Knowledge-based systems : tools or toys?
a feasibility study of knowledge-based planning

van den Berg, R.J.G.

Award date:
1987

Knowledge-based systems: tools or toys?

A feasibility study of knowledge-based planning

Master's thesis

Rob J.G. van den Berg

Philips Corporate ISA
Corporate CAD Centre
Centre for Quantitative Methods

Eindhoven University of Technology
Department of Mechanical Engineering

WPA-0484

Acknowledgements

Writing a Master's thesis is virtually impossible without the co-operation of others. For this co-operation I am very grateful, therefore I would like to thank all the people who supported me during the last eleven months.

Especially I would like to mention the people who made it possible to carry out the research for this thesis: Prof. Dr. G.R. Joubert and Frank de Bruyn from the CAD Centre, who made it possible for me to work in their group, Leo Fortuin of the Centre for Quantitative Methods, whose claimed ignorance of Artificial Intelligence proved to be most inspiring, as well as Piet Mikkers of the Eindhoven University of Technology, who gave me the opportunity to explore this unusual subject for a student Mechanical Engineering.

Furthermore I owe many thanks to the following people:

André Boogaard, Annemie Damen, Corry van den Berg-Irikx, Frans Peters, Gerard van den Berg, Gilles Ampt, K. Moody, Linda Rickelman, Mynt Zijlstra, Prof. Ir. J.G. Balkestein, Toon Korsten, Wim Boot.

Abstract

Knowledge-based systems, or expert systems as they more commonly are called, have acquired much interest over the last few years. Especially production planning is considered to be a field where knowledge-based systems could be an improvement to current methods. In this report the feasibility of knowledge-based planning systems is discussed. Regarding knowledge-based planning a number of researchers state that the complexity of the planning domain requires a structured knowledge representation. Suggested methods use a frame-based approach or an object-oriented approach.

An important part of the study that led to this report was the development of a prototype planning system for a stylized job shop planning problem. The prototype is based upon a general framework for a knowledge-based planner. The framework states that such a system should employ a hybrid knowledge representation to be able to represent the two different knowledge types: the environmental knowledge represented by frames, the heuristic knowledge, i.e. the planning techniques, represented by rules. Moreover, a knowledge model is defined for the environmental knowledge.

Based on the literature study and the development of the prototype it was concluded that knowledge-based systems provide a feasible enhancement for job shop planning.

Preface

The last few years a new development has emerged in the field of computer science. Following the success of data bases and 4th generation languages, interest is now moving towards a new phenomenon, called expert systems.

Such systems are the result of research done in the field of Artificial Intelligence (AI). They try to solve problems by using expertise, coming from humans. This human knowledge is contained in a knowledge base. Therefore, a more accurate name for expert systems is knowledge-based systems. The most successful expert systems are those developed for diagnosis: systems that diagnose human diseases or identify faults in processes. In these situations the structure of the knowledge is very clear and well documented.

It would now be interesting to examine the performance of knowledge-based systems in situations where the structure of the knowledge is cloudy and not precisely defined. An example of such a domain is production planning: planning in a factory environment. Traditionally a field where problems are solved using Operational Research (OR) methods. Operational Research is the application of scientific methods for dealing with complex problems arising in the management of large systems in industry, business etc.. The approach OR uses consists of the development of a scientific model of such a system. The model incorporates measures of factors as chance and risk, with which to predict and compare the outcomes of alternative decisions and strategies. OR uses scientific methods to advise on management's actions.

It is noted that OR methods generally are limited in their performance. In the field of production planning knowledge-based systems might prove to be an improvement.

Examining the benefits of knowledge-based systems in production planning therefore seems a worthwhile subject. It would give some answers to the application of knowledge-based systems in problem areas where knowledge is relatively unstructured. Knowledge-based technology could also improve the performance of production planning systems and enhance OR methods for advising management.

Being a student at the Eindhoven University of Technology (TUE), Department of Mechanical Engineering, I chose this subject for my Master's thesis. The research for this thesis was done at the

'Nederlandse Philips Bedrijven', in the Advanced Developments group of the Corporate CAD Centre. In this group Prof Dr. G.R. Joubert was my supervisor, assisted by Mr. F. de Bruyn and Prof. Dr. F.J. Peters. As the CAD Centre is inexperienced in the field of production planning, another department of Philips was involved in the project: the 'Centre for Quantitative Methods' (CQM), where I was supervised by Dr. Ir. L. Fortuin. During this project contact with the TUE was maintained by the supervision of Ir. P. Mikkers.

Rob J.G. van den Berg
Eindhoven, September 1987

Table of Contents

PART III: Knowledge-based systems: tool or toy?

1

Introduction

One of the reasons that justify the development of a knowledge-based system for a problem is the possible high pay-off. In most production situations the use of an effective planning system will increase profits substantially. Current planning systems show a limited performance in this area. Hence, improved planning systems can lead to high pay-offs. Production planning has therefore attracted great interest among AI researchers. For Philips planning is an important matter. Hence, Philips too became engaged in knowledge-based planning systems: at the Corporate CAD Centre of Philips a project was instigated. It was carried out as a Master's study in co-operation with the Centre for Quantitative Methods (CQM), also a department of Philips.
In this chapter the following subjects will be discussed:

- The possible advantages of knowledge-based systems in production control.
- The definition of the Master's degree project.
- The goals of the project.
- The approach with which these goals will be pursued.

Possible advantages of knowledge-based systems

Current software packages employed for production planning show a number of weaknesses:

- The plans are sub-optimal. The difference between the generated plan and the optimal case has been observed to be considerable.
- The computational effort required to obtain these plans seems fairly large.
- The planning heuristics and the underlying knowledge are often partially, or completely undocumented and cannot be explored by the user in an interactive manner.
- The user cannot influence the planning heuristics applied by the software.

The last two points are mainly due to current programming techniques. Programs consist of instructions necessary to solve the problem and instructions to control the computer. These two packed together make it difficult to understand the planning specific knowledge (i.e. instructions), and make the program inflexible.
One of the main features of knowledge-based systems is the separation of knowledge and control in modules.
An advantage of this modularity is the incorporation of heuristics (rules of thumb) as used by human planners. In knowledge-based systems they are easy to represent as well as to keep up-to-date. Furthermore it might prove, resulting from the combination of OR and heuristics, that knowledge-based systems will show an increase in performance.

The task

Viewing these advantages it seems worthwhile to instigate a project with the following task:

examine the benefits of knowledge-based systems for use in production planning.

This task was assigned to me as a subject for my Masters thesis. As already mentioned this project was carried out at the Corporate CAD Centre of Philips, in co-operation with the Centre for Quantitative Methods.
As the time scheduled for the project was only nine months and production planning being so vast a field, the assignment was restricted to job shop planning: a frequently used form of short-term planning in manufacturing environments.

The goals

The main goal for my study was to give answers to the following questions:

- What are the specific demands for a knowledge-based system that is to be used in job shop planning?
- What should be the structure of the knowledge base?
- Is it feasible to implement a job shop planning system by means of knowledge-based technology?

I hope to answer these questions by building, i.e. developing and testing, a prototype knowledge-based planner. This prototype will be a knowledge-based planning system for a problem supplied by CQM. Prior to the development of the prototype, literature will be consulted regarding knowledge-based planning systems.
The development of a prototype has a number of subordinate goals:

- Identification and documentation of the knowledge categories involved in the planning process.
- Devising and demonstrating representation mechanisms that will accommodate the previously identified categories of knowledge. This is to help structure the knowledge which is the basis of the planning system.
- The construction of a knowledge-based system that generates production plans and resource schedules for a limited class of problems. Within its limited scope the prototype should show improvements with regard to the four weaknesses mentioned on page 2.

