S A Journal of Industrial Engineering, Vol. 2, No 2, December 1988 pp 15-24

EXPERIENCES WITH BUILDING A KNOWLEDGE SYSTEM AN APPLICATION IN INDUSTRIAL CONTROL

AR Greef and R Reinecke
Department of Industrial Engineering
University of Stellenbosch

ABSTRACT

The Centre for Robotics at the University of Stellenbosch set itself the objective of building a fairly complex manufacturing cell, including an unskilled human as a system component. As a precursor to this effort we constructed a simple cell requiring both supervision of a robot and supervision of a human using a Micro-Prolog knowledge system to do so. This was successfully done but at the cost of low execution speed and difficulty in integrating machine code instructions for machine control interfacing with the largely consultation orientated software which rendered the approach unsuitable for the more complex cell. MicroExpert was used for this new effort but we also tried Turbo-Prolog to mimick the inference engine of the former software. This experience made us decide to trade the higher speed achieved by Turbo-Prolog for the much more rapid knowledge base development in MicroExpert. To do so it was necessary to build a special frame data structure and to enhance Micro-Expert for supervising such manufacturing cells.

OPSOMMING

Die Sentrum vir Robotika aan die Universiteit van Stellenbosch is die taak gestel om 'n komplekse vervaardigingsel te bou wat 'n ongeskoolde mens as stelselkomponent sou moes insluit. As voorloper vir hierdie poging het ons eers 'n eenvoudige sel daargestel waarby toesig oor sleg 'n robot en 'n mens nodig was deur middel van 'n kennisstelsel gegrond op Micro-Prolog. Dit was suksesvol maar ten koste van betreklike lae uitvoerspoed en omslagtige instruksies in masjienkode vir die toesig oor die robot se werking, met die programmatuur was eintlik toegespits is op diagnose en konsultasie. Terwyl hierdie poging geslaagd was was dit duidelik dat die benadering nie so suksesvol op die beplande komplekse sel sou wees nie. Daarom is MicroExpert aangewend vir die finale sel waarby ons verder Turbo-Prolog gebruik het om die inferensie-enjin van Micro-Prolog na te boots. Hierdie ervaring het gelei tot die besluit om die huidige hoër spoed van Turbo-Prolog in te boet vir die gemak waarmee die ontwikkeling van kennisstelsels in Micro-Expert gedoen kan word. Die gebruik van MicroExpert vir die besondere toepassing het ons egter genoodsaak om 'n spesiale raam datastruktuur daar te stel en om MicroExpert verder uit te bou.

1.0 INTRODUCTION

The method of building a knowledge system (KS) is not an exact science but rather an experimental process of testing and modification until the desired results are obtained. To simplify this process, a number of construction steps, which have been developed through the experience of knowledge engineers, can be followed. These iterative steps are fully described in Hayes-Roth et al (1983) who deals with building large expert systems (500 - 3000 rules) and Harmon & King (1985) who recognize a distinction between large and small knowledge systems thereby providing construction guidelines for both.

What is evident is that knowledge system construction can use a large amount of time and money. It is important therefore to test ideas on KS's which allow rapid prototyping and easy modification such as shells and symbol processing languages. Once the best knowledge representation, control system and inference strategy has been determined a more specific construction tool can be used.

To build a micro-computer based knowledge system for control applications in an industrial environment we initially constructed a prototype KS. This Mark-1 KS was tested in a simplified control application to test its architectural feasibility and limitations. The prototype was then discarded in favour of a Mark-2 system. The Mark-2 version of the knowledge based controller, although not a complete system, does contain the components and organizational structure suited to solving the problems associated with control applications. What remains to be done is the streamlining of the control system's operation and the development of a knowledge base capable of solving a wider range of problems.

2.0 THE APPLICATION AREA

The machining sector of the manufacturing industry has become increasingly dependant on computer controlled automated equipment such as Computer Numerical Controlled (CNC) milling machines and robots for the production of low cost, high quality parts. In certain instances material transfer devices (eg. robots) and material transformation devices (eg. milling machines) may be synchronized to manufacture a set of products. The resulting system is termed a Flexible Manufacturing Cell (FMC) which is driven by two levels of control. The first level concerns the control of a device and the second level concerns the management of the FMC. The FMC is said to be data driven as manufacture is determined by information indicating the status of parts, fixtures, tools and devices. It is performed by executing robot task programmes and machine tool programmes on the device controllers. A change in the data thus results in a change in device and FMC operation.

