Knowledge Engineering, Expert Systems, and Real–Time Environments

N. Solntseff, A.D. Hurst and W.F.S. Poehlman
Department of Computer Science and Systems
McMaster University
Hamilton, Ontario L8S 4K1

This paper describes the work on artificial intelligence and real–time systems being planned in the newly formed Department of Computer Science and Systems at McMaster University. After a brief overview of the problems being considered, the goals of the proposed research are presented.

INTRODUCTION

The Department of Computer Science and Systems was formed in July 1985. It is in the process of improving its research resources through the hiring of additional faculty and the acquisition of research computers. A donation by Sperry Inc. of a SPERRY EXPLORER[1] workstation together with the KEE[2] Software Development System has provided the seed for the amalgamation of two research areas within the department, namely, those dealing with real–time data acquisition and artificial intelligence, into the Intelligent Real–Time Instrumentation Systems (IRIS) Group.

Knowledge engineering is one of the areas to be strengthened in the next three to five years and the goal of the IRIS Project is to investigate the application of knowledge–engineering techniques to the design and real–time control of data acquisition and other instrumentation systems for use in such applications as automatic laboratories, power stations, and particle accelerators.

The structure of currently available expert systems commonly consists of a *development engine*, used by the knowledge engineer to construct a *consultation environment*, and an *inference engine* which interprets the semantic structures in the consultation environment to produce executable code, most commonly Lisp or Prolog. The latter forms the *program* which is used by a client to obtain advice concerning an application area. This structure is directly analogous to that of a high–level language (HLL) translation system or a compiler in which HLL source code is first syntactically and semantically analyzed to produce intermediate code. This intermediate code is then processed by a code generator which yields the final executable machine language. This paradigm for expert systems can be used as a guide to the formal specification, description, and generation of expert systems.

---

[1]EXPLORER is a trademark of Texas Instruments, Inc.

[2]Knowledge Engineering Environment (KEE) is a trademark of Intellicorp.

This paper concerns the narrower problem of the design and construction of expert systems capable of improving the implementation and operation of real–time environments. The wider problem of the formal treatment of expert–system shells will be discussed elsewhere.

BACKGROUND

Modern real–time systems must have adequate high–speed performance. In order to be able to operate such systems effectively, the user should be familiar with their internal operation . People with these qualifications are not easily found and, as a result, real–time systems are rarely programmed to utilize the full capability of the available hardware. As hardware components take on more of the operations formerly performed by software, more detailed hardware knowledge will be needed on the part of instrumentation–system personnel and it is here that knowledge engineering can make a significant contribution through the automatic generation of code to operate the hardware [POE85, WIN84].

An example of such a system is the IBM shell for building expert systems to analyze operating–system performance [HEL85]. In this case, the expert system is used to analyze the results of performance measurements and to report the steps that the operator could take to improve the overall efficiency of the operating system. The next step, is to provide the expert system with the means (i.e., runnable programs) to change the system parameters in order to improve system performance. The goal of such a system would be to generate optimized software that would also be reuseable and, more importantly, adaptable to changing conditions [GOG86].

An environment of this type must be based on a standard peripheral bus e.g.,the Small Computer Systems Interface (SCSI) [WIN85]. It should also be possible to code time–critical operations at the lowest possible language level, e.g., at the microcode level when a writable control store is available. The present trend towards embedded systems implies there is a need for networking, which is being met at both the departmental [POE86] and university levels at McMaster by the provision of Ethernet connections to departments within the Faculties of Science and Engineering [ELO86], as well as locally within DCSS.

METHODOLOGY

During the initial phase of work, KEE will be used on the will EXPLORER as a high–level development engine to produce Forth–, LISP–, or Prolog–coded device handlers from a high–level real–time specification language, which is to be designed and implemented in as user friendly a manner as possible. The handlers are not just conventional interrupt–driven device drivers, but rather are virtual–device handlers capable of reporting an appropriate combination of signals from several actual devices within a designated time interval [POE83]. This approach will require substantial work on the creation of a syntax knowledge base, as well as a semantic knowledge base incorporating one of the usual models, namely, frames, decision trees, or object–oriented programming.