The CQM problem

This section will give a short description of an important subject for this report: the CQM problem.
The CQM problem is concerned with planning in a workshop consisting of two machines (fig. 1). On these two machines a number of products are being manufactured. Each product has to flow through both machines. Every week a number of orders, i.e. requests for a number of products, have to be processed. Planning this workshop consist of arranging these orders so as to achieve an efficient schedule: a sequence of orders that can be processed

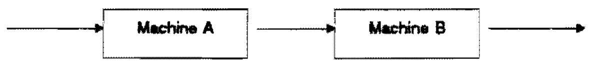


Figure 1: The workshop lay-out.

in the shortest possible time. A detailed description of the CQM problem is presented in Chapter 5.

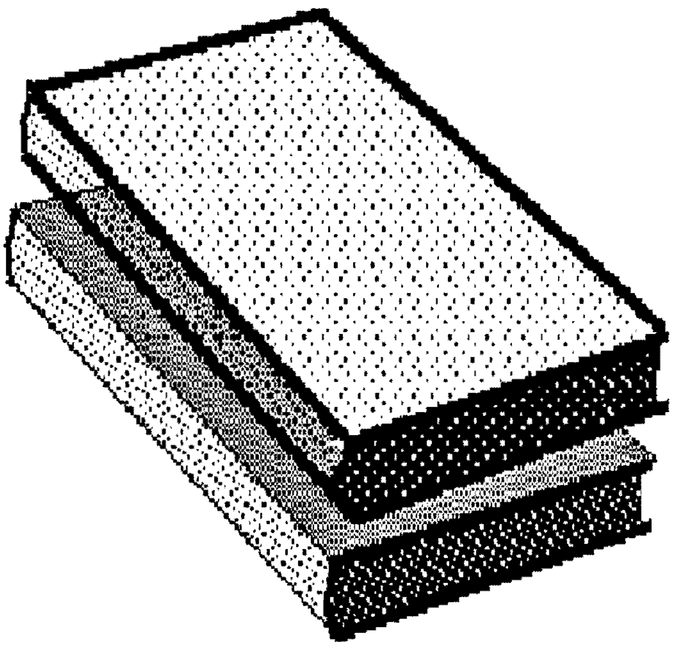
The report

This report gives an account of the research carried out for this study. It is divided into three parts:

- **PART I** : prior to the construction of a prototype a literature study was carried out. In this part a summary is given of the findings of other researchers in the fields of knowledge-based systems and production planning.
- **PART II**: in this part the CQM problem is presented in detail. Also the prototype for that CQM problem will be described.
- **PART III**: In this part as a result of parts I and II some remarks are made regarding the feasibility of knowledge-based systems for production planning. Also subjects for future research will be given.

PART I

Literature Study



2

Production control

To increase productivity in production organizations production control is indispensable. In this chapter some general aspects of production control are discussed. Also a more detailed description of manufacturing environments and control within these environments is presented.

The main source of literature on this subject has been the book 'Production planning and Inventory Control', by D.W. MacLeavey and S.L. Narasimhan [1]. The information gathered from this book will be discussed on pages 7 - 11.

Introduction

There are few production situations with unlimited resources. In most situations it is, therefore, the goal to achieve the maximum result, using the resources as efficiently as possible. This process is known as production control. Note that this does not imply that production control is always optimal: generally it is not.

By production is not meant just the manufacturing of products, but in general, the transformation from an initial state to a goal state with a finished product. Familiar examples are the production of consumer products (cars, furniture, computers), but also for the supply of services, e.g. in national defence or commercial airlines, production control is vital. Production control is of great interest for most organizations that have an urge to produce.

The following subjects are considered to be part of production control:

- Supply of materials: which materials should be ordered when and in what quantity?
- Determining of activity-precedence: In what sequence should the order flow through the work stations?
- Order scheduling: When and on which work stations should an order be processed?
- Dispatching: Which order should be released next?
- Status control: Is production on the level we want?

The questions in these subjects are taken from a manufacturing environment. Examples from other domains are of course also conceivable [2].

Production control, in general, deals with the planning, execution and control of transformation processes. The central issue in production control is trade-off, to make compromises between conflicting entities. For instance, a hairdresser is faced with the trade-off between the quality of his work versus the number of clients he has a day. In a factory there is, for instance, a trade-off between the service degree and a low inventory.

According to MacLeavey et al. [1] every production organization has seven conflicting measures of performance.

- Volume: how much is produced in a time-period, how many customers are served in a time-period?
- Variety: how many different types of products(services) are produced by the company?
- Quality: is the quality of the product acceptable?
- Timeliness: can a product be delivered on time?
- Place: can a product be delivered in the right place?
- Satisfaction: are participants in the transformation process satisfied or fulfilled in their work?
- Cost: have the products, or the services, been created at minimum or acceptable cost?

The first six of these measures can be expressed in cost. To use only cost as a measure is highly impractical for it is not possible to quantify the cost of some of these measures, e.g. customer satisfaction.

Production environment

The nature of the CQM problem means that this chapter is mainly focused on manufacturing environments. A classification and a characterization of manufacturing systems is presented in this section.

Each manufacturing organization has its own character. Production of large quantities of one product is different from producing a wide variety of product in small quantities. A suggestion for a classification of these organizations is given in table 1:

	Low sales volume	*High sales volume*
Uncertain product specifications	*Job shop or flow shop production to order(e.g. moulded rubberparts).*	*Hybrid shop production to stock and assembly to order (e.g. machine tools)*
Certain product specifications	*Batch or intermittent production to stock (e.g. textiles)*	*Repetitive or mass production (e.g. automobiles)*

Table 1: Manufacturing Classification Scheme (from [1]).

As stated in the Introduction, we our consideration of production control somewhat further: the flow shop and the job shop are in this report the environments that are of interest.

Job shop

In a job shop the work flows in batches (jobs) through a set of facilities. These jobs follow different routings, each batch having its own flow pattern.

The facilities generally consist of general-purpose machines able to use a large variety of tools. Hence, different types of jobs can be performed at the same facility. A result of the large variety is that job shops require skilled workers: they have to adapt the machine for every different job. Moreover, the large variety in job routing and machines results in a large variation in:

- sequences of jobs at different machines;
- processing time;
- number of operations per jobs.

Also jobs compete with each other for resources. Competition for resources together with the variation mentioned above, causes job shops to have a large lead time, and they thus tend to have a large work-in-process level. Large lead times and work-in-process make job shops not a very efficient production system.

Planning a job shop is universally regarded as the most complex and difficult industrial planning problem. The complexity is the result of the large variation in, for example, job sequence, processing requirements [3]. Examples of job shop environments are: hospitals, service stations and diffusion plants for chips (the electronic type).

Flow shop

Unlike a job shop, in which each job has its own routing, a flow shop is a production system in which the flow through the facilities is the same for every job. Jobs do not flow in batches through the shop, but as single products. These shops are also known as assembly lines.

Flow shops have a job routing in which operations on products are performed repeatedly in every work station. The machines in these work stations can perform only one type of operation. They require less-skilled workers than in a job shop, because the task for which the work station is used is embedded in the design.

Due to the uniform flow through the shop, flow shops are more efficient than job shops. However a number of problems have been noticed in flow shops [1]:

- Inventory level: in a flow shop products are generally made to stock. Forecasting is an important task.However, due to the erratic demand, a difficult one. Errors in forecasting can lead to high inventories of raw material and finished goods (Note: while keeping a low work-in-process level).
- Machines: in flow shops special purpose machines tend to have a special purpose design. The initial investments are therefore high. This makes them only suited when dealing with large production volumes.
- Repetitive nature: As a consequence of the high specialization of flow shops, the work is monotonous and affects worker morale.