Data driven manufacture, however, is not intended or suitable for supporting humans in fault diagnosis, error recovery and in the management of systems of automated equipment which requires logical problem solving. Manufacturing have needs to be both knowledge driven and data driven. This is particularly so in South Africa where automation should be used to support unskilled workers in skilled work. Industrial controllers need to be able to retain a skilled human's knowledge concerning a particular device or process, on site for use in supporting an unskilled worker in the initialization, operation, synchronization and maintenance of micro-computer controlled devices and FMC's.

A micro-computer based knowledge system was thus conceptualized which could supervise man and device in an industrial environment. The KS, termed a Device Supervisor

(DS), is required to interpret the current situation existing in the real world and thus instruct a device controller to execute a particular programme required to perform a material transformation or transfer task. A DS is piggy-backed onto each device and is considered as an extension of the device controller. FMC control is performed by linking the DS computers together with a network communications medium. This then forms a distributed FMC supervisory system. The DS integrates knowledge and data which is used to drive the FMC and support an unskilled worker in the FMC environment.

Two other micro-computer based KS controllers : HEXCON (Lattimer Wright et al, 1986) and SCD (Komoda et al, 1984) have been applied to problems involving the real-time control of industrial equipment. Their knowledge representation and architectural limitations does not allow them to manipulate control data. That renders them unsuitable for implementing the DS. Their organizational structures which enable the KS's to operate in real time, however, demonstrate the specific programming techniques which need to be considered when building a micro-computer based KS controller. These programming techniques are not needed with the more common micro-computer based consultation KS's.

3.0 BUILDING THE PROTOTYPE DEVICE SUPERVISOR

3.1 Tool Selection

The software tool selection criteria for the rapid development of a prototype KS is usually based on availability, the degree of user familiarity with the tool, the problems knowledge representation structural requirements, and the inference engine characteristics. The most rapid prototype development is obtained through the use of a KS shell which provides a knowledge representation structure and an inference engine so that knowledge concerning the problem domain can be immediately coded into the knowledge base and tested for validity. Most commercially available micro-computer based shells, however, tend to be designed as consultation systems which lack the communications interface features required for the DS control computer. The next best tool, therefore, was considered to be the Micro-Prolog[1] symbolic programming language. The raw Micro-Prolog environment could be used as a KS shell as it provided a built-in relational data base, a control system, inference strategy and knowledge representation structure. A logic/conventional interface was also provided for linking user defined routines to primitive predicates for performing data communications management. Micro-Prolog was thus used as a KS shell to implement the prototype DS.

3.2 Problem Identification

The prototype DS was designed to implement only a part of the complete DS function, namely that of supervising a single device. This was demonstrated by the use to the prototype to supervise a robot in a robot-centred FMC. As shown in figure 1, the DS was assigned to the robot controller and all other devices in the cell (a milling machine, lathe, camera and pneumatic chuck) were treated as peripheral equipment. The DS was required to solve three problems. The first problem was that of supporting a human during the initialization of the robot controller and the robot arm. This was performed by instructing the robot and human to perform certain initialization checks and tasks. Secondly, the DS was required to operate the robot by instructing it as to which programme to execute for performing a specific material transfer task. Thirdly, the DS was

---

[1] Micro_Prolog is a trademark of Logic Programming Associates Ltd

required to synchronize the robot and peripheral equipment motion during the loading and unloading of parts into and out of a device's fixtures. The FMC's operation is detailed elsewhere (Greef & Reinecke, 1987.)

