DMG - Object-Oriented Iterative Knowledge Acquisition for the Generation of Diagnostic Hypertext Systems

Pasi T. Tyrväinen
tyrvaine@rc.nokia.fi
Department of Knowledge Technology
Nokia Research Center
P.O. Box 156, SF-02101 Espoo, Finland

Abstract

DMG is a development environment for building diagnostics applications of electrical equipment. Knowledge contained in an equipment model is compiled into a decision-tree like diagnostic logic, which is then used visually to pinpoint deficiencies in the performance of the equipment. The delivery application is a hypertext document which combines the diagnostic logic with links to sources of additional information. Benefits and experiences in real industrial use of DMG are presented; and practical issues on using a CLOS-based graphical user interface library are discussed.

1 DMG Overview

DMG [5, 8, 9] is a development environment for diagnostic applications of electric equipment. The diagnostics is based on structural models of the equipment around which all knowledge is organized. In addition to the structural models, knowledge about pathways of causal interaction about available tests is needed. Faulty parts are located by using the structural isolation method [2, 6]. The results of applying the structural isolation method is transferred into an decision-tree-like object structure called a diagnostic network. These diagnostic networks represent the logic of a diagnostic task in a compiled knowledge form. They are executed in a PC delivery environment called HyperExpert [3].

The main advantage of DMG is the fast generation of easily modifiable diagnostic advisory systems that can be verified and validated graphically. The applications generated using DMG are most valuable:

- when the production of a new product is starting, even if no heuristic knowledge is available,
- in the customer services, and also
- when considering the testability issues of a new product in the design phase.

1.1 Parts of the Development Environment

The system architecture of DMG is described in Figure 1. The DMG development environment runs in an engineering workstation and it consists of three parts:

1. the *model editor* for modelling the equipment,
2. the *generator* for generating diagnostic networks from the model, and
3. the *graph editor* for displaying and editing generated diagnostic networks.

Both the equipment model and diagnostic network are represented as structures of interconnected object instances, and both are displayed for the user using instances of window classes.

1.2 Model Editor for Structural Equipment Models

The model editor is used to create and modify hierarchical equipment models. These models contain objects such as modules, ports, connections, and tests. Each of these interconnected objects has a graphical representation in a model editor window.

As a model is represented in one window, an internal model of one module may be represented in another window. Thus, the user may access different levels of a hierarchical model at the same time, and the diagnosed system may be developed in a hierarchical manner.

Different kinds of faults can have dissimilar pathways of causal interaction and separate sets of tests

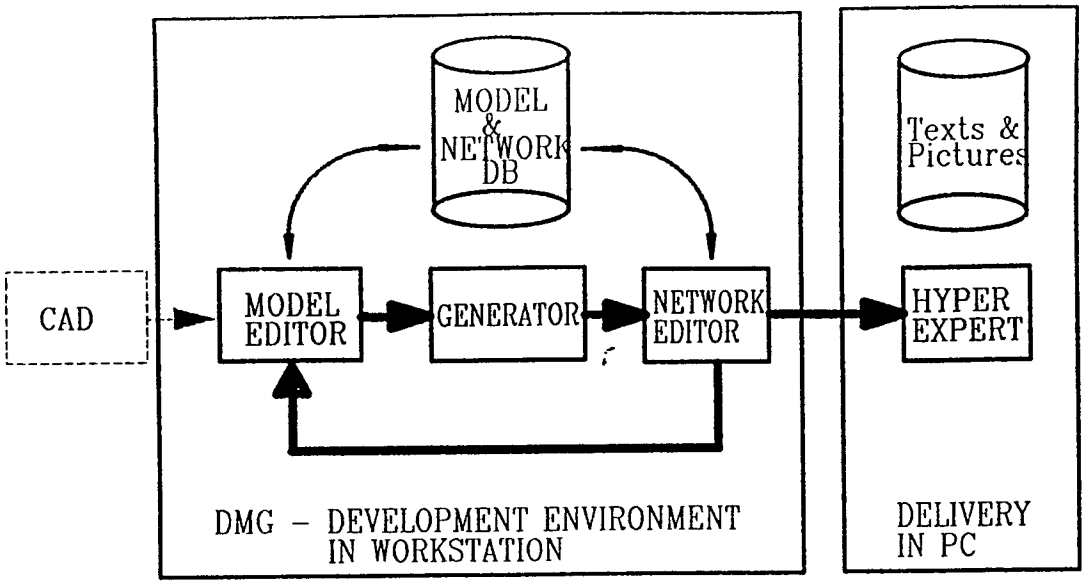

Figure 1: Architecture of DMG.

relevant to probe the propagation of faults. In DMG, different views of the same physical structure are defined as subsets of all connections and tests in the model. These views can be used to describe e.g., different dimensions of a signal (amplitude and frequency) or different functions of a piece of equipment (booting a PC or starting a program).