Intermittent flow shop

An intermittent flow shop is a mixture of a job shop and a flow shop. It processes batches of products, but has the layout of a flow shop. Intermittent flow shops are useful when high production levels are needed on a periodic basis [1].

Job shop planning

For planning in a manufacturing environment there are a large number of methods available. One of them is most frequently used: job shop planning. The reason for this is twofold. First, job shops are the most widely used production organization. Second, it is possible to abstract above mentioned environments to one: a general job shop. An intermittent flow shop is a job shop with no alternative routings, while the assembly line type of flow shop (page 8) can be viewed as having batches equal to one product. Although there are other methods for planning flow shops, it is possible to approximate these methods by means of job shop planning.

The control of a job shop, the job shop production activity planning (fig. 2), consists of three phases:

- planning: determining an efficient workload by means of arranging jobs in a certain order;
- execution: dispatching of orders to the shop floor;
- monitoring: reporting of the progress of the jobs, monitoring of the workshop.

In this report our interest is focused on the planning phase. The planning phase can be divided into three stages:

- scheduling;
- loading;
- sequencing.

These will be discussed in detail in the following subsections.

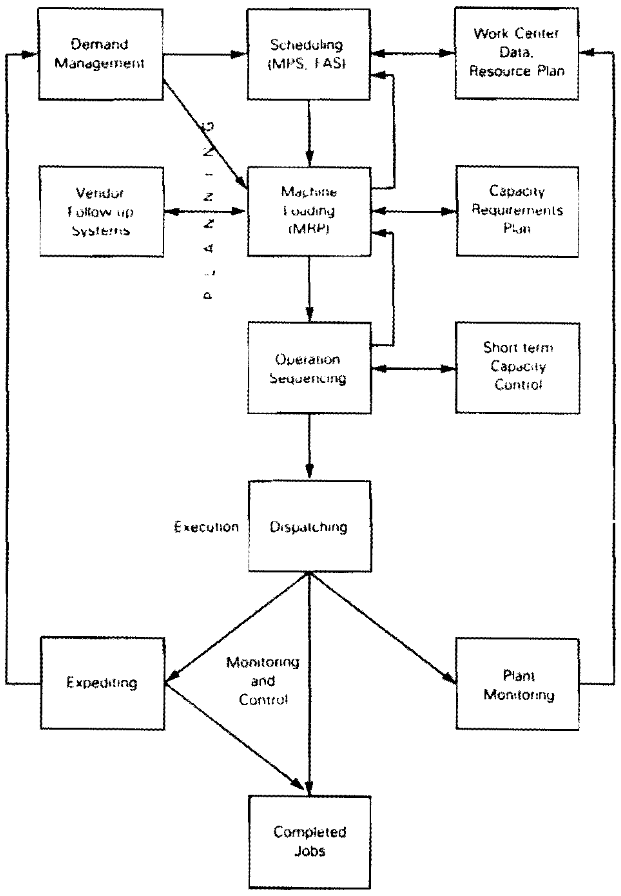


Figure 2: Job shop production activity system(from [1]).

Scheduling

In most situations there is a trade-off between availability of resources and production objectives, e.g. due dates, cost. The goal of scheduling is to optimize this trade-off. Scheduling is the selection of orders to release that are most suitable for production. The suitability depends on the trade-off mentioned above.

Scheduling means assigning dates (i.e. start times) to specific jobs or activities (part of jobs). A crucial problem is that job shops show stochastic behaviour. Factors like: uncertainties in the process, machine breakdown, absenteeism, etc. cause that planned start and finish times may not be met. Stochastic behaviour makes scheduling a complicated task: it may be possible that the best schedule cannot be performed. On the other hand it is important to have reliable schedules: customers expect their products on time, management has set goals for the production based on schedules. A reliable schedule is very important because it improves the efficiency of the plant and thus reduces cost.

The techniques used for scheduling depend on their objective. Scheduling in order to obtain a minimal lead time can have different requirements than scheduling for minimal cost.

Loading

When orders are scheduled they have to be assigned to resources. The process of assigning operations to resources is called loading. There are two methods for loading: infinite and finite loading.

Infinite loading assigns work to resources without regard to the capacity. Finite loading compares the required resources with the available capacity. The last method has the advantage that a more realistic workload can be determined. It is, however, very hard to accomplish.

Sequencing

Sequencing is: specifying the order in which jobs have to be processed. A number of sequencing methods are known. Often applied methods use static or dynamic priority rules, that select orders according to one of their properties, e.g. processing time, time of arrival. Other methods use OR techniques like mathematical programming, heuristics or simulation.

Planning revisited

Having presented the phases into which planning a job shop can be divided, a few problems of job shop planning will be discussed in this subsection.

The first problem deals with the relation between the three planning phases: in the above subsections it is suggested that the phases are independent of each other. In reality they are strongly related to one another. While scheduling orders, one has to take into account the effect the schedule has on the loading of the work stations on the floor.

Furthermore, a plan is not isolated: usually planning is a hierarchic process, i.e. plans having a high level of aggregation are input for planning on a lower level of aggregation. For instance: plans concerning different plants are input to a planning for one plant.

Together with the stochastic behaviour in a job shop, this makes planning a complex task.

This complexity poses another problem. Planning a job shop is a search in a large state-space. Furthermore, it is a NP-complete problem: there is no known algorithm with polynomial time-behaviour to solve it, only ones with exponential time-behaviour have been found.

The exponential behaviour results in an impossibility to find an optimal solution. Given 85 orders passing through 10 operations without alternatives, with a single substitutable machine for each operation and no machine idle time, there are 10^{880} possible schedules [4]. Generally in an extensive situation the only methods for analysis are simulation and the analysis of highly abstracted models. Results from abstracted models have a limited validity. Simulation, however, is very time consuming. Moreover, results are only applicable to that one situation [5].

Conclusion

As can be seen in this chapter, the problem of production control specific to the job shop type of factory is very complex. There is a high interaction between the different planning phases, competition between orders, as well as stochastic behaviour in the shop. In general, optimal solutions cannot be found as a result of the NP-completeness of job shop planning. Planning a job shop strongly relies on heuristic methods based on experience. The aid of computer techniques is restricted to analysis of abstracted models or simulation.

Another problem is the quality of a schedule. What performance measures can be used to judge a schedule? Cost seem a good measure, but not all measures can be expressed in cost. How much does it cost to get better worker morale, and how much is the benefit? The use of costs as a single measure is here impractical.

The use of multiple measures introduces another problem: how does the satisfaction of one measure affect the satisfaction of another? The use of multiple measures in this case seems the best solution. It is better to have problems in dealing with different performance measures, than have a inadequate performance measure.

3

Knowledge-based systems

Knowledge-based systems, or expert systems as they are more frequently called, are the result of research done in the seventies in the field of Artificial Intelligence (AI). In this chapter a description of knowledge-based systems is given. Also their relation to AI, their history, and examples of existing systems are discussed. 'A Guide to Expert Systems' by D.Waterman [6] was the main source of information for this chapter. On control strategy, P.H. Winston's book 'Artificial Intelligence' [7] was an important source.

Artificial intelligence

In the early days of electronic computing the rapid development of computers suggested that it could be possible for a computer to simulate human behaviour. In 1956, at a conference at Dartmouth College, USA, the term Artificial Intelligence (AI) was launched. The scientist participating in that conference forecast that within 25 years computers would be able to run intelligent programs. Unfortunately[1], after thirty years of research this goal still seems a distant one [5].

1 *Some would say "luckily".*

In those early days the aim was to devise general-purpose problem solvers that could deal with a range of problems. As these showed no progress, the research became more practical, with the aim to create programs which show intelligent behaviour in narrow fields.