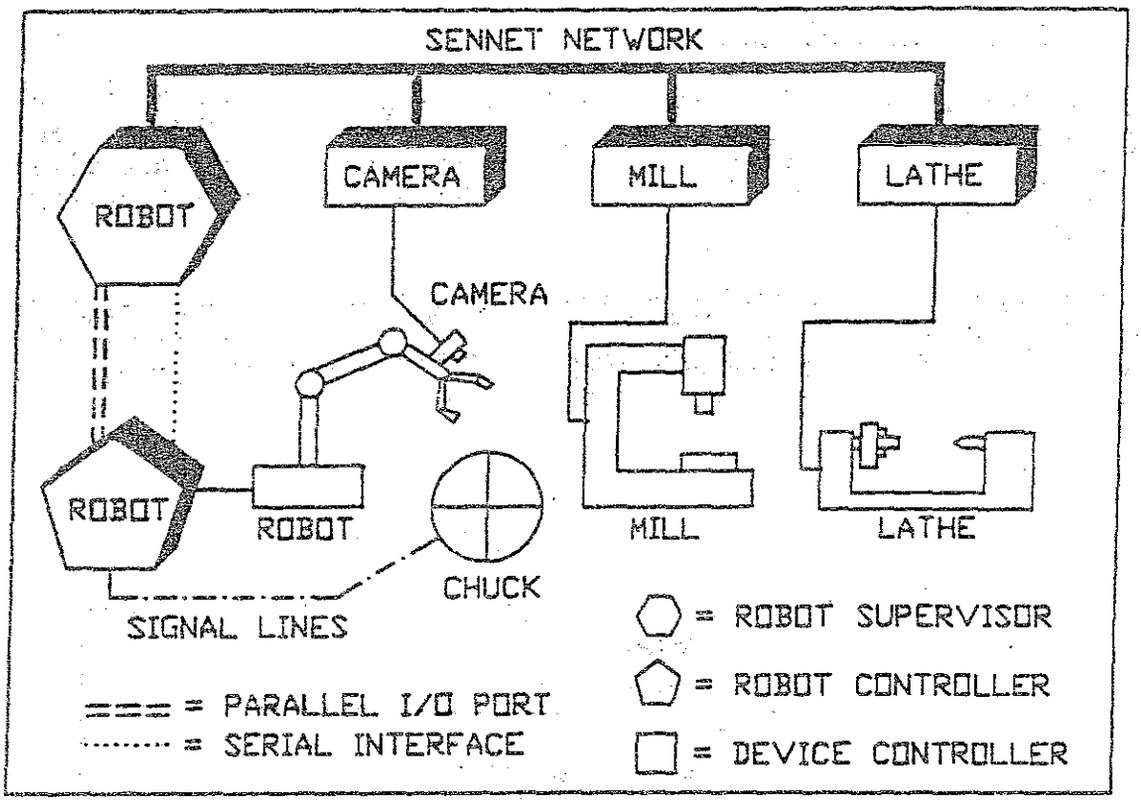

FIGURE 1 : THE PROTOTYPE DS AND FMC INTERCONNECTIONS

3.3 Conceptualizing the Knowledge

Both declarative and procedural knowledge was required for DS operation. The procedural knowledge consisted of three parts. One part was concerned with the initialization of the robot, the second with the operation of the robot and the third with the synchronization of the robot and the peripheral devices. This procedural knowledge was represented as Prolog Horn clause implications which could be given a procedural interpretation. An example of a Horn clause implication is :

```
is( robot_gripper, loaded) if
is (robot_gripper, not_loaded) and
is (instruct_robot, pick_up_gripper).
```

The declarative knowledge consisted of a static and dynamic part. The static part formed a data base of facts, such as which grippers were available for the robot. These facts remained constant during robot operation. An example static fact is :

robot_gripper(flat_gripper).

The dynamic part of the data base formed the world model which represented the state of the robot (eg. up, down, idle, busy), its components (eg. which gripper was attached) and its jobs to be performed. An example dynamic fact is :

robot_state(idle).

This declarative knowledge, represented as unary Horn clauses could be modified from within the procedural logic rules.

3.4 Implementing the Knowledge System

Assembler coded procedures, for transmitting and receiving data via a serial and network interface, as well as for setting and detecting signals appearing on a parallel port, were linked into the Micro-Prolog interpreter. This created a number of primitive predicates which transfered parameters and control to, and which received instantiated variables from, the interface routines.

