE (Node1 = Node_Number) AND (Z > 0.0);
```

retrieves the total number of segments on those wires for which the first endpoint has a positive z-coordinate. Notice that this query requires information from both the Nodes Table (for the z-coordinate) and the Wires Table (for the number of segments).

EXAMPLE 2. The SQL query

```
SELECT W1.Wire_Number, W2.Wire_Number
FROM Wires W1, Wires W2
WHERE ((W1.Node1 = W2.Node1)
   OR  (W1.Node1 = W2.Node2)
   OR  (W1.Node2 = W2.Node1)
   OR  (W1.Node2 = W2.Node2))
  AND  (W1.Wire_Number < W2.Wire_Number);
```

finds each pair of adjacent wires (temporarily called "W1" and "W2") and retrieves their wire numbers. It combines a copy of the Wires Table with itself. The result is shown in Table 3 below.

Wire_Number	Wire_Number
1	2
1	3
2	3

Query Result
Table 3

A complete database management system will provide capabilities other than simple storage and retrieval functions. These capabilities usually include multi-level security, backup and recovery in case of software or hardware failures, concurrency control to allow more than one user to access the database at the same time, and indexing and clustering to speed up access to the most frequently used data. Commercial database management systems usually provide tools for creating user interfaces that facilitate access to the database. And they allow database files to be imported from and exported to other operating system and database files.

The ORACLE Relational Database Management System

The authors built a prototype of an integrated antenna modeling environment based on a relational database management system (RDBMS). The prototype demonstrates the feasibility of using a RDBMS for providing the following data management capabilities:

- a central uniform data repository
- efficient access to the data, either directly by the user or transparently through application programs
- low-level data validation
- conversion of file formats between those used by the database management system and those used by the application programs
- documentation of the history of models and their various versions and
- archiving of models for long-term storage.

Low-level data validation involves checking constraints on the data imposed by the conceptual model. Additional constraints on the data imposed by specific analysis programs are assumed to be handled by the analysis software itself. Security, concurrency, and network access were not considered to be important at this time.

We decided to use a commercial RDBMS for several reasons. First, commercial systems are extremely reliable. Second, the user interface tools provided by most commercial systems minimize the time needed to develop and update application software. And third, it would take many years to develop a system that would provide the same functionality as that currently provided by commercial systems.

There are many commercial RDBMS's on the market. ORACLE[1] was chosen for this project to achieve some standardization with other Army and Navy projects. ORACLE is a complete database management system providing all the capabilities mentioned above. It runs on a variety of machines under all major operating systems allowing application software to be easily ported. Versions of ORACLE are available that run over a network. ORACLE supports embedded SQL commands in both C and FORTRAN.

Description of the Prototype

The prototype runs under MS-DOS[2] 3.3 and ORACLE RDBMS 5.1B on

---

[1]ORACLE is a registered trademark of Oracle Corporation.

[2]MS-DOS is a registered trademark of Microsoft, Inc.

an IBM-PC[3] or compatible with at least one hard disk drive and at least 256K of extended memory. The extra memory is required for the ORACLE RDBMS. The PC version of ORACLE is a single-user system.

The central data repository is a relational database with predefined tables. Users can directly access the database through SQL commands or indirectly through application programs and database forms. The database contains model input and output, model histories, control data, and some intermediate model-derived data.

In addition to the database, the prototype includes the following programs:

- the Junction code from the University of Houston for computing currents, charges, far fields, and near fields from models described by a combination of wires segments and triangular surface patches
- an ORACLE menu for navigating through the system
- ORACLE forms for inputting data and displaying results
- several C programs with embedded database queries for converting between the ORACLE database and Junction's ASCII formatted files and
- a supervisory program written in C.

Included in the Junction code are routines for producing surface patches from generic shapes such as cones, cylinders, and spheres.

During a typical modeling session, a user would

- define a new model or select an existing model for update
- input or update the model geometry and electromagnetic data including definitions of wires, surfaces, sources, frequencies, and far and near field locations, as desired.
- select the desired output such as currents, charges, far fields, and/or near fields
- request the system to compute the desired output
- view the output

The supervisory program ensures that programs are executed in the correct sequence and that each program receives its data in the proper format. The computation of output can be divided into several phases: 1) creating wire segments and surface patches, 2) computing additional geometry parameters such as the midpoint of wire segments and locations of body-wire junctions, 3) computing currents, and 4) computing charges, far fields and near fields, if desired. Each of these phases can be executed separately and intermediate results viewed. These phases must be executed in this

[3]IBM is a registered trademark of International Business Machines Corporation.