The following list contains the fields in which AI research is currently being done [8]:

- **Problem solving:** techniques used for solving puzzles or playing games evolved into fundamental AI techniques of search and problem reduction.
- **Logical reasoning:** work concerned with logical deduction, most notably theorem provers.
- **Natural language processing:** Because natural language is the manner to display intelligent behaviour, language understanding and synthesis have attracted many AI researchers.
- **Programming:** The search for systems that can write computer programs given the its specification.
- **Learning:** Machine learning is the most interesting subject for AI, however it is generally not present in AI systems.
- **Expertise:** Also known as knowledge engineering, this area is involved in the development of knowledge-based systems. These systems are problem solvers using knowledge of a domain to achieve a solution. This field has aroused most interest outside AI.
- **Robotics and pattern analysis:** creating programs that manipulate robot devices. Understanding visual images is a very important issue in the control of robots.
- **Systems and languages:** The development of software tools is an important part of AI. With the aid of these tools intelligent systems may be more easily built.

Knowledge-based systems

The aim of early AI researchers was to simulate thought processes in man by applying general problem solving methods to broad classes of problems. They tried to find a general problem solver applicable in a large domain of problems.

These general problem solvers, however, are difficult to build. Moreover, they presented no significant breakthrough in the simulation of human intelligence. Furthermore, it appeared that the more general purpose a program was, the less purpose it had for individual problems.

The failure of general problem solvers led to a changing approach in the late seventies. Instead of using a general problem solver, AI scientists realized that adding domain knowledge to that problem solver could help find a solution to a problem [6].

From the more general methods there was a shift to special purpose programs: experts in their domain. These systems are known as knowledge-based systems or expert systems.

The two most famous examples of existing knowledge-based systems are:

- XCON: a knowledge-based system for configuring VAX computers used by DEC. Reports say that it saved vendors around $200,000 a month. Developed at Carnegie Mellon University [9].
- MYCIN: A tool for physicians that selects the appropriate anti-microbial therapy for patients with bacterial, meningitis and cystitis infection. Developed at Stanford University [10].

A more complete catalogue of existing knowledge-based systems can be found in [6].

Why knowledge-based systems ?

One might think after this introduction: So what! Why should human experts be replaced? Table 2 gives some possible advantages of artificial expertise, i.e. knowledge-based systems, over human expertise.

The Good News

Human Expertise	*Artificial Expertise*
Perishable	Permanent
Difficult to transfer	Easy to transfer
Difficult to document	Easy to document
Unpredictable	Consistent
Expensive	Affordable

Table 2: Comparing human and artificial expertise: the good news (from [6]).

- Human experts have a restricted period in which they work. Secondly, they have to practise and rehearse constantly to avoid forgetting knowledge. Artificial experts can preserve their expertise much more easily.
- Transferring knowledge is a tiresome learning process for humans, involving years of hard work. Transferring knowledge of a program can be done by simply copying a program or data file.
- As a consequence of the formal notation of artificial knowledge, documenting this knowledge is simple.
- A human expert may take different decisions in comparable situations because of emotional factors. Knowledge-based systems have no emotions. Lack of emotions makes them more consistent in their decisions than humans.
- The price of human experts is high: experts are scarce. Knowledge-based systems are relatively cheap and easy to duplicate.

But, as may be suspected, knowledge-based systems also have their weaknesses (Table 3).

The Bad News

Human Expertise	*Artificial Expertise*
Creative	Uninspired
Adaptive	Needs to be told
Sensory experience	Symbolic input
Broad focus	Narrow focus
Commonsense knowledge	Technical knowledge

Table 3: Comparing human and artificial expertise: the bad news (from [6]).

- Humans are much more creative than the smartest programs. Furthermore, they have an ability to react to unexpected events.
- Humans can adapt to changing circumstances through learning. Learning in knowledge-based systems is still an almost unexplored field.
- Humans can sense things through their sensory perception. Computers can only handle symbols that represent these things.

- Restrictions on the force of knowledge-based systems restrict their focus of the problem. Humans look at problems from a wide angle; computers do not have that capability. Some say that it is a result of the limited computer performance, others say that computers never will be able to achieve this. This problem lies somewhat out of the scope of this report. I refer in this case to Hofstadter, who has some interesting remarks on this subject [11].
- In every act we do, we use common sense knowledge. The enormous amount of this acquired knowledge is, by present standards, impossible to implement in knowledge-based systems.

In view of these aspects it is possible to draw the following conclusion: in situations where human expertise is scarce or where expertise may disappear (e.g. retirement of experts) knowledge-based systems can be a powerful supplement or substitution of human expertise. However, knowledge-based systems have a limited capacity for solving problems requiring knowledge that lacks a clear structure or contains much common sense.

This limitation also has an effect on the effort needed to create a knowledge-based system. Given a problem domain where the knowledge is structured, the acquisition of knowledge and the implementation in the knowledge-based system is a relatively easy task. If the knowledge is vast and seems unstructured, the construction of a knowledge-based system is a complex and time-consuming task.

Basic concepts

Now that we have seen what the possible advantages of knowledge-based systems are, we will describe the essential concepts and parts of a knowledge-based system. In general, three constituent parts can be distinguished: (fig. 3):

- **Knowledge base**: contains the knowledge of the domain, possibly gathered by a knowledge acquisition module.
- **Inference engine:** supplies the methods for manipulating the knowledge, in order to arrive at conclusions.
- **User-interface:** a tool aiding the interaction between the user and the system, e.g. a sentence parsing program to understand natural language queries from the user.

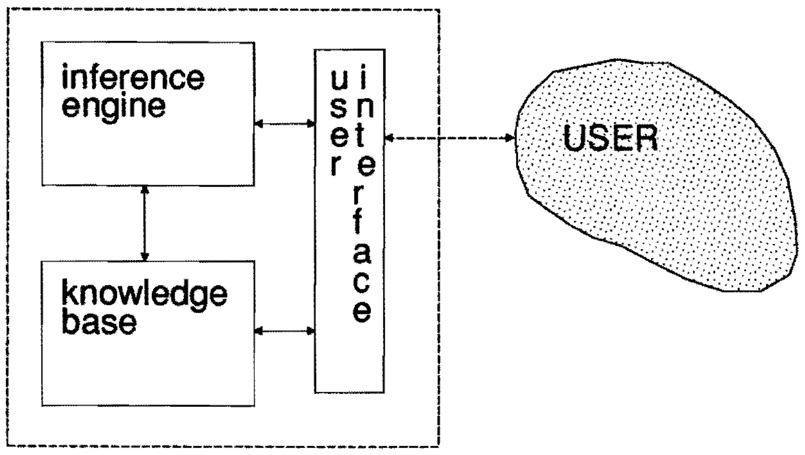


Figure 3: Knowledge-based system

In the following subsection two of the basic elements, the knowledge base and the inference engine, will be presented in detail.

Knowledge base

The knowledge base is the heart of a knowledge-based system: it contains the knowledge with which problems can be solved. The key problem for the design of a knowledge base is *representation*, how should the knowledge be encoded in it, how should the encoded knowledge be structured?
Of the many representation methods used, those most widely used will be discussed here: the production systems and the object-based systems.

Production systems[1] are based on the idea that knowledge can be represented in the form: IF condition THEN action. These are called production rules: each rule produces an action, like in the following rule [7]:

IF guest is sophisticated

THEN wine is indicated

1 *Note that 'production' is used here in an entirely different sense from that in Chapter 2.*

When in the current state the condition in the IF-part is satisfied, the action specified in the rule will be executed. This action may be: to draw a conclusion, to direct the program or to react to the outside world. The actions can have a different effect. Rules can also be used to guide the reasoning process i.e. rules that select which rule should be fired next [6].

