UDC681.3.02.001:658.512:621.771

Development of the Plate Finishing Control Support System Using the Planning Expert System Building Tool K1

Toshimitsu Baba*1 Yoshiteru Matsuo*2

Abstract:

This describes the planning expert system building tool "K1" and the Plate Finishing Control Support System (PFCSS) developed by Nippon Steel using K1. K1 has an original inference engine able to reason fast enough for a large-scale, advanced-function knowledge base by utilizing an original pattern-matching algorithm. A new knowledge base architecture makes positive use of the object-oriented approach while representing knowledge in the conventional if-then format. The descriptiveness and processing efficiency of production rules in the rule base are improved by allowing C++ code, an object-oriented languages. K1 itself is also implemented in C++ for portability among different platforms. The PFCSS is designed to efficiently control the flow of plates from the upstream rolling line to the downstream finishing line in accordance with real-time operating conditions. Through the development of PFCSS, K1 has achieved greater benefits than other commercial expert system building tools.

1. Introduction

Improving in productivity and efficiency is an important challenge for large-scale manufacturing companies. To meet this challenge, the manufacturers have widely adopted computer-integrated manufacturing (CIM). The steel industry has long aimed at automating processes by using computers. With the progress achieved by process technology in recent years, the role of computers has become ever more important.

While in the conventional CIM environment, computers basically processed data, today's computers are required to bring intellectual judgment to problems. Expert systems are one branch of artificial intelligence and are designed to perform intellectual processing in place of humans[1]. Expert systems rapidly spread during the 1980s when low-cost, high-performance engineering workstations and shells (expert system building tools) appeared. Early on, the steel industry expected expert systems to be practical and effective, and therefore aggressively developed expert systems. During this development, Nippon Steel found that using commercial shells with the primary objective of versatility was not effective for solving actual problems within the company.

In 1991, Nippon Steel circulated a questionnaire survey entitled "Bottlenecks in the Development of Expert Systems". The survey targeted inhouse expert systems. The results of the survey are shown in **Fig. 1**. Most problems concerned knowledge representation, user interface, and processing speed. The forms in which the expert systems are implemented are shown in **Fig. 2**. Reflecting the increasing size and complexity of problems, few expert systems were implemented by the classical rule-based method alone. In light of the trends of the time, many were

*1 Head office

*2 Electronics & Information Systems Division

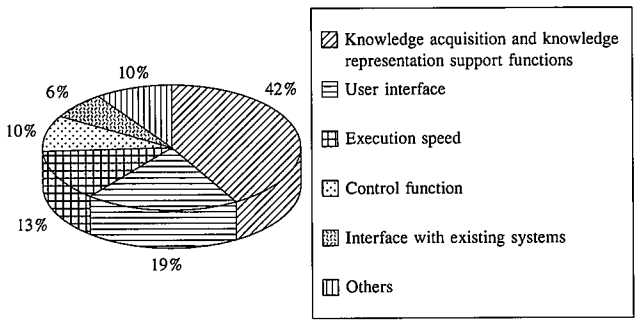
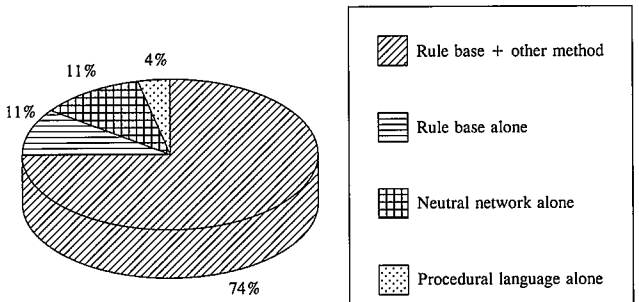
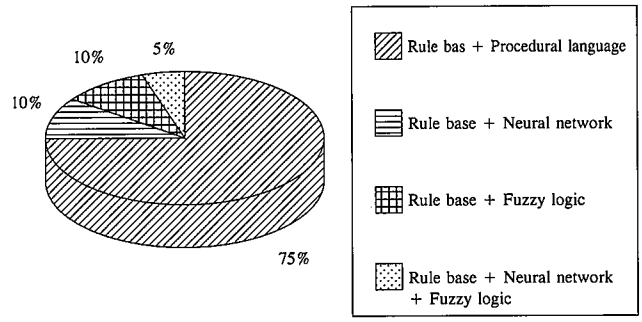


Fig. 1 Bottlenecks in development of expert systems



(a) Forms of implementation



(b) Techniques used in combination for building rule-based expert systems

Fig. 2 Forms in which expert systems are implemented

combined with procedural languages FORTRAN or C, or with fuzzy logic and neural networks.