The solution space of the DS could be depicted using an AND/OR graph as shown in figure 2 and subsequently translated into logic rules and facts. At power up, the DS polled the keyboard and serial interface continuously until the user was ready to start.

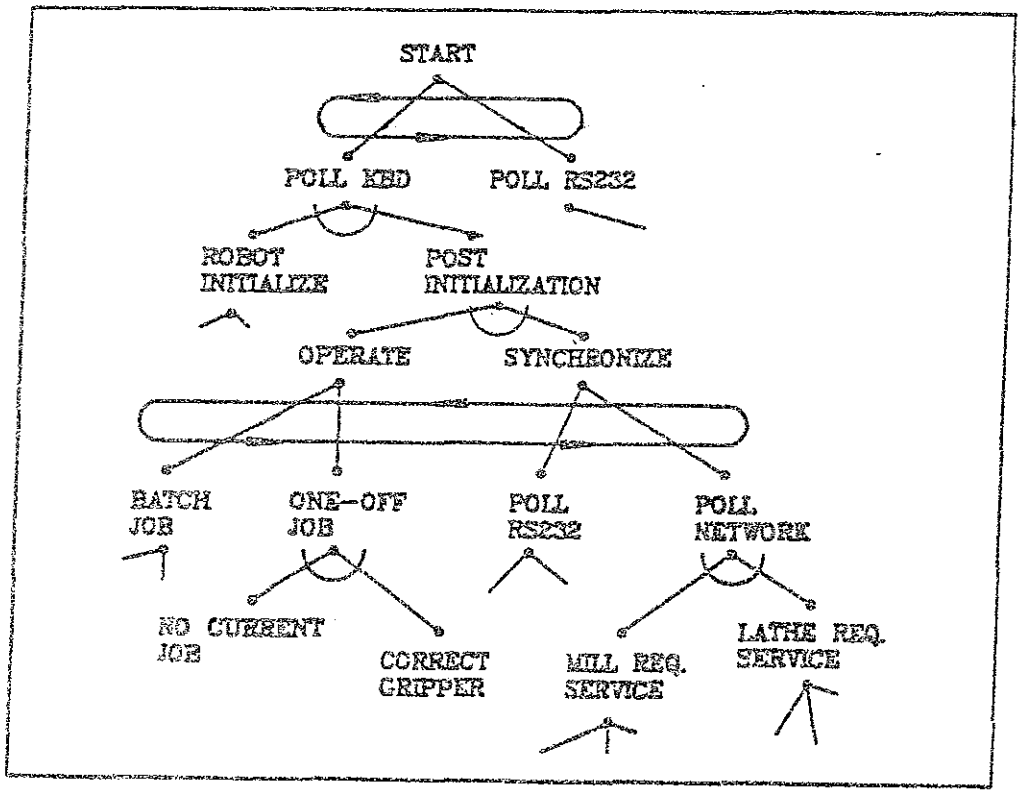

FIGURE 2 : THE DS's PARTIAL AND/OR GRAPH

The DS then leads the human through the initialization routine until the robot controller and arm are initialized. Once initialization is complete, the DS continuously polls the serial and network interfaces, and the internal job queues. Messages arriving on the interfaces are used to synchronize the robot or update the world model and batch queues. The operation of the robot is dependant on the state of the job queues and the world model. Micro-Prolog's backward-chaining, depth-first control system searches the rules and facts in the knowledge base and attempts to evaluate the operation and synchronization goals. This goal driven process drives the supervision of the robot during the manufacture of parts.

3.5 Assessment of the Prototype Knowledge System

The prototype Mark-I DS demonstrated the feasibility of using a KS for the supervision of a device in a device-centred FMC. Rules and facts proved adequate for representing the procedural and declarative knowledge required for the initialization, operation and synchronization of a device. Using Micro-Prolog as a KS shell, furthermore enabled the knowledge part of the DS to be rapidly constructed.