Direct manipulation of properties of the displayed objects is used to modify fault probabilities of modules, test descriptions, costs of tests, etc. In this manner, the information of model objects is easily accessible. This improves model consistency based on visual validation and reduces the effort of model building and maintenance.

1.3 Generator

The generator is a recursive LISP procedure that creates a compiled knowledge representation from the knowledge in the model. It optimizes the order of the performed tests minimizing the average time required for isolating a fault in the system.

The result of a generation process is the diagnostic logic represented as a decision-tree like structure of interconnected test nodes. Each of the nodes can also be expanded to a set of nodes that represents the links to the relevant textual and pictorial material in the delivery environment.

1.4 Graph Editor for Diagnostic Networks

The DMG Graph editor windows are used to display and manipulate generated diagnostic networks. The graphical representation gives a fast overview of the size and quality of the diagnostic application. Distinct trouble shooting sessions are represented as paths from the root of the tree to separate solution leaves via a set of test nodes. This kind of visual validation of results is efficient, and inconsistencies and shortages of a system model are easily pinpointed in the structure of the generated network. The verification and validation results give feedback to the model builder in an early phase and it can be used to verify the testability of a newly designed equipment. In some cases, minor changes to the model may cause major changes to the generated network.

Figure 2 illustrates the iterative knowledge acquisition process in DMG development environment: modify model, generate diagnostic network, validate network visually, modify model ... etc ... until the application is ready to be delivered in the PC environment. The knowledge acquisition by iterative knowledge compilation relies on the visual validation of the diagnostic networks and direct manipulation of the model objects. For these purposes, object-oriented user interfaces are needed.

1.5 HyperExpert - Delivery Environment

HyperExpert [3] is an expert system tool for advisory applications. It provides a MS-Windows-based graphical interface and combines expert system technology with hypertext features. Knowledge is represented by a network formalism that contains several node types. Nodes contain text, pictures and external

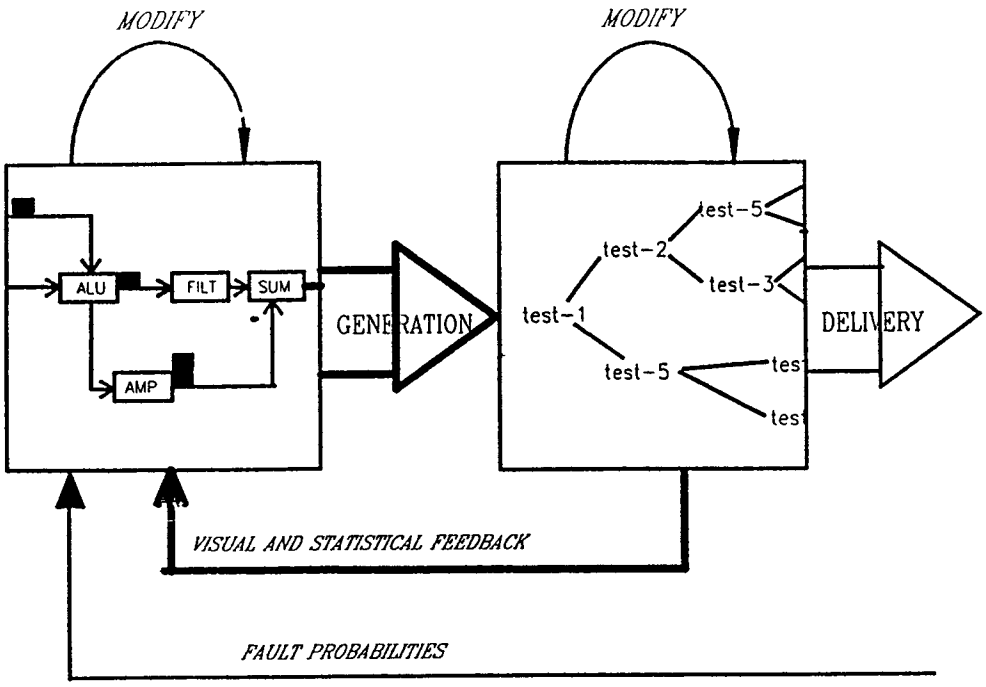

Figure 2: The iterative knowledge acquisition process in DMG.

function calls and are connected with branches and loops. Since the networks are stored in a database structure, no practical upper limits exists on the application size.