Analyzing the expert systems from the standpoint of application areas, many of the early systems were analytical-type systems such as fault diagnosis systems. The Systems Research & Development Center of Nippon Steel's Electronics & Information Systems Division developed the "Diag-Solver" analytical expert system building shell and used it to build several expert systems[2,3]. Diag-Solver uses simple knowledge representation in diagnosis and has a flexible inference engine. It can be used by field operators and is designed for the efficient development and easy maintenance of practical expert systems.

Planning expert systems have as yet no general-purpose techniques for solving problems. Hypothetical reasoning (as an artificial intelligence technology) and optimization and mathematical programing (as operations research technologies) are general algorithms for planning expert systems. When planning expert systems start to solve problems, they encounter such challenges as limitations in computer memory size and difficulty in formulating problems. Though constructing practical applications is no easy task, there is an increasing need for synthetic-type expert systems that include planning-type systems.

Because of the above trends, the authors developed their expert system building tool K1 with powerful object-oriented knowledge representation and fast inference performance.

2. K1

2.1 Development Goals

2.1.1 Improvement in descriptiveness of rule-based programming language

The production rule in the if-then format uses a knowledge representation method already popular in the field and is sufficiently knowledge descriptive to be applicable to the control and management of future integrated expert systems. However, it is very difficult to represent an entire expert system using only production rules. The construction of a practical expert system requires the combination of structured knowledge and procedural knowledge. Therefore, the rule-based descriptive language of K1 referred to the object-oriented language C++[4,5] and used its characteristics.

An example of the descriptiveness of structured knowledge is shown in **Fig. 3**. In the example, 'product' is defined as a subclass of 'aPlate', has its own slots (variables) 'contentC' and 'contentSi', and inherits the parent slots 'width' and 'height' and the method 'mkdaemon'. The merits of object-oriented knowledge representation are also manifest in the development phase of expert systems[6].

Generally, in the development phase of systems, specifications are frequently changed. This tendency is particularly marked in the building of expert systems. Programming with an object-oriented language can meet this requirement more flexibly than programming with a procedural language.

Many of the knowledge representation units of K1 appear as classes in object-oriented languages, so that the merits of object-oriented system development can be put to effective use in the analysis, design, and implementation phases of expert systems.

2.1.2 Compatibilty with existing systems

In actual system environments, many software programs run on different computers in a mutually beneficial way. When commercial shells are utilized, constraints often occur in integrating an expert system with an existing system. Most of the constraints arise from the closed system architecture of the shell

```
defwmeclass aPlate {
   class:
      int    counts;
      void  mkdaemon() {
                  cout << ++c ounts << "created.¥n";
               }
   instance:
      double width;
      double length;
      double thickness;
};

defwmeclass productA : aPlate { // Inheritance...
      double contentC;
      double .....contentSi
};
```

Fig. 3 Example of working memory element class

and the characteristics of the language used to build the expert system.

K1 offers the C++ direct coding function that allows statements in the general-purpose development language C++ to be directly described in any desired portions of the knowledge base. This fusion with the C++ language allows open system-type expert systems to be easily built. A typical example of direct coding is given in **Fig. 4**. The section between symbols ??(at the beginning of one line and symbols ??) at the beginning of another represents a statement in C++. The C++ direct coding function is advantageous in that it can be used not only for integrating an expert system with an existing system, but also for describing procedural knowledge in the knowledge base.

Furthermore, K1 is generated as one class of C++ language, allowing the expert system to be easily controlled from an outside system by message passing. Conversely, classes defined in C++ can be directly incorporated into the knowledge base by declaring them as user-defined class types in the knowledge base.

When the user defines a knowledge unit (working memory element) class that calls for a data structure with quite complex functions, the user can define the class as a C++ language class and import it into the knowledge base as such.

In this way, K1 is a flexible means for combining with existing systems and external modules developed in other languages.