There were however, a number of drawbacks associated with using Micro-Prolog. As it is an interpretive language, it has a slow execution speed. The interface message handling routines had to be written in machine code and were difficult to interface to the Micro-Prolog environment. McCabe et al (1984) suggests that due to these difficulties, complex data structures such as lists should not be passed from a user routine to a Micro-Prolog primitive predicate and that only constants and numbers should be passed. This limits the amount of data which can be passed between the conventional and logic parts of the knowledge system which is a severe limitation when working in an information rich environment. The continued use of Micro-Prolog as a shell was further negated by comments from the literature such as:

> "Prolog is a programming language, just as LISP is. It is not a knowledge representation language, ..." (Jackson, 1986, p. 184), and
>
> "in most cases Prolog will act as a good vehicle for 'knocking off' a cheap prototype before building an efficient system, using the appropriate tools." (Yazdani, 1984, p. 107)

Ultimately, the undesirable points of Micro-Prolog outweighed its desirable points leading to it being abandoned as a vehicle for the construction of the Mark-2 version of the DS.

4.0 BUILDING THE Mark-2 DEVICE SUPERVISOR

4.1 Reconsidering the Tool

The prototype KS showed that a simple knowledge/conventional interface as well as a procedural and declarative knowledge representation structure was required for the operation of a DS. An expert system shell, MicroExpert,[2] implemented in PASCAL and

---

[2] MicroExpert is a trademark of Micro Expert Systems

designed as a developmental package, provided such a simple knowledge /conventional interface, a production rule knowledge representation structure and an inference engine. There were however, no facilities for holding and manipulating declarative knowledge. Therefore, as it is much easier to design and use knowledge structures in Prolog, which leads to rapid knowledge base development, a Prolog compiler, Turbo-Prolog, was also considered for use in developing the the Mark-2 DS.

As a comparison between conventional and symbolic programming languages , MicroExpert's knowledge representation structure and inference engine was mimicked using Turbo-Prolog.[3] Although there was a large reduction in the number of programming lines (approximately 800 in Turbo-Prolog compared to 2400 in Pascal) there was no noticeable difference between the two shells' execution speeds. The decision of which language to use, then, was a play-off between rapid knowledge representation development in Turbo-Prolog with a slower, cumbersome logic/conventional interface development; and slower know-ledge representational structure development in PASCAL with an existing, simple knowledge/ conventional interface. Based on previous experience with Micro-Prolog and the fact that the PASCAL system could be structured to execute at a much faster rate, the MicroExpert shell was chosen as the development package for the Mark-2 DS.

4.2 Expanding the Problem

The prototype DS integrated data and knowledge driven manufacturing at the device level of control. The expanded problem entailed the integration of data and knowledge also at the cell level of control. A demonstration FMC was constructed as shown in figure 3 which could manufacture any one of 54 different key-rings, in any order and at any time. A DS was assigned to supervise each device in the cell and co-operate in the distributed supervision of the FMC. The Mark-2 DS would indicate the feasibility of using a KS in this control application and show the architectural limitations of the enhanced MicroExpert shell.

4.3 Enhancing the Knowledge Base

MicroExpert provided a production rule knowledge representation structure which comprised attribute-value pair condition and conclusion clauses. For example :

```
IF  robot gripper IS not loaded
AND robot instruction IS pick up gripper
THEN robot gripper IS loaded
```

Futhermore, PASCAL functions could be executed from within a rule's condiction clauses and procedures from within a rule's conclusion clauses. This implemented the procedural knowledge representation structure and the knowledge/conventional programming interface.

A simplified frame data structure was built to collect related data together for a more expressive declarative knowledge representation and data base. An example frame is :

```
FRAME robot,
  gripper1 = flat gripper,
  gripper2 = round gripper
```

---

[3] Turbo-Prolog is a trademark of Borland International Inc

The frames comprised a header (eg. robot) by which it was indexed and slots (eg. gripper1 = flat gripper) consisting of attribute-value pairs. As well as representing static and dynamic declarative knowledge as in the prototype KS, it was now possible to hold FMC related data such as part data and production statistics in the frame structure. Functions and procedures, invoked from within the rules, enabled frames to be created, deleted, modified (by changing a slot's value) and transmitted over the serial and network interfaces. Frames received over the interface ports were added directly to the knowledge base.