DMG uses HyperExpert as the delivery environment for generated diagnostic applications (use of some other hypertext tool is also possible). The final generation in DMG incorporates computational links to the additional information available in the delivery environment. This final version is then transferred to a PC and read into the HyperExpert database. The extra information is added as a set of files which can be updated without editing the diagnostic network itself.

When a new release of the diagnostic network is delivered, possible changes made to the old network are lost, but changes to the additional files are still available. This feature can be used to tailor the applications for different environments. The feedback to the DMG models can include fault statistics collected by an additional program in the delivery environment.

1.6 The Shower Example

The model "Shower" (Figure 3) in the DMG-Model-Editor window contains five modules connected to each other by connections that describe the pathways of causal interaction such as the flow of water from "Cold-water-system" via "Cold-water-pipes" (or via "Warm-water-system" and "Warm-water-pipes") to "Mixer" and out from the shower. A fault in any of these five modules can cause a fault in the flow of the water.

In addition to the "Flow" view of the Shower model, there is a view "Temperature" which is highlighted in the picture. The modules "Warm-water-system" "Warm-water-pipes" and "Mixer" and connections between them are included in the view as these modules can cause a fault in the temperature of the outcoming water.

Tests are displayed as smaller boxes and are used to measure the outputs of modules; Test "Water-out" includes the question "How is the outcoming water?" and answers "No water", "Only cold water", or "Flow and Temperature Ok" directing the fault isolation process to the relevant view.

The DMG-Graph-Editor window in the bottom displays a diagnostic network generated from the Shower model. If the trouble-shooter responds "Only Cold Water" to the test "Water-out", the test "Hot-tap" is performed next. The action in "Hot-tap" is "Try some other tap for hot water. Is it Ok?" and the possible results are "Ok", "Fail", or "Unable to perform the test" resulting in a Mixer-fault and the initiation of two uniqueinstances of test "Temperature" respectively. After the "Fail" answer to the test "Hot-tap", the test "=1=Temperature" can either attribute the fault to "Warm-water-system" or "Warm-water-pipes", or can report an ambiguity set of cardinality 2.

If the fault is attributed to a module that has an in-

ternal model, the fault can be analyzed further according to the internal model (The "Warm-water-system" model might have modules "Water-container" "Electricity" "Thermostat" etc.)

2 CLOS and User Interfaces - Experiments and Practical Issues

2.1 DMG in Use

The DMG system is now running in a workstation environment, and has been used by Nokia's business units during 1990. It was employed in several locations within the corporation - including diagnosing radio links, microcomputers, mobile telephones, and modems.

The experiments gained from the field tests show that the direct manipulation facilities of object-oriented user interfaces are important for the visual knowledge verification and validation, which enables the iterative knowledge acquisition process and, thus, fast knowledge acquisition in DMG.

According to our experiences, the use of DMG has reduced the time needed to design the diagnostic logic by 70%. In our case, about half of the time to produce a diagnostic application is used to the production of diagnostic logic and an other half to the production of supporting pictorial and textual material. Thus the total time savings is less dramatic; about 35% instead of 70%. In the near future we expect to cut down the effort needed to produce the support material by better utilization of existing material, aided by hypertext features of the delivery environment.

The application builders (including some casual users) that are using DMG are fairly satisfied with the system. They spend most of their time thinking, and the speed of the user interaction and generation is fast enough for their purposes. The functionalities of DMG can be adapted to their wishes more easily than in systems programmed in traditional programming languages.

2.2 Object-Oriented User Interfaces

DMG was first prototyped on a Symbolics LISP machine in 1987, and the implementation relied heavily on the object-oriented features and graphical tools of the Genera environment. After the prototype phase we had to find a way to port DMG on top of LISP in engineering workstations. For the similar needs of several projects at Nokia Research Center, we had to chose an object system and a user interface tool library. For the object system we chose Common Lisp Object System (CLOS [4]) and for the user interface tools, a library called AIGT was build.

AIGT (Application Interface Generation Tools) [1, 8] is a modular library of predefined window class definitions which is designed to be used in the building of user interfaces for Common Lisp applications. AIGT consists of a library of predefined window classes organized in a class hierarchy. An application uses AIGT by instantiating these classes with application specific instance variable values and by inheriting predefined classes to application specific classes. Only the class definitions needed are loaded to the application environment. In addition to basic interaction windows (textual I/O, command panels, form windows etc.), the library includes large application frameworks for specific application areas.