2.1.3 Fast inference engine

Generally, the function of the inference engine is to compare the condition part of knowledge (production rules) stored in the knowledge base with the data in working memory and to fire the production rules in a chain-like manner. This pattern matching consumes a huge amount of memory and computing time, important performance factors in a real expert system. This is the reason why many researchers have sought fast pattern-matching algorithms.

Many of today's high-performance inference engines are based on the RETE pattern matching algorithm[7]. The RETE algorithm is well accepted as a general-purpose pattern-matching method, but has been improved in various ways and is available in several variations. The Systems Development & Research Center developed a pattern-matching algorithm that is faster than the RETE algorithm and needs less memory, especially for large-scale combinatorial problems[8]. This algorithm consists of the following three internal modules:

DTREE	: Digital search
TREE, ONET	: Optimized inference NETwork,
CR	: Conflict Resolver

```
rule move
{
      $P( product status == unprocessed );
      $A( materialA );
      $B( materialB );
      # ( $A. typeCode == $B. typeCode );
      [ $P.deadline ]
=>
      ??( // C++ Direct Coding ...
            int       remains = $P. counts - 1;
            cout << remains<<"remaining.¥n";
      ??)
      modify( $P  status = finished );
      remove($4);  remove($B);
}
```

Fig. 4 Example of C++ direct coding

DTREE reduces the amount of unnecessary pattern matching for a working memory element. DTREE is based on the idea of a digital search[9]. Digital searching originally addressed character strings alone, but is expanded here for matching the returned values of functions. ONET performs the pattern-matching process at high speed using a special network structure focused on pattern-dependent relations. CR takes charge of conflict resolution and selects the rule to be fired next from among the working memory altered by DTREE and ONET.

A unique feature of K1 is that the sort function, the most basic function for planning and scheduling problems, is used in the inference engine. Standard rule-based descriptive languages have no internal sort function and call for the user to create such a function by combining rules.

The results of inference performance tests conducted using the rules shown in **Figs. 3** and **4** are given in **Fig. 5**. The tests focus on a simplified scheduling problem. When the number of products to be made is input, K1 searches for the combination of "material A" and "material B" required to make the product, and provides a solution. In combinatorial problems like this one, K1 has a higher reasoning performance than the RETE algorithm. This test involves very simple combinatorial rules, but patterns of this type frequently appear in actual expert systems.

2.1.4 Portability

Expert systems operate on a variety of platforms, from personal computers to mainframes. For this reason, K1 is designed for portability as well. K1 can generate the application source code in C++ language through a dedicated translator, and the source code does not depend on any particular platform.

Furthermore, K1 itself uses C++ language. For platforms on which the C++ language compiler cannot operate, C++ source code is transformed back into C source code. The C source code can be ported with minor modifications to platforms on which only the C language compiler operates.

2.2 K1 development environment

The K1 development environment, which consists of five modules, is schematically depicted in **Fig. 6**.

2.2.1 Translator

The translator has unique language specifications and transforms the knowledge base into a C++ source code. Knowledge represented by production rules is not translated into procedural knowledge. Instead, each knowledge element is reconstructed into an appropriate structure. For example, a data type like the working memory element class (wmeclass) is