Rules are a natural way to describe experts' knowledge. Human experts often depict their expertise as rules. An example of such rules can be found in legislation or regulations. Also processes driven by complex and rapidly changing data can easily be described by rules. These rules can specify how the program should react to the changing data without requiring prescience about the flow of control. [6]

One of the greatest advantages of production systems is their modularity. Rules and facts can be added (or deleted) independent of the rest of the knowledge base. The modularity of rules is also a main problem: the increasing knowledge (i.e. rules) makes the knowledge base difficult to survey and to support; moreover, the reasoning process also becomes unclear and slower, as a result of the time-consuming search through the extensive knowledge base. To limit the search, more control rules to guide the reasoning process have to be added [6].

Under the collective term **object-based systems** two types of knowledge representation can be described: semantic nets and frames. Semantic nets consist of nodes connected by relations and organized into a hierarchy. Each node represents an object/situation, that can be described by attributes and values associated with that node. Nodes at low positions in the hierarchy automatically inherit properties of higher level nodes. The higher the node is situated in the network, the more general the knowledge it represents[6]. Viewing the network in figure 4 the node 'ocean liner' inherits the property 'floats' from the node 'ship'.

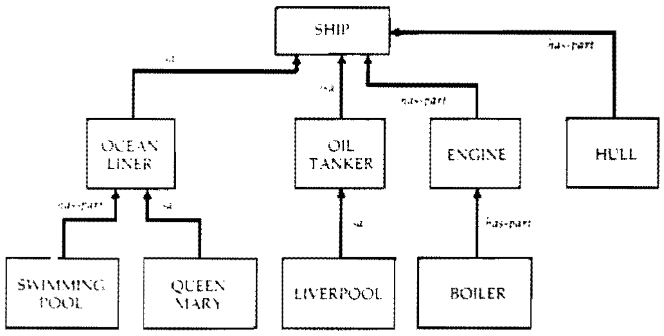


Figure 4: A simple semantic net for the concept of a ship (from [6]).

More than in a production system these nets provide a structured representation. Instead of a mass of rules, the nodes of the net have a clear relation to one another. However, contrary to rule-based systems, the problem solving method (control strategy) is not totally dictated by the inference engine. In a semantic net the control strategy is partially determined by the objects and the relation between the objects as well as the inheritance in the network.

A frame is a node in a net, that is related with other frames (nodes) through an 'IS-A' relation, e.g. as in figure 4 'oil-tanker' IS-A 'ship'. It represents information using a record-like structure that contains values of common attributes, called slots, of the object/situation that is represented by that frame. The information contained in the slots of the frames can have different facets: it can be a value inherited from more general frames, it can be a value inserted during creation of that frame or it can contain procedures that are executed (called demons) when information is needed or changed. Among the object-based systems the frame-based systems are the most commonly used.

Although object-based and rule-based systems may appear to exclude one another, current research is aimed at combining these representations. Coupling the modularity of rules with the structured knowledge in object-based systems. These hybrid systems make it possible to use the representation tationthat best fits the problem; e.g. represent that part of the problem that is best represented in rules, while the other part is represented in frames.

In table 4 an overview of knowledge representations and their specific qualities is given.

REPRESENTATION SCHEME	FORM	ADVANTAGES	DISADVANTAGES	SYSTEMS BASED ON SCHEME
PRODUCTION SYSTEMS	Production Rules	-Highly modular -Easy to understand -Easy to add to or modify	-Control flow hard to follow -Hierarchies hard to capture -Inefficient for large systems	MYCIN XCON (R1) DRILLING ADVISOR
SEMANTIC NETWORKS	Nodes representing objects or descriptions joined by links	-Flexible -Easy to capture hierarchy -Easy to trace associations	-Exception handling is difficult -Meaning attached to nodes may be ambiguous	PROSPECTOR INTERNIST SCHOLAR
FRAMES	Set of slots and associated values represent an object	-Easy to include default information -Can detect missing values -Can see if frame matches available data	-Much of the work is still at experimental stage	PIP CENTAUR

Table 4: Schemes for representing knowledge (from [35]).

Inference engine

The inference engine supplies the control strategy to guide the process of knowledge utilization: which part of the knowledge base should be consulted next. For instance, a popular form of problem solving is search in a state-space, an abstract space containing all possible states that can occur within a situation. In knowledge-based systems the aim is to decrease the search space by applying domain knowledge to that search. The task of searching for and selecting of knowledge is done by the inference engine.

It is important to avoid combinatorial explosion of the search process and to limit the required computer resources by avoiding blind search. In other words: to use an efficient search-method (e.g. by using early pruning, pruning of partial solutions).

Choosing a control strategy is related to the representation method: inferring in a rule-based system is done by rules triggering rules, whereas in frame-based systems inferring is done by examining slots of frames related to one another. Winston [7] distinguishes three different control strategies:

- **Action-centred control** is used when the knowledge about what action should be performed is embedded in previous actions. Action-centred control is often used in rule based systems.
- **Object-centred control**: knowledge specifying how to deal with objects in a class, is given by a class description. Object centred control is used in frame-based systems where through inheritance information slots are altered.
- In **request-centred control** actions/objects know their own purpose and can respond to requests, either by sending messages directly to one another or by using a so-called blackboard: actions read and write messages on this blackboard.

A remark on blackboard systems: their architecture makes them very suitable for problems in which different types of knowledge are needed or where reasoning on different levels is important. Furthermore, they are very suitable for problems requiring different knowledge bases that use different control strategies [6].

Another aspect of the inference engine is the problem of how a goal will be achieved. The two most frequently used methods are called backward chaining and forward chaining.

Backward chaining tries to prove a hypothesis by searching the knowledge base for a rule whose conclusion makes the hypothesis true. The condition of the selected rule will then function as a hypothesis for another rule. The search continues until a rule is found with a condition that is true in the initial state. Backward chaining is also known as a goal-driven strategy.

Forward chaining starts with an initial state and tries to find a rule that applies to this state. The action causes a state-transition. That new state might be the condition for a new rule and so on, until, hopefully, the goal state is reached. Forward chaining is also called a data-driven strategy.

These strategies can, of course, also be used with frame-based systems: In this case the information is embedded in the slots of the frames.

The choice between backward chaining and forward chaining depends on the domain. In a situation where a large number of rules is applicable and where the goal is to achieve a conclusion, forward chaining could well prove inefficient for it evaluates possible paths that are fruitless. Backward chaining only fires the necessary rules to make a hypothesis true.

In a situation where a system has to react to changing circumstances (process control) a forward chain seems useful.

Uncertainty

Human experts take a decision when they are almost certain of their conclusion. In general, a decision is reached by eliminating possibilities with a larger uncertainty. Hence, the final conclusion one has drawn is also uncertain.

For this reason knowledge-based systems also should be able to deal with uncertainty e.g. uncertain facts, uncertain conditions. Among the methods used for uncertainty are:

- Bayesian theory [12].
- The use of "certainty factors", a derivation of bayesian theory [13].
- Fuzzy logic [14].

Tasks for knowledge-based systems

In this section a list is given of the categories of knowledge-based systems. Also the areas in which they are used are presented. Waterman [6] gives a list of possible categories of, in his terminology, expert systems (table 5).

Category	*Problem Addressed*
Interpretation	Inferring situation descriptions from sensor data
Prediction	Inferring likely consequences of given situations
Diagnosis	Inferring system malfunctions from observables
Design	Configuring objects under constraints
Planning	Designing actions
Monitoring	Comparing observations to expected outcomes
Debugging	Prescribing remedies for malfunctions
Repair	Executing plans to administer prescribed remedies
Instruction	Diagnosing, debugging, and repairing student behavior
Control	Governing overall system behavior.