order, but not all changes to the input require recomputation of each phase. For example, a change in a source location requires that the currents, and hence charges and fields, be recomputed, but a change in a near field location requires only that near fields be recomputed. To control the execution sequence, tables in the database record which types of data have been defined by the user and which types of data have been computed by the system. By referring to these tables, the supervisory program prevents users from attempting to compute output data before all prerequisite input data have been entered, and it avoids computing data that are already available in the database.

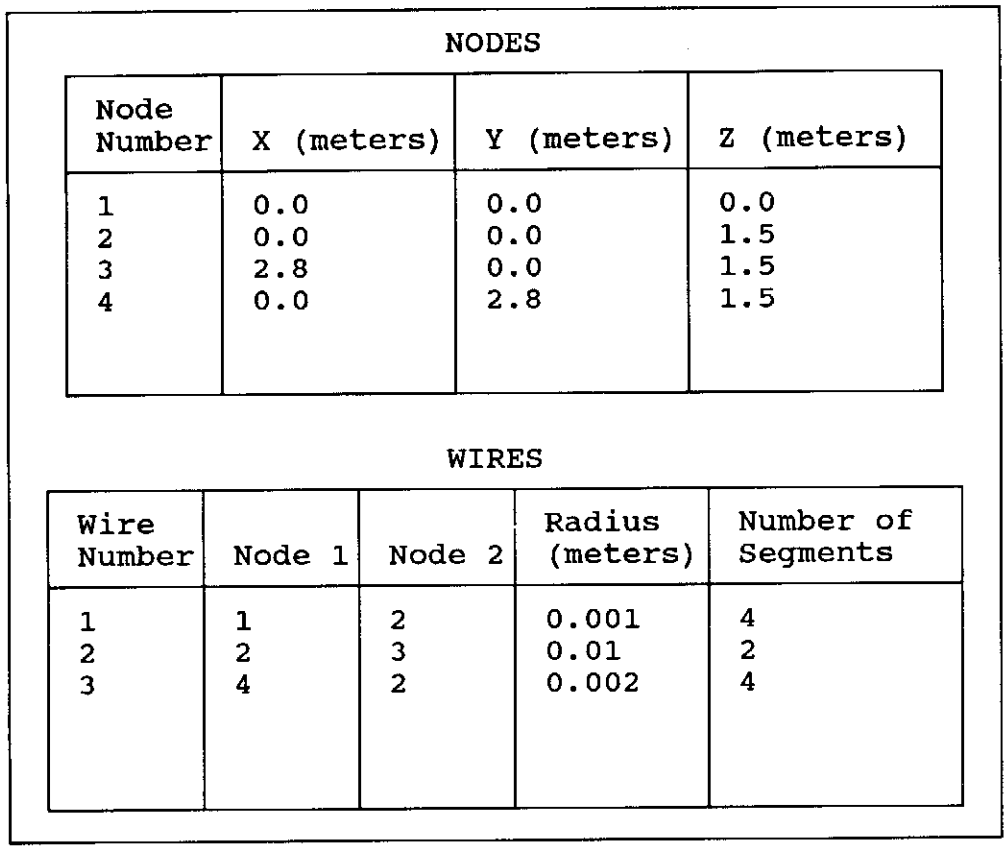
The user traverses the system by means of a menu. Menu and sub-menu commands ultimately execute database commands, operating system commands, database forms, or other application software. A background menu, accessible from anywhere in the menu system, allows advanced users to execute their own operating system or database commands directly in order to accomplish specialized tasks.

A form-based user interface facilitates input and output of non-geometric data and geometric data for small models. Forms allow for the entry of data independent of the analysis software. Forms can do low-level data validation, such as type-checking and range-checking, before data is committed to the database. This helps to maintain the integrity of the database, and it provides immediate feedback to the user when mistakes are made. Help messages and default values can be provided for each field in the form. The developer can designate certain fields as key fields which must have unique values. Non-key fields can be designated as either mandatory or optional. (Optional fields may contain null values.) And user updates can be restricted to a subset of the available fields, if desired.

Since these forms are closely integrated with the database, they can perform functions not common to other forms software. For example, forms can be designed to make simultaneous updates to other fields, such as foreign keys, in other database tables (see Example 2 below). This helps to maintain the consistency of the database. The same form that is used to insert or update data in the database can be used to retrieve data from the database. The user performs queries by providing a value or a range of values in one or more fields of the form. The form will then retrieve all database records that include those field values. To allow additional functionality, database operations and/or C code can be tagged to certain events (such as updates, queries, or cursor moves) that are executed when the user causes those events to occur. These "triggers" are used for more advanced data validation and housekeeping functions.

The following are some examples of the capabilities of database forms. The form in Figure 1 references the Nodes and the

Wires Tables discussed above.