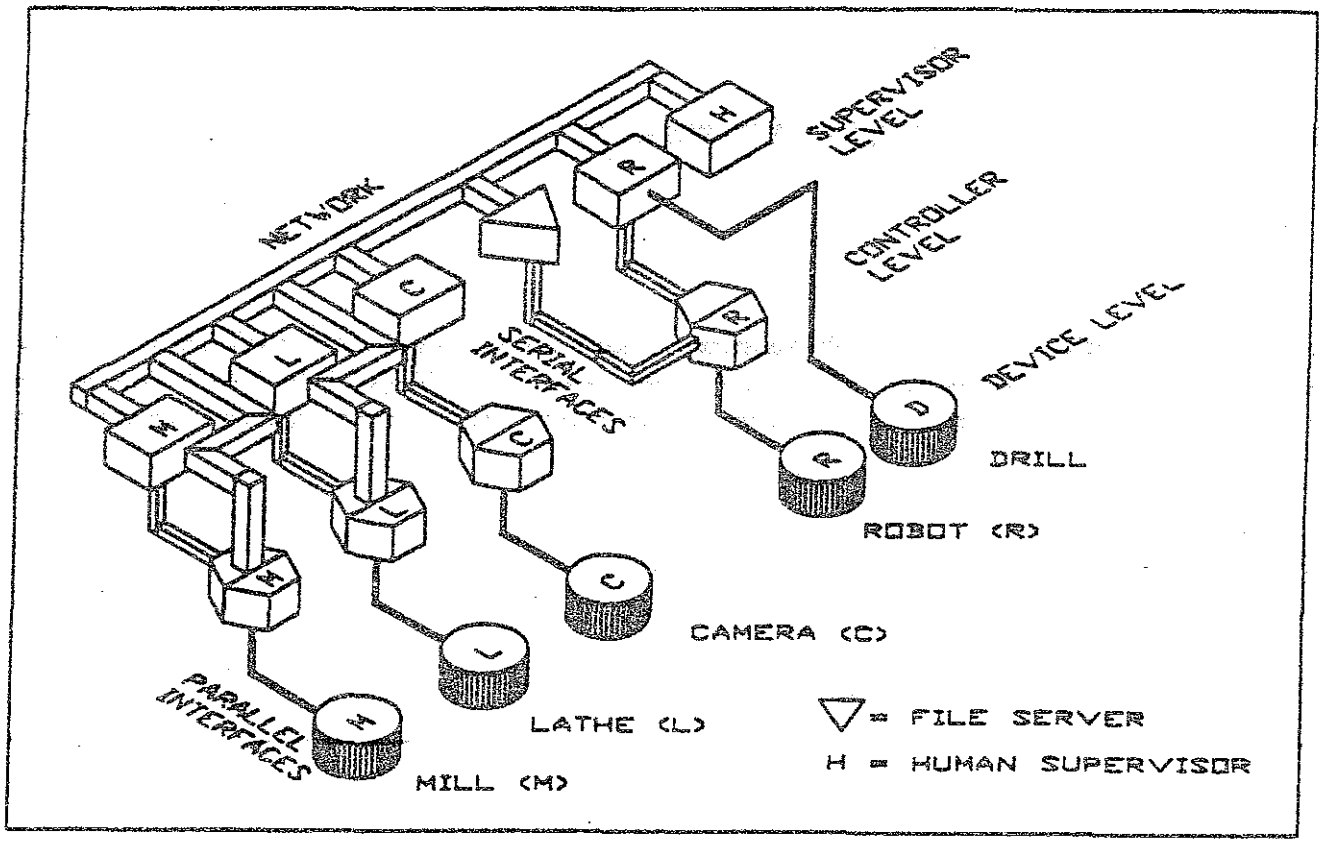

FIGURE 3 : THE Mark-2 DS AND FMC INTERCONNECTIONS

A 'single-fact' internal data structure was designed to hold message information received over the interface ports. A symbolic name was assigned to each I/O part which formed the attribute of the value message received. The attribute-value pairs were then retained in the knowledge base for matching against the rule's condition clauses. Messages were transmitted to the interface ports by rule invoked functions.