Table 5: Generic categories of expert system applications (from [6]).

Another, more popular, categorization is by their application area. Waterman also gives the following list of areas. The lion's share of knowledge-based systems is used for diagnosing faults in industrial processes, human disease, etc. (table 6).

Agriculture	Manufacturing
Chemistry	Mathematics
Computer Systems	Medicine
Electronics	Meteorology
Engineering	Military Science
Geology	Physics
Information Management	Process Control
Law	Space Technology

Table 6: Application areas for knowledge-based systems (from [6])

Limitations

Present day knowledge-based systems do have a number of limitations. The following are considered to be the most important:

- Representing spatial or temporal knowledge can require huge amounts of memory if the state of things have to be recorded at various points in time, or if spatial relations between objects have to be represented.
- Common sense reasoning: because of the large quantity of this type of knowledge and the difficulty to acquire this knowledge from others , implementation has not yet been possible.
- Knowledge-based systems also have difficulty dealing with erroneous or inconsistent knowledge, because knowledge-based systems only contain abstracted knowledge of the domain. They do not have the ability to look outside their domain to recognize this, hence, do not have the possibility to recognize erroneous knowledge or reason about inconsistencies.
- Human knowledge, especially expert knowledge, is often implicit: a human does not really know how she/he reached a conclusion. It is therefore very hard to extract knowledge that is implicit from an expert. This makes knowledge acquisition a time-consuming task.

These limitations mean that in many real world problems, knowledge-based systems still act as 'idiots savants'. They are unable to learn, to reason on different levels, to look at a problem from a different perspective, to know when to break their own rules and to understand the reasoning behind their own rules. It is clear that most human tasks are still not suitable for a knowledge-based system. It is expected that the improvement of hardware and software will give knowledge-based systems more possibilities.

Conclusion

Now that we have seen what knowledge-based systems are, let us see what the differences are between these and conventional programs (i.e. programs which process data in one way or the other). Teknowledge[1] characterizes these as shown in table 10.

Data Processing	*Knowledge Engineering*
Representations and use of data	Representation and use of knowledge
Algorithmic	Heuristic
Repetitive process	Inferential process
Effective manipulation of large data bases	Effective manipulation of large knowledge bases

Table 7: Comparison of data processing and knowledge engineering (from [6]).

The largest difference however is embedded in the philosophy with which these systems are designed. Usually models of problems are built using a systems design method in which one is forced to forget a priori experience, and to build a model from scratch. On this simplified model standard solutions can be applied. The systems design method is used in the field of operational research [5]. Model building in AI tries as much as possible to use the problem-solving methods used by humans. Reduction of a problem in order to apply standard solution methods is not the goal in AI model-building

The implementation of a knowledge-based system is also different. Traditional software systems are implemented as complete systems: when the model describing the problem is created it is implemented as a whole. Knowledge-based systems, due to their model-building philosophy are designed incrementally: knowledge is acquired gradually.

1 *A prominent firm involved in AI research.*

4

Job shop planning using knowledge-based technology

In this chapter an attempt is made to illustrate the possible advantages knowledge-based systems can have for job shop planning. Also an overview will be given of current research in this field. The findings discussed in this chapter are mainly based on the publications by M. Fox et al. ([4], [15], [16]) of Carnegie-Mellon University, USA and T. Grant and P. Elleby from Brunel University, UK ([5], [17]).

As mentioned in Chapter 1 current software packages show a number of weaknesses. Besides the argument formulated in this chapter that knowledge-based systems have advantages because of their modular design, the following aspects give further indication of the benefits of knowledge-based systems for job shop planning:

- Scheduling orders in a job shop is a **complex** task: when in a job shop there are dynamic arrivals or there is stochastic information, the problem can be classified as NP-complete. [18]. Given that AI research is involved in this type of problems, it seems interesting to use the findings of this research.
- As a result of the NP-completeness of planning problems, they are often solved using **heuristic techniques.** As we have seen in Chapter 3, heuristic techniques are the basis for problem solving in AI.
- Heuristics developed by OR are often only applicable in abstracted situations. Hence, OR tries to abstract the 'real

world' into models in order to be able to use these heuristics. AI tries to model the 'real world' to a larger extent, employing heuristic techniques that are being used in that 'world' for problem solving.
- Through the modularity of the knowledge base it is relatively easy to make heuristics applicable in different environments.

In short: In planning problems knowledge-based systems can use an efficient search strategy for an NP-complete problem. The efficiency is achieved by applying domain knowledge to that search. The modularity of the knowledge base make that they are easily adaptable to other situations.

Specific AI problems

The use of knowledge-based systems for job shop planning presents a number of specific problems [19]:

- Representing and reasoning about time, causality and objectives.
- Uncertainty in the execution of plans and dealing with the real world.
- Sensation and perception of the real world and interpretation of these: planning in a job shop depends on the current state of the job shop.
- Multiple agents who may cooperate or interfere: each order (agent) has its own goal. The satisfaction of a goal has effects on other orders.
- Physical or other constraints on suitable situations: planning is often considered to be the satisfaction of constraints.

Some of these problems were already mentioned in Chapter 4 as being limitations of current knowledge-based systems. These problems make the planning domain also very interesting for AI research.

Existing systems

Recently the interest in AI techniques for production planning has assumed large proportions. The research, resulting from this interest, evolved in the development of a number of knowledge-based planners. From the, already long, list of these planners a selection will be given in this section. The selection is made a bit haphazard, for those systems are selected which are mentioned in a number of articles.

Much work in knowledge-based planning is done at the Intelligent Systems Lab of the Institute for Robotics at Carnegie Mellon University (CMU). At this university ISIS-II [15], and Callisto [16] were developed. ISIS-II is a job shop scheduler of gas turbine parts production. The approach used in ISIS is to generate schedules by heuristic search using evaluation functions. These evaluation functions are based on constraints associated with, for example, cost, process availability. Limitations of ISIS are that the domain is deterministic (i.e. no uncertainty in the shop) and ISIS has no ability for process planning.

The goal for the Callisto project is to develop a project planning system, in which an integrated theory of activity modelling is the core of the knowledge contained in the knowledge base.

NONLIN is a development of the Department of AI, Edinburgh University. It was designed for the construction of project networks. These were represented as a partially ordered network of action.

Another scheduling system in a completely different field is that developed by Slagle et al. at the Naval Centre for applied AI. It is designed for scheduling military resources (e.g. weapons) in battlefields [20].

Knowledge representation

How

In planning situations there are many categories of knowledge: goals, resources, heuristics, etc. The environmental knowledge consists of elements which are strongly related. The nature of the environmental knowledge suggests the use of an object-based system. The use of a object-based system, i.e. a frame-based system, is supported by, among others, Fox et al., [4], Kunz, [21] and Goldstein et al. [22].

A variant to the frame-based method is suggested by Grant [23]. He uses an object-oriented approach for the production planning domain. The objects, in his context, are entities that have properties describing their state, and 'actions' describing how these entities react to certain conditions. Objects influence each other by passing messages [6]. Grant states that the essence of object-oriented planning is to model resources as objects.

These two methods, however different in detail, have a strong resemblance for they both are based on objects and relations between objects.

An enhancement to a object/frame-based system could well be the use of a mixed representation: frames and rules. Rules provide a natural way for expressing the heuristic knowledge often used in planning. In such a system entities that describe the environment could be represented in frames, whereas heuristic knowledge about scheduling can be represented as rules. Jackson et al. give an example of a project planning system using such a hybrid system [24].

For representation of the knowledge it is further suggested to use a subdivision into three levels, in order to separate the different types of knowledge from each other. A type represents a level of abstraction of the knowledge. The use of three levels was first introduced by Brachman [25].

