The International Joint Conference on Artificial Intelligence

August 1989, Detroit, U.S.A.

Workshop 4 : Integrated Architectures for Manufacturing

AN INTEGRATED OBJECT-ORIENTED ARCHITECTURE FOR MANUFACTURING KNOWLEDGE SYSTEMS

D.H. Norrie(*+), O.R. Fauvel(*), D.T.Guo(*), B.R.Gaines(+)

Department of Mechanical Engineering (*)
Department of Computer Science (+)
The University of Calgary, Calgary, Alberta, Canada T2N 1N4

INTRODUCTION

The system being developed is envisaged as a complete interactive system comprised of modules which carry out process, planning, cell grouping, scheduling, simulation and other functions, with each module having its own 'rules' or expertise embedded within it. The user interacts through a supervisory module with the process planning, cellgrouping, scheduling, simulation and other modules which also interact with each other through their submodules. Each module, submodule or other component of the system is an object, with interaction by messages sent from object to object. The whole would appear to the user as an 'intelligent' system. At a later stage, a design system would be interfaced with this planner to facilitate integration of design and manufacturing planning and to allow part designs to be downloaded. As well as the part geometry, material tolerance, surface finish and so on, the part description in terms of features would be included since there is a strong correlation between a part feature (prismatic surface, hole, etc) and the manufacturing operation required for it.

GENERALITY, PARTICULARITY AND INHERITANCE

For a class of objects, such as CNC_Machine, the general properties and capabilities will be defined at the Class level and instances of the class may inherit these. Every effort will be made to retain generality as far down through the system levels as possible and to introduce particularities at the lowest levels practical. One cogent reason for doing this is to allow new objects such as planning, schedules, machines and AGVs to be later introduced into the system with the minimum disturbance (i.e. changes) to the higher levels.

THE OBJECT KNOWLEDGE BASE AND INFERENCING

The same principle of specifying the particularities at the lowest level possible will be used in structuring individual objects. In general, within each object, the specific 'knowledge' (data) for the object will be separated into a *facts* sub-object (e.g. an Array) and a *rules* sub-object (containing the specific 'rules' (procedures) for the object's actions). Together, the facts and rules (data and procedures) would comprise the knowledge base in the expert system sense. What corresponds to the inference engine

code would be elsewhere within the object (for a very specific kind of interpretation) or inherited from a higher level (for more general kinds of interpretation or inferencing). To illustrate, an instance of a CNC_Machine could have facts such as manufacturer, model number, maximum horsepower and maximum height of a part, stored in anArray and rules such as [(IF)'spindle is set to on' (AND) 'part program begins' (THEN) 'close safety gate'] stored in some other convenient data structure. Since the-rule set may simply require consecutive execution, the 'inference engine' may only be a sequential line-by-line interpreter. In contrast, a process planning object may have a backward-chaining inference engine which makes use of the whole knowledge base of the object i.e. both facts and rules. It will be seen that each object can have, in effect, its own expert system incorporated for 'deciding' how to act or react. The advantage is that not only each object can have its 'expert system' embedded, as simple or sophisticated as needed, but that this capability can be inherited from a higher class. To change the characteristics of a particular object, may require only an appropriate change to a single knowledge base.

MODULES AND OBJECTS

For the process planning and cell grouping modules currently being worked on, the process planning and cell grouping objects would carry out the following functions:

Process Planning: This module would determine for a part the operations to be carried out, which machines are suitable for each operation, fixtures and tools needed, operation times and so on.

Cell Grouping: The procedure for cellgrouping would include setting up a parts-machines matrix, processing this matrix to show potential machine clusters for part groups, and defining the cells from the machine clusters taking other relevant factors into consideration.

For the simulation module, the following would apply:

Simulation: In an object-oriented simulation, components such as a machine tool or automated guided vehicle (AGV) would be an object which stores data about itself and responds to messages from other objects. The whole simulation therefore would comprise an interactive process by which parts, machines, AGVs, planners, schedulers, and so on interact with each other through messages in a sequence or order depending on individual conditions. For example, if a machine tool found that a part was prismatic it would message a particular scheduler but if the part was rotational a different scheduler would be addressed. A supervisory object could be messaged to request higher level instructions e.g. "Have broken cutter, what to do now?"