The implementation of AIGT was planned to be based on two standards: Common Lisp Object System as the object system, and X Window System (developed by M.I.T.) as the graphics system. We hoped that these components would have provided a common, portable and fairly stable foundation for AIGT.

The prolonged standardization process of CLOS made it necessary for us to adopt an object solution of our own, called NOS (Nokia Object System). NOS has been designed using Common Lisp and implements features of object systems needed for AIGT such as multiple inheritance. The development of NOS was guided by early CLOS standard proposals in order to minimize the conversion work necessary as CLOS becomes available. This goal was reached by delimiting the available functional features in NOS to those which were likely to exist in CLOS.

The implementations of X Windows System and interfaces to LISP came too late for our needs; therefore, it was replaced with the Window Tool Kit developed by Lucid Inc. This software was used because it provides a uniform window interface independent of the hardware used and was available for many platforms used by our customers (Sun, Apollo, HP, and MicroVax). Thus the first implementations of AIGT and DMG were based on NOS and Lucid's Window Tool Kit.

2.3 Porting DMG to Engineering Workstations

In 1988, DMG was ported to workstations on top of Lucid Common LISP and AIGT user interface tools. Relevant general purpose window classes for DMG

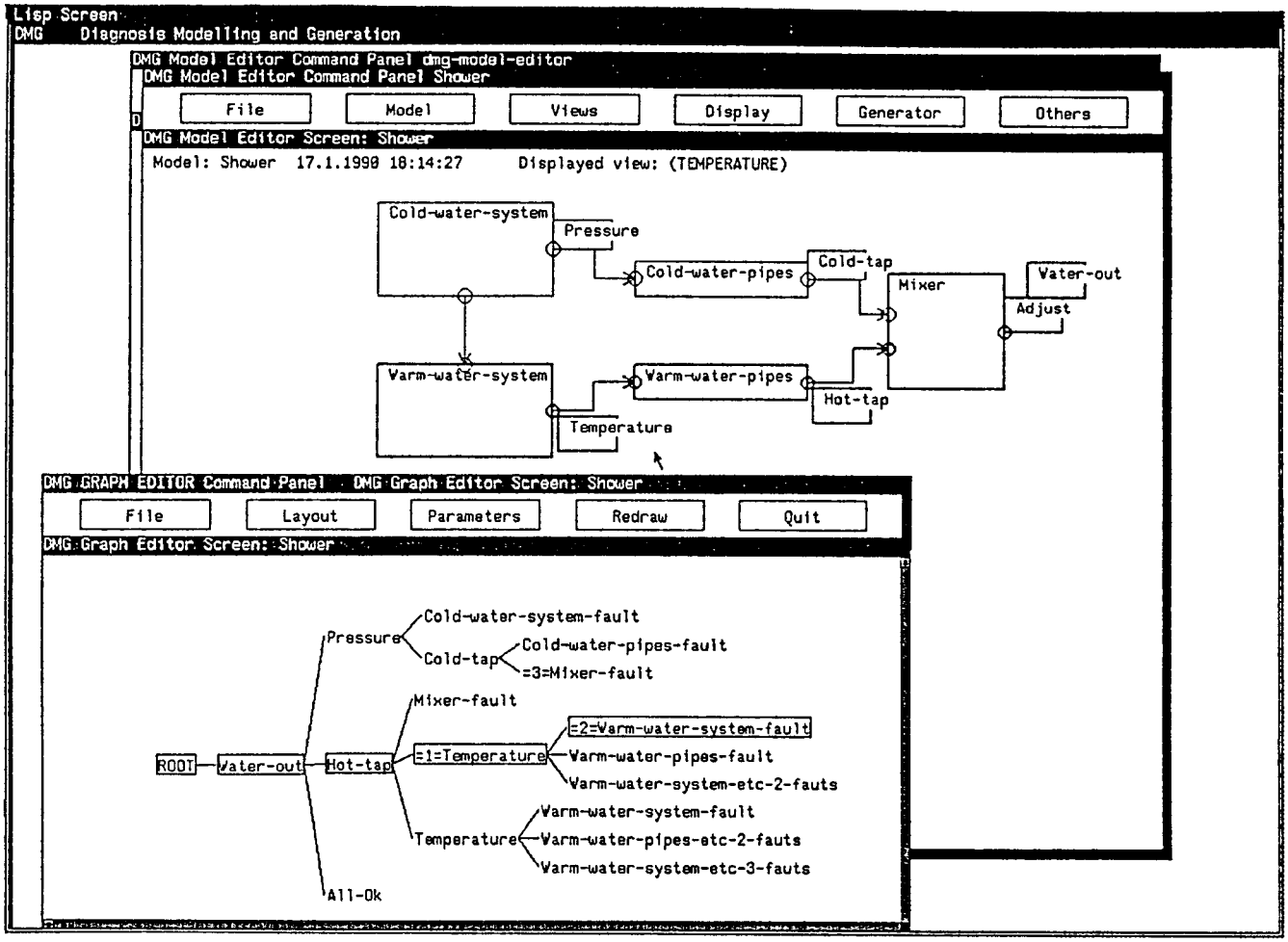

Figure 3: DMG development environment. A model "Shower" is displayed in a model editor window and a diagnostic network generated from the model is displayed in an overlapping graph editor window.

were found or added to the AIGT library, facilitating the reuse of lots of previous work. The effort required for porting DMG to AIGT was about 10% of the original programming effort, excluding the additions to the AIGT library. The facilities of the Genera environment and some parts of the DMG code were replaced with the methods of AIGT classes. In addition, we were able to leave out irrelevant parts of the Genera environment, reducing the total size of the runtime application. Speed of the runtime application depended more on the size of central memory available than on the implementation environment.

About 40% of the source code in current DMG system is DMG specific code and about 60% is reused AIGT code. The size of the DMG environment is kept small by loading only needed class definitions from the AIGT library, although a minor amount of unutilized AIGT code is included in loaded class definitions. (The use of task-specific shells enabled some demonstration applications to reuse the programming effort of more than 10 man-months with the effort of one man-month. In general the amount of reused AIGT programming code has been about 60-90% of the source code of an application. This figure tends to include about 10-30% overhead of unutilized AIGT code.)

Figure 4 illustrates the use of AIGT window classes in DMG. The DMG-Model-Editor inherits the class Block-Editor-Window and the DMG-Graph-Editor inherits the Graph-Editor-Window [7] respectively. The access and graphical representation methods of objects are based on inherited methods of corresponding Block-Editor-Windows and Graph-Editor-Windows object classes, Blocks, Connection-Ports, etc. The other window types for user interaction were heavily used in implementing DMG.