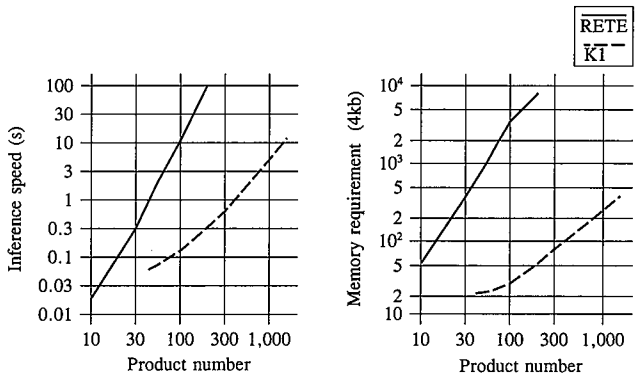


Fig. 5 Results of performance test

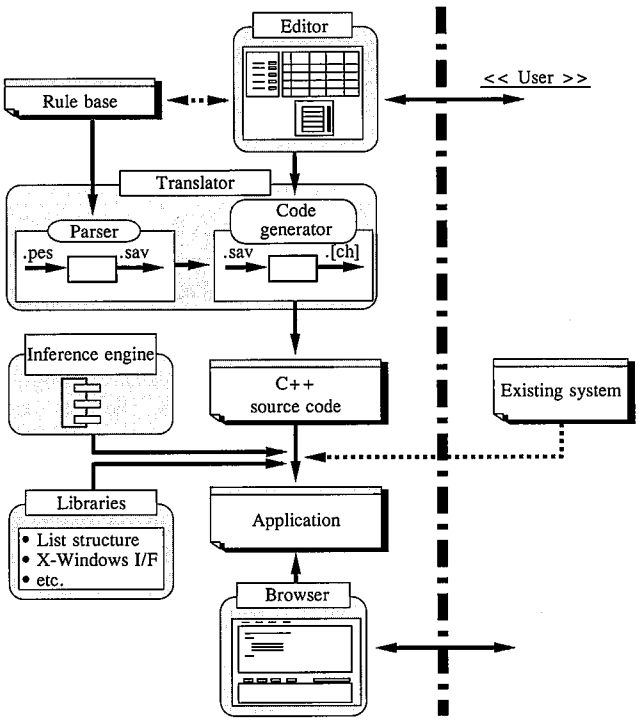


Fig. 6 Development environment of K1

transformed into a C++ language class definition, and the condition part is developed into a discrimination network. After rule preparation is complete, adequate processing (for example, running the make command in UNIX) generates the execution form of an expert system.

2.2.2 Inference engine kernel

The inference engine kernel consists of two main functions. The first function is the library that takes charge of drive mechanisms for such basic functional elements as pattern matching, conflict resolution, and rule firing. This inference engine is generally classified as a forward reasoning type. It is natural that inference is event driven in K1. All working memory elements are treated as objects in the reasoning process. When the working memory is updated or when an event occurs, a rule object is transmitted to execute a rule, producing a new event.

The expert systems constructed in K1 are derived from a single inference engine class and share working memory. These expert systems can also be designed to coordinate with each other while sharing working memory. The other main function of the inference engine kernel is the top-level function that provides interface for development phase interactive operations such as reasoning, tracing, and working memory display.

The user can use the top-level function to trace and control the execution of the expert system on an interactive basis. Of course, all expert system functions can be directly accessed from a user program in detail without using the top-level function.

2.2.3 Editor

K1 also provides an intelligent editor. Dragging and dropping are effective for entering mutually associated pieces of knowledge. For example, when creating a rule that refers to a working memory element in the condition section, this condition section can be automatically generated by dragging and dropping the working memory element on the screen of a rule editor.

When a correction or deletion is required, the editor automatically performs a syntactic check according to the language specifications. When deleting an instance of knowledge referred to from another piece of knowledge, for example, the editor checks the relationship between these pieces of knowledge and displays a warning message if necessary.

The two functions shown here, the checking and instantiating function for generating only syntactically correct rules, and the entry relation consistency checking function for guaranteeing consistency among relations, are very useful in maintaining a large-scale knowledge base. Since the machine on which the expert system is developed is sometimes different from the machine on which the expert system is run, a function for converting the knowledge base between editor and text files is also provided.

2.2.4 Browser

Many commercial shells provide tools for debugging expert systems under development. K1 has the above-mentioned top-level function for controlling the inference process. This is a minimum necessary function. A dedicated browser is available to complete an advanced-function, user-friendly debugging environment. The browser can also be used to trace inferences while visually showing such inference process information as the current status of working memory elements, the state of pattern matching, and candidate rules to be fired.

2.2.5 Libraries

Although not indispensable for the K1 development environment, several useful libraries are supported. The list data structure is the most basic data structure for planning and scheduling expert systems. K1 does not support the list data structure. The list classes described in C++ language are provided as libraries.

The user can import the list classes and utilize them as classes or use them in the knowledge base. Data exchange between the expert system built using K1 and another system is also important. A file I/O class is provided for storing the inference results in a file or reading the inference results from a file. The file I/O class can be used to interrupt or re-execute reasoning at any time while the expert system is being driven.