The simulation process envisaged would be dynamic with more similarities to an ecological system than to a hierarchical model. A key advantage to the object-oriented approach, especially where a language such as Smalltalk is used, is that new objects and even whole classes of new objects can be grafted into the simulation with minimum disruption previously existing system. This offers the possibility of beginning with a small system covering say, process planning, and progressively extending it over all aspects of a factory. The ultimate aim is to simulate a complete factory system e.g. an existing plant (large or small) or a factory-of-the-future. The initial simulations would not be described as intelligent but since adaptability (using, for example, rules in a knowledge base) could be incorporated into objects, such simulations might well be precursors of later 'intelligent' systems. While current simulations would run on a single CPU computer, future ones will run on parallel processing systems. New developments in the

'virtual time' concept (References 1, 2) offer attractive possibilities in parallel processing for object-oriented simulations.

STRUCTURE OF AN OBJECT-ORIENTED KNOWLEDGE SYSTEM

The design of an object-oriented knowledge system for manufacturing needs to proceed from the top down as well as from the bottom up. In the preceding sections, the structure of the basic building block, the Rule-Based Object has been outlined, together with its relation to function and property. It is now appropriate to consider the structure of the system further, from the top down.

In an integrated manufacturing knowledge system, there are many functions such as process planning, cell grouping, scheduling and simulation to be included. At the top level, the system is divided into major modules according to these general functions.

System Control - higher system level functions

Application Model - application-specific functions and entities

Interface - system interactions with user, hardware, other systems

From an implementation point of view, the grouping into subsystems is important. System Control and the Interface are not application related and could therefore be developed as a general-purpose shell. Only the Application Model is specific to the application and could be modified for different purposes.

In a subsequent publication, the way in which these three subsystems can be implemented will be described. For the Smalltalk implementation now being developed the approach under consideration has some similarities to the Model-View-Controller (MVC) technique used therein. Alternative groupings and their implementation are also being examined, particularly with respect to system extendability. Potential user interfaces such as Object Sequence Browser, Class Hierarchy Browser and Why and How User Interfaces are also being reviewed.

SYSTEM OPERATION

Consider the sequence of operation for an object which receives a request (message) to accomplish some task which for example, might be to carry out a process planning function or search for and return specified information. The incoming message triggers the corresponding method (procedure) which, say, calls a particular inference engine to execute the object's own rule-set, beginning with the first rule. Part way through the rule-set, the inferencer encounters another rule directing a transfer to another rule-set (in object of another class) for a certain sequence and to then return. Further down the rule-set, a similar branching requires the use of rules in an object higher or lower in the hierarchy of the same class. Yet further down the rule-set, a supervisor is appealed to and (say) 'decides' that a different inference engine should now be used with some other rule-set(s) to obtain needed information, following which the original sequence resumes. In principle, any object can be given methods to request action(s) of any other objects, to call inferencing engines or other more general 'reasoning' objects, to reference facts or rules, and to have any appropriate sequence carried out. The value of object hierarchies is evident since the more general methods, facts and rules can be specified at the higher levels and these used in association with the more specific

procedures and data be given at lower levels of the hierarchy.

THE DATABASE

For the database for the system, there are at least three possibilities: (a) the object-oriented language's own capability for storing data (e.g. in Smalltalk, using arrays for data storage), (b) a relational data base manipulating tables or arrays of data storel and (c) a fully object-oriented data base. It will be noted that

although facts and rules are conceptually shown attached to an object, these are actually stored in a database and only references to these are retained in an object. While a relational database can be used, an object-oriented database such as GEMSTONE or G-BASE has major advantages. In a large system, a similar approach could allow most of an object's methods to be kept on secondary storage. Extending this concept further for very large systems would leave only a module 'index' or 'reference' object in the system, with the module and its component objects in the database available for retrieval as needed.

REFERENCES

1. Jefferson, D. et al, *Distributed Simulation & the Time Warp Operating System*, Operating Systems Review, vol. 21 (5); pp 77-93, November 1987.

2. Lomow, G., Cleary, J., Unger, B., *A Performance Study of Time Warp*, Distributed Simulation 1988, pp 50-55, Society for Computer Simulation, San Diego, February 1988