Forth is particularly suitable as an implementation language since not only does it have suitable real–time characteristics, but it is also highly extensible and very portable to a wide range of architectures. [SOL82, SOL83, SOL84] A great advantage of Forth over other expert–system languages is provided by the capability of a Forth development system to cross compile extremely compact code. Finally, it should be noted that Forth can be readily used for numeric computations unlike the non–numeric languages, such as Lisp or Prolog which require the presence of special numeric processors. Real–time–critical code can therefore be implemented directly in microcode [SOL79] to yield the highest performance possible.

The third aspect of the work is the development of techniques to download code generated by the development engine to a satellite processor system by means of a local area network such as Ethernet or the Manufacturing Automation Protocol (MAP). DCSS already has a number of machines (a VAX–11/780, three Pixel–80s, a number of PDP–11 machines, as well as many personal computers) connected together either by Ethernet or an RS–232 switch.

A long term goal is to replace KEE with a complete FORTH–based system which will have small memory requirements, be capable of running at speeds approaching those of pure machine code, and which will be highly interactive and easily extensible.

SUMMARY

The long–term goal of the IRIS project can be stated as follows: To investigate the application of formal syntactic and semantic techniques to the description of knowledge engineering software considered as a general language–translation problem. The short–term goal is to apply knowledge–engineering and expert–system techniques to the generation of code suitable for real–time instrumentation systems, namely:

(1) To use KEE to generate code in Forth, LISP, Prolog, POP–11, for use in a real–time environment;

(2) To implement Forth on the Explorer with a view of providing an environment for the creation of portable, high–performance development engines for expert–system such as KEE.

(3) To implement a run–time system on the Sperry EXPLORER in Forth whose compiler/interpreter is to be loaded into the writable control store of the host machine;

(4) To connect the Sperry EXPLORER into the DCSS network with a view of creating a distributed and concurrent real–time system;

REFERENCES

[ELO86] S. Elop, R. Shepard, and W.F.S. Poehlman, "ETHERNET -- The Engineering experience at McMaster", submitted for presentation at the Ontario Universities Computer Conference, (June 2–5, 1986).

[GOG86] J. Goguen, "Reusing and interconnecting software components," *IEEE Computer, 19*, No. 1 (February 1986), pp. 16–28.

[HEL85] J. Hellerstein and H. van Woerkom, "YSCOPE: A shell for building expert systems for solving computer–performance problems," IBM Research Report No. RC11463 (51517), October 22, 1985, 18 pp.

[POE83] W.F.S. Poehlman, R.G. Winterle, and D.R. Raymond, "Towards a general data acquisition system," Internal Report No. IDL–8301, Institute for Materials Research, McMaster University (September 1983), 10 pp.

[POE84] W.F.S. Poehlman, R.G. Winterle, and D.R. Raymond, "Asynchronous events in a generalized data acquisition system," *Proc. Fourth Real–Time Systems Symposium, Austin, Texas*, IEEE Computer Society (1984), pp. 208–211.

[POE86] W.F.S. Poehlman and C. Bryce, "McMaster's new Department of Computer Science and Systems networking and communication system," submitted for presentation at the Ontario Universities Computer Conference, (June 2–5, 1986).

[SOL79] N. Solntseff, "Experiences with a Portable Minicomputer Language", *INFOR, 17* (February 1979), pp. 52–57.

[SOL82] N. Solntseff, "An Abstract Machine for the Forth System", 1982 Rochester Forth Conference on Data Bases and Process Control, Rochester, N.Y. (May 1982), *Proc. 1982 Rochester FORTH Conference*, (October 1982), pp. 149–155.

[SOL83] N. Solntseff, "An Instruction Set Architecture for Abstract Forth Machines", 1983 Rochester Forth Conference on Robotics, Rochester, N.Y. (June 1983), *Proc. 1983 Rochester FORTH Conference*, (October 1983), pp. 175–183.

[SOL84] N. Solntseff and J.W. Russell, "An Approach to a Machine–Independent Forth Model", 1984 Rochester Forth Conference on Real Time Systems, Rochester, N.Y. (June 1984), *Proc. 1984 Rochester FORTH Conference*, (October 1984), pp. 121–139.

[WIN84] R.G. Winterle and W.F.S. Poehlman, "Asynchronous words for Forth," *Proc. 1984 Rochester Forth Conference*, (October 1984), pp. 32–41.

[WIN85] R.G. Winterle and W.F.S. Poehlman, "Using SCSI peripherals on DEC RT–11 systems," presented at the 18th Annual DECUS Canada Symposium, February 19–22, 1985, *Proc. DECUS Canada* (in the press).