In recent years, applications with GUI have become standards. Usually, a GUI-based application and an expert system cannot coexist because of their different main loop functions. K1 provides dedicated Xlib plotting classes that do not conflict. Signal handling classes are furnished to emulate the real-time reasoning required by a control expert system. Once the running time and period are set, the signal handling function defined in the knowledge base is called periodically.

3. Plate Finishing Control Support System (PFCSS)

3.1 Description of system

K1 was used to develop the prototype of an automatic material flow control system for use in the finishing process of a plate mill. The finishing process follows the rolling train and includes conditioning, painting, and inspection. Since upstream rolling processing capacity exceeds downstream finishing processing capacity, the finishing line has a buffer called a yard that is used to stock plates. A skilled operator at the mill monitors the

material flow from the operation room and controls the flow of plates in the finishing process.

The following are the four main control tasks:

1) Material flow control (set priority of crane operations and set flow ratios at confluent points)
2) Stock (work-in-process plate pile) control (meet delivery dates)
3) Crane control (operate cranes efficiently)
4) Communication between processes and cranes

This control is difficult to perform for the following two main reasons:

1) Because the situation changes in real time, it is difficult to predict effects at setup.
2) Priority of the evaluation indexes changes with the situation.

3.2 Configuration and functions of system

The configuration of the system is shown in **Fig. 7**. Several techniques were considered at the system design phase. Among them were:

- Operations research (OR) technique
- Physical model (material flow simulation)
- Expert system

The OR technique was excluded because of computational complexity. The physical model suited the purpose because it allowed a clear material flow pattern to emerge, but its complexity and made it extremely difficult for the operator to represent the knowledge peculiar to his or her expert domain.

The expert system is the best method for extracting such domain knowledge, but cannot clarify the material flow pattern involved. The system was built as a hybrid configuration utilizing the merits of both the physical model and expert system methods. The following ideas were devised to make up for the shortcomings of the two methods:

(1) A rough model replicating the basic physical behavior of the production line is used to observe the material flow pattern.
(2) Knowledge not embedded in the model is represented in the expert system.
(3) The prediction interval of the model is shortened so that the expert system can modify the model before an error compounds.

The system consists of two main modules: the material flow model and the modification module. The material flow model simulates material flow according to the given operation status and equipment operation plan. Its output is integrated with the evaluation of other elements (overall line conditions) and shown on the CRT. The operator decides the validity of the solution. If the obtained solution is not satisfactory, the modification module selects another operation strategy and material flow is simulated again by the material flow model.

The modification module adjusts the parameters of the material flow model and changes the operation strategy. When the initial and line conditions are given, the material flow model calculates the production line conditions every 2 seconds. After 5 reiterations or 10 seconds of the model time, the results are transferred to the modification model, which in turn determines the production line conditions based on the output of the material flow model.

To obtain one scheduling candidate, this cycle is iterated for 10 minutes of the model time (about 10 seconds of real time). The user can make as many attempts as he or she wishes to search for a better candidate. **Fig. 8** shows the relationship between the material flow model and the modification module in detail. The interface between the operator and the material flow control system is a graphical user interface (GUI).

The material flow model is implemented in C++ because of the advantage in modeling ability of object-oriented languages. In contrast, the modification module is implemented as an expert system in K1 because it must decide on an appropriate operation strategy by utilizing symbol-based knowledge that allows it to evaluate the output of the material flow model. The modification module makes use of expert knowledge supplied by skilled workers and represents this expert knowledge by many rules. The rules are grouped by the grouping function of K1.

The present knowledge base consists of 41 working memory element classes, 887 rules, 51 rule groups, and 26 contexts.

The system was developed on a UNIX workstation. The material flow model and modification module exchange messages through interprocess communication. Openwindows is used for the GUI.

4. Evaluation

The benefits of the PFCSS and the effectiveness of K1 in the development process were evaluated.