4.4 Re-implementing the Device Supervisor

MicroExpert's inference engine was enhanced to force the control system to verify an unknown attribute by looking in the message and frame data base if a rule could not be found to evaluate. If an attribute could not be matched in the data base then the KS prompted the user for input. Using this control strategy, the KS continuously assessed the current world representation modelled in the data base and reacted accordingly.

The FMC was driven by part data held in frames which were transferred and transformed along with the transfer and transformation of the parts. Part data contained a parts type, order number, quality, location and destination. This information was sufficient for the DS's to coordinate the production of the key-rings. Messages were exchanged between the DS's to indicate device states and to synchronize interacting devices.

The knowledge contained in each DS performed the initialization, operation and synchronization of each device and ultimately the entire FMC. This required an expanded DS knowledge base so as to encompass the knowledge and data driven requirements of the FMC.

4.5 Assessment of the Mark-2 Device Supervisor

The DS was kept simple by retaining separate facts and rules knowledge bases. Both knowledge and FMC driven data were adequately represented and manipulated using frames and production rules. These data structures gave the knowledge base a more explicit appearance. The knowledge/conventional programming interface allowed a number of knowledge representation structures, data handling and control strategies to be rapidly constructed and tested.

Each DS's knowledge base contained less than 90 rules. One rule was required to hold each one of a device controllers programme activation conditions and one rule was required to hold each of a device controllers programme termination conditions. This resulted in the large number of rules which could be reduced at a later state through the use of variables, as in the SCD controller. This would decrease the amount of rules in a knowledge base and increase the number of facts held in the frame data structure thereby making system operation more efficient. Reaction times of the KS ranged from 5 to 8 seconds. In terms of the machining times actually needed in the FMC this was effectively real time operation. For significantly shorter cycle times of procedures this Mark-2 controller would be slow although we believe that the current personal computers using 80286 and 80386 processors would bring reaction times to under one second. That implies real time control for quite short cycle operations. Choosing PASCAL as the KS implementation language, however, allowed for the future implementation of more efficient data structures and knowledge compilation techniques as used in the SCD and HEXCON controllers (also see Hayes-Roth et al). This could not be done with a Prolog system.

5.0 CONCLUSION

When building a small KS, a tool should be selected which allows rapid prototyping of the system. This enables a knowledge representation structure and inference engine to be quickly constructed for testing the problem suitability for solution using KS techniques. For control applications, where as much emphasis is placed onto the data manipulation and interchange as is on the reasoning process, it is best to use a KS development tool which allows for the rapid construction of the knowledge and conventional parts of the KS. This is essential as the construction of the KS is an iterative process which means constantly revising and modifying the original KS and concepts.

6.0 REFERENCES

Greef, A.R., Reinecke, R., "Logic for Robot Programming and Control", Interdisciplinary Conference on Mathematical Logic and Related Subjects, ITERLOGICON 87, University of Natal, Durban, July, 1987.

Harmon, P., King, D., "Expert Systems. Artificial Intelligence in Business", John Wiley & Sons, Inc., New York, USA, 1985.

Hayes-Roth, F., Waterman, D.A., Lenat, D.B., "Building Expert Systems", Addison-Wesley Publishing Co., Massachusetts, USA, 1983.

Jackson, P., "Introduction to Expert Systems", Addison-Wesley Publishing Company, Inc., 1986.

Komoda, W., Kera, K., Kubo, T., "An Autonomous, Decentralized Control System for Factory Automation", IEEE Computer, December, 1984.

Lattimer Wright, M., Green, M.W., Fiegl, G., Cross, P.F., "An Expert System for Real-Time Control", IEEE Software, March, 1986.

McCabe, F.G., Clark, K.L., Steel, B.D., "Micro-Prolog 3.1, Programmer's Reference Manual, MSDOS version", Logic Programming Associates Ltd., London, UK, 1984.

Yazdani, M., "Knowledge engineering in PROLOG", in Forsyth,R., "Expert Systems. Principles and case studies", Chapman and Hall, London, UK, 1984, pp 91-111.