The use of these levels makes the knowledge base more structured and presents a logical structure to the user. The complexity of the job shop planning domain as can be derived from Chapter 2 supports the structured representation. Furthermore, it is stated that different knowledge types should be represented at different levels. The methods as presented here could be of use for that.

What

An example of a structured knowledge base is the frame-based system used by Fox [4]. He uses a knowledge base with a stratification in three levels:

- A level containing general knowledge of how to deal with frames: the epistemological level.
- General knowledge about planning and decision making is represented at the conceptual level.
- A level representing domain specific knowledge: the domain level.

Using such a structured, hierarchic representation, it is clear that the knowledge of interest to us is to be represented at the conceptual level and the domain level. At the conceptual level more general aspects will be represented. The domain level is filled with the domain specific knowledge.

The following concepts are likely to be represented at the **conceptual level**:

- state;
- act;

- time;
- causality; acts/states have temporal relations but are also causally related, e.g. grinding after milling if milling is completed;
- possession of resources;
- composition, aggregation and abstraction of acts and states.

As for the **domain level,** Fox [4] and Sathi [16] consider the following types of knowledge necessary for planning/scheduling in a production environment.

- Activities,
  - required activities;
  - duration of each activity, its range of duration including a probability density function;
  - activity precedence;
  - descriptions of how activities are performed;
  - authorizations regarding persons performing activities
  - conditions under which activities may be performed;
  - logical connections between activities.
- Resources:
  - required resources;
  - when are these resources required;
  - transformations applied to resources;
  - substitutability of resources;
- Representation of changes in the product;
- Aggregation and abstraction of activities;
- Interaction with the user.

The concept considered vital to the knowledge-based planner, constraints, will be represented in both levels. In both levels there are concepts to be constrained, e.g. time at the conceptual level, resources at the domain level.

Constraints

Fox [15] and Grant [5] found that planners in a manufacturing environment spend 80-90% of their time determining which constraints affect their schedule. Viewing planning as the satisfaction of constraints imposed by the environment, makes representation and use of constraints a vital issue for knowledge-based planning. Constraint representation is recognized by many authors as being crucial to the knowledge-based planner. A detailed method is described by M. Fox for using constraint-directed reasoning. He distinguishes five categories of constraints [15]:

- **Organizational goals**, set by the organization function as constraints for the planning.
- **Physical constraints** limit the functionality of a resource, activity, e.g. different machine set-ups, maximum length that can be machined.
- **Causal constraints** define what conditions should be met before an activity can start. Precedence and resource requirements are examples of these.
- **Availability**[1] of resources.
- **Preference constraints** can be viewed as abstraction of the other types of constraints. Preference for a machine may be the result of costs or quality, but sufficient information does not exist to devise actual costs.

These constraint categories are represented, using a frame-based system. According to MacCallum, Fox employs three different types [26]:

- **Range constraints**: constrain the values of a attribute, attached to a frame.
- **Relation constraints**: define a relation between a domain and a frame.
- **Frame constraints**: define which constraints apply to a frame e.g. combinations of range-constraints.

1 *Comment: in my view availability is a limit in functionality of a resource. Availability can therefore be seen as a form of a physical constraint. Hence, to me this category is superfluous.*

Essential to constraint-representation, however, is that constraints come in two groups: **hard** and **soft** constraints. Hard constraints cannot be relaxed, soft constraints can. Examples of hard constraints are physical constraints, due dates. Preference for a resource is a soft constraint.

The reason for introducing this distinction is the different way in which a planner has to deal with them. Hard constraints are a part of the 'world' model and *have* to be satisfied. Soft constraints have a guiding task. The problem solving is guided by these, and when a path leads to a constraint conflict, the guide can be made to choose another path, i.e. the soft constraint will be relaxed [18].

Constraint management is one of the reasons that AI may prove to be more successful than OR, for there is little evidence of constraint management in OR methods [18].

A method for dealing with soft constraints is also suggested by Fox [15]. Fox's method is based on the concept that each constraint is assigned a weight. The weight is based on the absolute importance of a constraint as well as on its importance relative to other constraints. Relaxation of constraints is done by means of comparison of its weight to other constraints.

The general idea behind constraint representation in a knowledge-based planner is that constraints limiting the search space should be considered flexible, unless they are hard. In OR constraints are considered hard, where solutions are examined using, for example, sensitivity analysis (flexing the constraints).

Time

Another important issue in a knowledge-based planner is the representation of time. Although this subject is not relevant to this study, a brief overview will be given in this section of representation methods of time. Two methods of time representation are used in knowledge-based planners: the point-based temporal logic based on the work of Bruce [27], and the interval-based temporal logic as stated by Allen [28].

The basic concepts in the point-based theory are those of an event and an activity, collectively called occurrences. An event is an instantaneous change while an activity is a change over time in the set of world states. Underlying this theory is the assumption about the existence of a time line. Events and activities, described by two events, are related to points on that time line [29].

Interval-based representation argues that no event is instantaneous as suggested in point-based logic. It consists of intervals, during which the activity takes place. Actions are related, with relations of the sort: A before B, C meets D, F overlaps G. Allen defines 13 basic relations [30].

In [29] these two methods are compared. Elleby suggests a point based representation that incorporates a solution to limitations as given by the interval-based logic advocates.

Conclusions

In this chapter an overview has been given of the development in knowledge-based technology for job shop planning. From this overview a characterization of the problems and specific properties of knowledge-based approach will be presented. Finally a preliminary answer will be given to the question wether knowledge-based systems show improvements over current techniques.

A key issue for knowledge representation in job shop planning is the use of object-based systems: the high interaction and the dependency between entities are strong advocates for the use of object-based systems. Many authors claim object-based systems to be the best method for representing knowledge for planning ([4], [15], 16]).

Another key issue is the use of constraints in problem solving. The difference with the use of constraints in OR is that constraints in knowledge-based planners are considered flexible. Researchers that support this are, among others: Mark Fox (CMU), Tim Grant (Brunel University) and Navin Chandra (MIT).

The representation of time is also a problem specific to planning. In most knowledge-based systems, facts and conclusions are either true or false (with an uncertainty), whereas in planning situations facts have a limited validity, dependent on time.

Two aspects of which, to date, no evidence is found, are:

- The use of blackboards: the blackboard mechanism seems interesting for it incorporates a method for interaction between different knowledge sources.
- The use of planning heuristics: contrary to the often mentioned main advantage of knowledge-based systems: the use of human (heuristic) knowledge, there is no evidence to conclude that these are used in existing systems.

We are now able to give a provisional conclusion regarding the improvement knowledge-based systems can provide for planning. A major advantage of knowledge-based systems would be if they generated better schedules than conventional programs. It seems, based on the literature consulted, that knowledge-based planning systems can generate better plans. The following illustrates this.
A plan is considered better if it:

- is more accurate;
- can be generated much faster;
- is more easily interpretable.