NODES

Node Number	X (meters)	Y (meters)	Z (meters)
1	0.0	0.0	0.0
2	0.0	0.0	1.5
3	2.8	0.0	1.5
4	0.0	2.8	1.5

WIRES

Wire Number	Node 1	Node 2	Radius (meters)	Number of Segments
1	1	2	0.001	4
2	2	3	0.01	2
3	4	2	0.002	4

A Form for Inputting and Displaying Wires
Figure 1

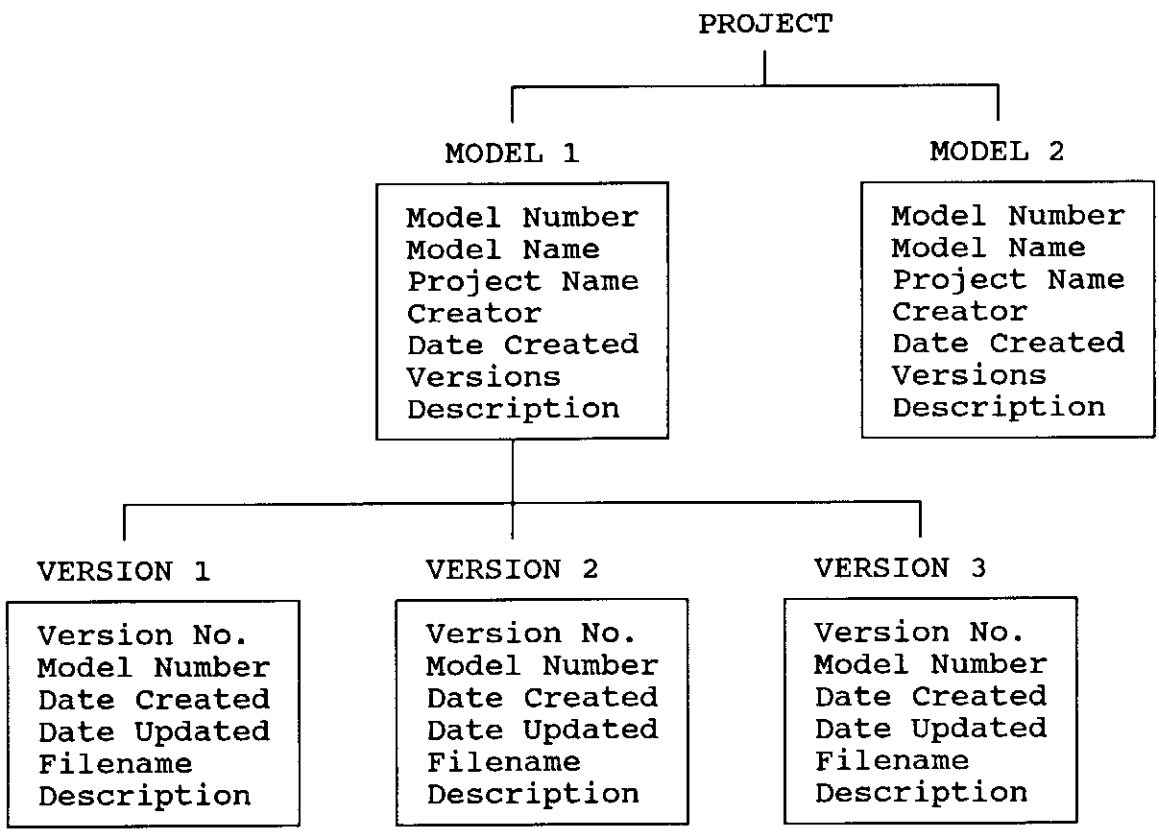
EXAMPLE 1. To insert a new wire, the user types a value for each field in the Wires Table and invokes the Insert function. The form can verify that the two nodes referenced by the new wire have already been defined in the Nodes Table.

EXAMPLE 2. If the user wishes to change the index of Node 4 to Node 5, the user changes the "4" to a "5" under Node Number in the Nodes Table and invokes the Update function. The form can propagate this change automatically to the Wires Tables, so that Wire 3 would then connect Node 5 to Node 2.

EXAMPLE 3. To list all the Nodes in the database with an x-coordinate greater than 0.5 meters, the user types ">0.5" in the X field of the Nodes Table and invokes the Query function.

The prototype also records the history of models as they are developed by the user. It allows the user to document different

projects, several models belonging to each project, and multiple versions of each model (see Figure 2). Each model is described by a model number, model name, project name, the name of the person creating the model, the date it was created, the number of versions it has, and a textual description of the model. Each version is described by a version number, the number of the model to which it belongs, the date it was created, the date it was last updated, the name of the file where it is stored, and a textual description. A new version is created automatically after the user computes the currents for the existing version. Thus, each time the currents are computed for a group of frequencies, the input data become fixed for that version. Any additional changes to the input data are reflected in the new version.