Some problems were encountered with the application framework types of windows which were not general enough to be tailored for the needs of DMG. In general, implicit assumptions and predefined paradigms of these task specific tools enable fast development of applications. On the other hand maintenance of generality requires either specialized methods to be modular and replaceable or use of intermediate class definitions and specialization via inheritance.

The library of predefined window classes is expanding all the time. Newly added classes are usually rewritten to improve generality after being used in more than one application. Very often it is also useful to make some changes to DMG instead of merely writing interface methods between new class definitions and DMG.

Redefining methods for library classes sometimes causes the typical software version management problems as the inherited AIGT methods change without the DMG builder being aware of it; therefore, good software engineering practice is needed. In any case, even if the inheritance of predefined classes causes some software management problems, it actually reduces them. For example, adding methods to print a window with a PostScript laser-printer for AIGT class Block-Editor-Window allows use of this functionality in DMG-Model-Editor with minimal extra effort.

3 Future Directions

3.1 Tools in the Near Future

The functionalities of NOS have been ported on top of CLOS using macros, and the newly added classes in the AIGT library can use all of the facilities provided by CLOS as needed. Thus DMG can run either on top of CLOS, PCL or directly on top of Common Lisp and NOS.

The window system used is still the Window Tool Kit of Lucid Common Lisp. Earlier we also considered porting AIGT on top of X Window System for greater speed and portability. So far, the speed has been only a minor problem when the proper hardware is used and we are able to run the applications on the platforms our customers are using.

A preferable possibility is to port the more advanced parts of AIGT on top of CLIM (by International Lisp Associates) or X Window System based CLUE (such as LispWorks by Harlequin and Delphi Common Lisp by Delphi S.p.A.). The primitive class definitions of AIGT could be replaced with CLIM classes which the more advanced AIGT classes would inherit. This is a feasible solution due to the modular structure of AIGT library. In the future either of these new standards can be the foundation upon which the more advanced tools of AIGT would be built, enabling use of DMG on top of a standard environment.

A standard for object-oriented databases is also needed for the LISP environment. So far, the lack of good tools for saving object structures in a permanent storage has forced DMG and other LISP applications to employ their own saving mechanisms. We are looking forward to commercial object-oriented databases that would help us in the future by providing a common solution for all applications.