4.1 Benefits of systemalization

The PFCSS was tested on a yard balance optimization problem using actual data from several typical situations. The test

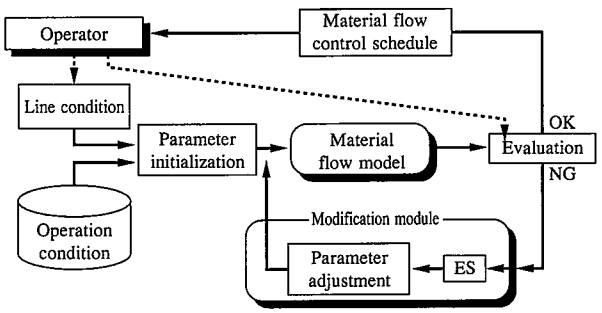


Fig. 7 Configuration of PFCSS

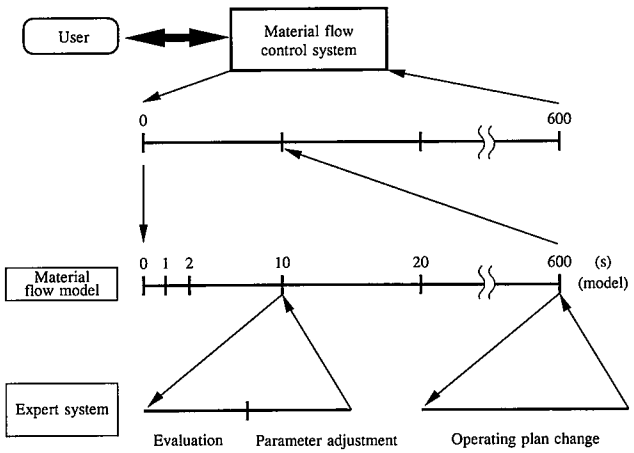


Fig. 8 Relations between material flow model and modification module

addressed the problem of controlling the flow of plates among three tables: E, H, and K (conveyors). The purpose of yard balance optimization is to distribute plates to the tables and to balance the piling of plates at the yard tables

The flow of plates to each table is shown in **Fig. 9**. Clearly, the number of plates piled at each table is properly controlled by the modification module incorporating the expert system. The K1 expert system obtained solutions that agreed with 80% of the judgments made by a skilled operator throughout the test. The 20% miss rate was due to unexpected events, but the system is designed to be manually overridden by the operator when such unexpected events occur.

The results of inference speed evaluation are shown in **Table 1**. As a comparison, the inference speed of a system implemented in a rule-based descriptive language using the RETE algorithm is shown. The inference speed of K1 is about seven times higher than that of the RETE algorithm.

The developed system is positioned as a prototype, but can compute scheduling in the plate finishing process at an adequate inference speed and can accommodate the future addition of rules. The authors firmly believe that if its accuracy is improved, the system will be commercially viable and able to reduce the number of workers required to finish plates and cut plate finishing costs.

4.2 Effectiveness of K1

4.2.1 Characteristics of object-oriented methodology

The system references and eventually stores as working memory elements an enormous volume of external data. That success demonstrated the effectiveness of initially designing working element classes according to an object-oriented methodology. The design phase clarified numerous items of data processing in the working memory elements and showed this data processing as a method of naturally representing data-dependent processing.

Some matching conditions were too complicated to be described by the grammar of K1. In such cases, the user can define a matching condition by a method and can represent a rule by referring to a method in the condition section. The material flow control system has a data base to store the results of reasoning. To avoid contradiction between the external data and working

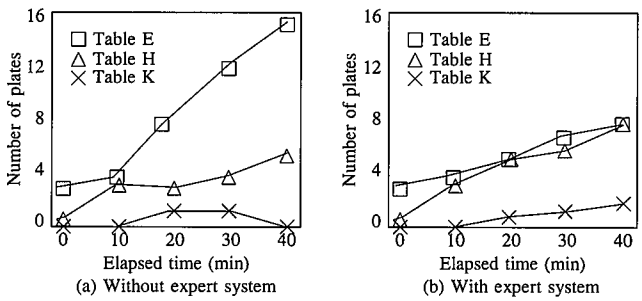


Fig. 9 Results of yard balance optimization

Table 1 Comparison of inference speed

	RETE	K1
Time/Rule(s)	0.0085	0.00125

memory elements, a method-like daemon starts when a working memory element changes.

Let us assume that a daemon is described by the user for an attribute, for example. When the attribute value changes, the daemon is called, automatically updates the external data associated with the attribute value, and ensures consistency with the working memory element. Inheritance, too, is often utilized. Structured knowledge can be naturally represented, the descriptiveness of the knowledge base can be improved, and the maintainability of the knowledge base can be enhanced. If knowledge is defined by inheritance, the change made in a parent is automatically reflected in all of its offspring.