The Description of Model Histories
Figure 2

Because of the limited memory of a PC, it would be difficult to store input and output data for all versions in the database simultaneously. Therefore, the database contains workspace for the current version only, plus a catalog of past and current models and versions. In addition, alternative sources, near field regions, and far field regions can be stored in the database. Previous models and their various versions are archived on disk and can be reloaded from the on-line catalog as needed.

Lessons Learned

A relational database management system proved to be well-suited to this application. It was possible to develop a fairly complex, modular system in a short time due to the database menu- and form-building software and the use of embedded SQL commands in application programs. These application-building tools also allow the developer to provide the user with transparent access to database. Thus, the typical user does not need to know that a relational database exists or how it is organized.

ORACLE, however, is primary a business-oriented database management system. Therefore, its query languages lack a complex data type and such built-in functions as square roots, exponentials, logarithms, and trigonometric functions needed for scientific and engineering applications. Thus, embedded C code is needed to perform these functions. A scientific database management system would provide these as extensions of the SQL language.

Another disadvantage of the ORACLE RDBMS for some is its size and cost. The database management system requires at least 1 MB of main memory and about 9 MB of disk space in addition to disk space for the database itself. Large amounts of memory and disk space, however, are becoming much more affordable. And ORACLE's portability and network capabilities may ultimately far outweigh these disadvantages.

The most significant advantage of the relational database is the flexibility that it provides. It allows data to be combined and retrieved in many ways. As the modeling capabilities of the system are extended in the future (to include, for example, loads, transmission lines, or material types), it will be necessary to add new tables or new columns of existing tables to the database. This database extension, however, does not require changing either existing data or existing code. As long as table and column names are not changed, existing analysis programs and support tools will still execute properly on the modified database.

It was difficult to keep more than one version of a large model in the database at one time due to the memory limitations of a PC. Therefore, the ability to make arbitrary comparisons of data across different versions or models is not a built-in capability of the system at this time. For advanced users, however, it is possible to make such comparisons by accessing the database directly. This requires knowledge of simple SQL commands in order to create new tables and to move data from one table to another.

Any environment that generates so much data, however, should provide for an efficient way to browse through the data, compare data from different models and versions, and do other kinds of data synthesis. In the future it would be advisable to run the database

management system on a dedicated workstation or minicomputer as a database server, so that numerous models and their various versions could be kept in the database at one time. A more sophisticated archival system is needed for expanding the work area as needed and for automatically storing data on disk or tape as available memory is used up.

Future Possibilities

It is anticipated that additional tools will be added to the integrated system in the future. These include graphical input programs for large geometries, 2- and 3-dimensional graphical output display programs, a windows-based user interface, and additional EM analysis tools, such as other Method of Moments (MOM) codes, Geometric Theory of Diffraction (GTD) codes, and Finite Difference Time Domain (FDTD) codes. Since this prototype was developed, ORACLE Corporation has marketed an interface for Microsoft Windows[4] 3.0 that will also allow developers and users to access ORACLE data from most Windows applications.

Different analysis codes (MOM, GTD, and FDTD) require slightly different input in different formats to model the same physical objects. In the past, as new antenna analysis programs were developed, new special-purpose tools were written to support model input. Thus, there is a need for tools that input and store generic objects that can be adapted to any analysis tool. This would allow input objects to be used with many different codes in many different applications. These codes could be modified to access the database directly thereby eliminating the need for file format conversion programs. Substantially the same tools are also needed to display such output as currents, charges, far fields, and near fields regardless of the program that generates this output. This need for reusable tools will become more important as the tools become more sophisticated and as integrated systems become larger. A common database format for describing geometries would facilitate the realization of this goal.

Finally, advantages can be foreseen for using standard databases and standard query languages in order to interface with other modeling systems. For example, data generated numerically on the computer could be more easily compared with data measured from physical antenna models if a common data format were used. This also would facilitate the integration of software for antenna modeling with software such as COEDS[4] for modeling entire communication systems. The use of a relational database management system that is widely portable can be an important step toward creating sharable data among such related applications.

---

[4]Microsoft Windows is a trademark of Microsoft Corporation.

References

1. Burke, G. J. and A. J. Poggio, "NEC - Method of Moments ", Naval Ocean Systems Center Technical Document 116, 1981, revised 1988.

2. Rockway, J. W., J. C. Logan, D. W. S. Tam, and S. T. Li, ***The MININEC System: Microcomputer Analysis of Wire Antennas***, Artech House, Inc., 1988.

3. Wilton, D. R. and S. U. Hwu, "Junction Code User's Manual", Naval Ocean Systems Center Technical Document 1324, August 1988.

4. Li, S. T., J. C. Logan, and J. W. Rockway, "Ship EM Design Technology," ***Naval Engineers Journal***, May 1988, pp. 154 - 165.

5. Korth, H. and A. Silberschatz, ***Database System Concepts***, McGraw-Hill, 1986.