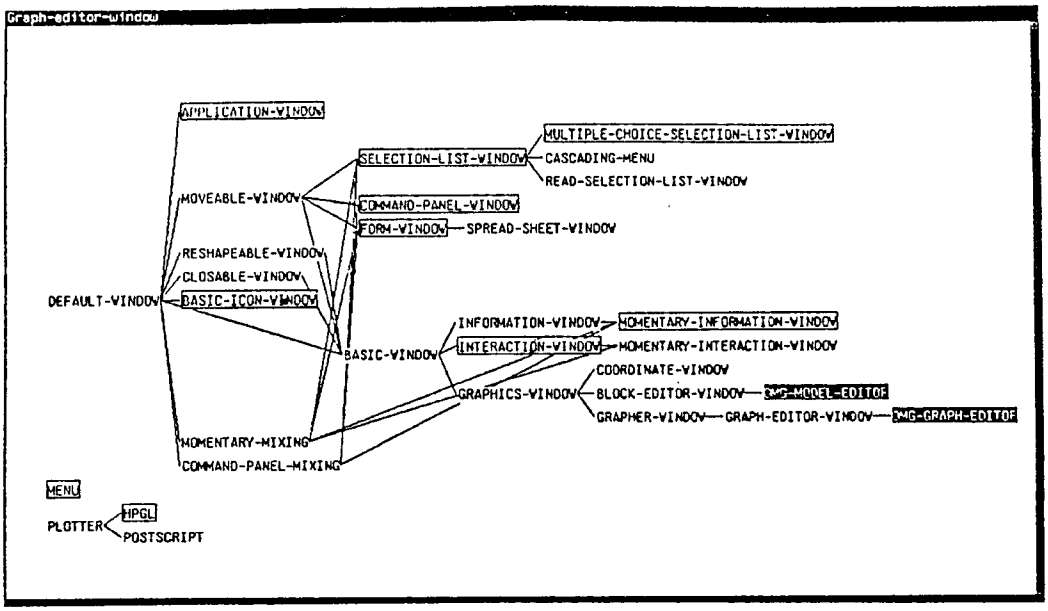

Figure 4: Class definitions of AIGT that are instantiated in DMG are boxed, and new DMG classes are shadowed.

3.2 Research Directions

In addition to further development of DMG and AIGT we are interested in applying automatic generation of hypertext to other design information. For these purposes we use models for the domain area (taxonomies, concept networks etc.) and for the tasks performed. The research areas related to this work are knowledge modelling and information science including natural language processing and hypertext related research.

The future work includes participation in the Esprit II project no. 2083, SIMPR, Structured Information Management: Processing and Retrieval. The SIMPR project is developing software for text indexing, subject analysis and classification, structured information management and interactive retrieval of large (400 MBytes) documents.

References

[1] *AIGT Application Interface Generation Tools,* Users guide, Nokia Research Center, Espoo, Finland, July 1989.

[2] Hamsher, W., and Davis R., "Issues in Model Based Troubleshooting," A.I. Memo 893, MIT AI Laboratory 1987.

[3] *Hyper Expert - A Short Introduction,* Version 1.2, Nokia Research Center, April 1989.

[4] Keene, S.E., *Object-Oriented Programming in Common Lisp, A Programmer's Guide to CLOS,* Addison-Wesley Publishing Company, USA, 1989.

[5] Lounamaa, P., Nurminen, J., and Tyrvainen, P., "DMG - A System for Diagnostic Modelling and Generation," Nokia Research Center, Helsinki Finland, March 1988.

[6] Milne, R., "Strategies for Diagnosis," *IEEE Transactions on System, Man and Cybernetics,* SMC-17 3, 1987.

[7] Robins, G., "Applications of the ISI Grapher," *The ISI Reprint Series,* ISI/RS-88-210, June 1988.

[8] Tyrvainen, P., "Reusable Object-Oriented Tools and Their Applications: AIGT - An Object-Oriented Interface Tool Library, DMG - An Application for Model Based Diagnostics," *Proceedings of First International Conference in Technology of Object-Oriented Languages and Systems (TOOLS'89),* Paris, November 13-15, 1989.

[9] Tyrvainen, P., "DMG - Model Based Hypertext Generation
for Practical Production of Diagnostic Advisory Systems," *The First European Conference on the Practical Applications of Lisp (EUROPAL'90),* Cambridge, March 27-28, 1990.