Inheritance appears to detract from the modularity of knowledge, which modularity is an advantage of production rule representation. Experience convinces the authors that the advantages of inheritance outweigh its disadvantages.

4.2.2 Integration with other systems

Two aspects of K1's system integration capability were evaluated. One aspect was direct coding in C++. An important merit of direct coding is the elimination of redundant rules.

For example, similar but different actions are sometimes taken according to the calculated results of action section of production rules. In such cases, direct coding in the C++ switch-case statements can sharply reduce production rule description, clarify the procedural nature of processing, and improve the readability of production rules. If direct coding in C++ is not used, all actions must be programed with rules.

In addition, reducing in the number of rules decreases the number of pattern-matching operations and increases the execution speed of the system. The C++ class import function is also convenient. In the analytical phase of the system, the material flow model's data structure was determined first. The data structure is reflected in the knowledge base of the system by using the import function of the C++ language classes. Practically no special programing was required to reflect in the expert system the results of evaluation by the material flow model.

The other aspect is integration with external modules in library format. Many subroutines (run in C language) developed in past projects were re-utilized in the development phase of the present system.

As described in the preceding section, these libraries can be easily linked by specifying a few directives in the knowledge base. This is a simple function that can be used along with direct coding in C++. Integrating the modules of K1 itself is as effective as integrating the PFCSS with external systems. Since K1 also supports the division of the knowledge base, the knowledge base of the material flow control system was divided into multiple modules, which were developed in parallel by many engineers.

For example, one engineer implemented a knowledge base belonging to one rule group, while another implemented a knowledge base belonging to a different rule group. Before they were integrated the knowledge bases were independently tested to check their operation. Separation of the knowledge bases in different contexts allows knowledge changes to be made without hindering smooth knowledge base development.

5. Conclusions

The planning expert system building tool K1 and the development of the PFCSS (Plate Finishing Control Suport System) using K1 are described. The object-oriented tool K1 improves the

descriptiveness of the knowledge base and uses the development methodology for rule-based programing. K1 is designed for application in the development of large-scale systems that specifically handle synthetic problems like planning and scheduling problems. The validity of the inference algorithm of K1 was verified through several tests. All K1 software is prepared in C++ for faster execution speed and portability.

An editor and a browser are provided as the development environment. The editor and browser should help engineers not familiar with rule-based descriptive languages to develop, build, and maintain effective knowledge bases more quickly. The effectiveness of K1 was verified based on its application to the development of the PFCSS. The PFCSS is now being prepared for practical application. The development of the PFCSS demonstrated the effectiveness of the K1 object-oriented expert system building tool and convinced the authors that K1 is a powerful tool for integrating an expert system with an existing system.

In the next phase, the PFCSS will be improved with the addition of new knowledge, and the results obtained in the development of the PFCSS will be fed back to the next version of K1.

References

1) Barr, A., Feigenbaum, E.A.: The Handbook of Artificial Intelligence. William Kaufmann Publishers Inc. 1982
2) Minami, E., Miyabe, Y., Dairiki, O.: Proceedings of the 3rd International Conference on Industrial & Engineering Application of Artificial Intelligence and Expert Systems. Vol. 2, 1990, p. 684-691
3) Minami, E., Hirata, T.: Proceedings of the World Congress on Expert Systems. Vol. 2, 1991, p. 1311-1318
4) Meyer, B.: Object-oriented Software Construction. ASCII Co. 1991
5) Stroustrup, B.: The C++ Programming Language. 2nd Edition. Addison-Wesley Publishing Co. Inc. 1991
6) Rumbaugh, J.: Object-Oriented Modeling and Design. Prentice Hall Inc. 1991
7) Forgy, C.L.: Artificial Intelligence. 19, p. 17-37, (1982)
8) Miranker, D.P.: TREAT: A New and Efficient Match Algorithm for AI Production Systems. Morgan Kaufmann Publishers Inc. 1990
9) Aoe, J., Yamamoto, Y., Shimada, R.: Proceedings of the First International Conference on Supercomputing Systems. Florida, 1985, p. 491-498