A vital factor is the **quality**: can a knowledge-based system generate more accurate schedules than conventional systems? Ergo, does a knowledge-based system provide more powerful scheduling techniques? Knowledge-based systems are problem solvers employing methods used by humans. Conventional planning systems generally use abstract methods, e.g. mathematical programming, priority rules. Conventional techniques are rigid and have problems reacting to changes often occurring in job shops, moreover they have problems dealing with the uncertainty that is common in job shops. In view of this it is to be expected that knowledge-based systems **can** generate better schedules: they are more suitable for the stochasticity and changes in job shops than conventional programs. This is supported by a number of authors ([34], [5]).
A factor which also improves the schedules is the simplicity with which they can be interpreted, how **user-friendly** is a knowledge-based system. One can imagine that results coming from a program cannot be used if it is impossible to interpret them. On this point knowledge-based systems definitely have an advantage over conventional systems. Three reasons support this:

- Knowledge-based systems consist of a general problem solver, the inference engine, that is able to draw conclusions on information from the user as well from knowledge contained in a knowledge base. The knowledge base and inference engine are separate modules. It is therefore much easier to understand the knowledge governing the problem, compared to systems where control and knowledge are mingled. If users can grasp the knowledge it is easier to comprehend how the schedule is created.
- Knowledge in knowledge-based systems is represented using methods which link up with the way humans think. These

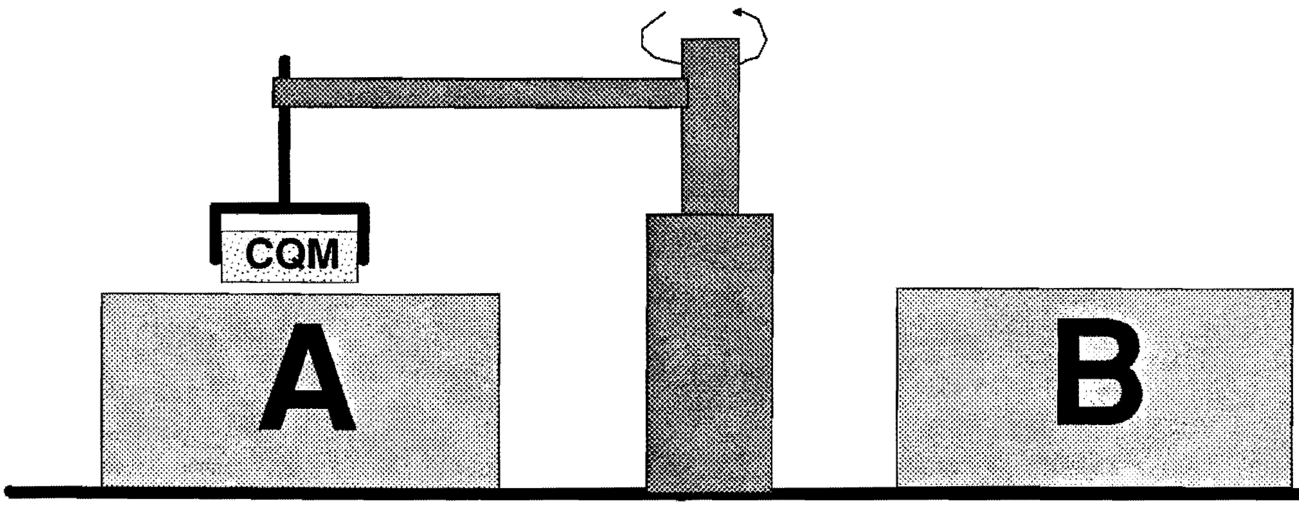
representation methods make it relatively simple to comprehend the represented knowledge.

- Most knowledge-based systems have an explanation facility which gives a report on the search process for a solution.

Another factor of influence is the **performance** of knowledge-based systems compared with conventional systems. A great advantage would be if knowledge-based systems generated schedules faster that at present. It then would be able to react alertly to changes in the environment, resulting in more accurate schedules. On this point there is no evidence that knowledge-based systems are able to generate schedules more speedily. On the contrary, present knowledge-based systems use software tools that are slow relative to current imperative languages e.g. 'C' or 'FORTRAN'.

Besides the performance, the effort needed to create a knowledge-based systems is a drawback. Knowledge-based systems try to emulate human behaviour in problem solving. A compelling demand is the requirement to understand how humans solve, in this case, planning problems. A problem is that generally humans are not aware of the methods they use to solve problems: they know *how* to solve a problem but do not know *what* they do when they are solving a problem. Extracting this implicit knowledge is a task which is considered to be the bottleneck in the development of knowledge-based systems.

PART II
The CQM Problem



5

THE CQM Problem

The main task of this study was to develop a prototype planning system based on knowledge-based technology. This prototype should be designed to solve a planning problem supplied by CQM. In this chapter the CQM problem will be presented in detail.

The CQM problem consists of two related planning problems in a workshop. These two problems will be discussed in the two following sections. A complete description of the CQM problem can be found in the appendix.

Problem I

In a workshop there are two machines ordered in a line lay-out (fig. 4). On these machines products are being manufactured. The following premises apply to this shop:

- Products are first processed on machine A and then on machine B.

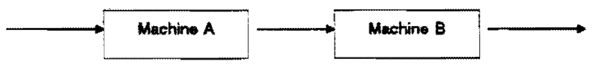


Figure 4: The workshop lay-out.

- There are 10 different products, each with its own processing times (see appendix).
- The orders to be processed in a week will be known at the end of the previous week.
- An order has to be ready on machine A, before it can go to machine B.
- There is no set-up time between different product types.
- Both machines have a capacity of 3000 minutes a week.
- The objective is to complete the orders in the shortest time (minimum timespan/makespan).
- Orders may wait between A and B.

The planner uses the following method to sequence these orders: he uses a 'left to right' and 'right to left' plan. He selects the order with the shortest processing time on A, as the first to be processed. He then selects the order with the shortest processing time on B, as the last to be processed. He repeats this selection procedure until there are no orders left.

Problem II

This problem is basically the same, but has the following differences:

- The change from one product to another takes 10 minutes set-up time on A, as well as on B
- Orders may be split in partial orders.

In this problem there are two planners. Planner A uses the same method as the planner in Problem I. Planner B uses the following method:

- He first examines which machine is the bottleneck, i.e. which machine has the largest amount of processing time.
- For that machine he groups the orders with the same products.
- He then makes a plan for the other machine so that there is no (or very little) idle time on the bottleneck.

Some remarks on this planning environment

The shop is an intermittent flow shop (see page 9): the routing is uniform. Orders flow through the shop as a whole. In this case the planning problem is reduced to a sequencing problem. The sequence of orders determines the start times on machine A and on machine B. Mere sequencing reduces the complexity of the problem: there is no need to consider problems like substitutability of machines, finite loading, alternative routing, etc..

The problem is deterministic: there is no uncertainty in, for example, processing time or machine performance. Determinism reduces the problem for there is no need for representation of uncertain information or methods dealing with such. Generally though, there is stochastic behaviour in shops.

6

A framework for a knowledge-based planner

In this chapter a framework for a knowledge-based planner will be presented. The goal of this framework is to provide a basis on which the prototype planning system for the CQM problem can be built. The framework is more general: it is also intended to be used for other job shop planning problems. The subjects presented in this section are grouped around two aspects of knowledge-based systems: knowledge representation and control strategy.

Knowledge representation

To make a choice for a representation method it is essential to identify and categorize the knowledge important to the problem domain. Hence, the presentation of this subjects flows along the following lines:

- Identification and categorization of knowledge.
- Representation mechanism: the choice for a representation method given the categorization of knowledge.
- Knowledge: what will be represented?

Identification and categorization

Viewing the CQM problem, one thing is very obvious: we have two major categories of knowledge.

The first category consists of items like 'orders', 'machines', etc.. This category could well be described as the **environmental knowledge**. The strong relation between the entities that form this categoryis striking: an order consist of products, products are produced on machines, etc..

The second category could best be described as the **heuristic knowledge**. An example of this knowledge is the method with which orders are planned: knowledge describing how to deal with the environmental entities.

When the first category is examined closely, it can be divided into two types, which show different behaviour during the planning process. On one side there is the environmental knowledge that is static and does not change during planning. This type describes with which restrictions the different entities should comply. On the other hand, there are the entities that change during planning: these are dynamic. The item 'order' is a static entity describing what properties a real order has. A real order inherits properties from 'order'. However, contrary to 'order', it alters during the planning process (i.e. start times change etc.). The division into two categories can therefore be refined by subdividing the environmental knowledge in a static part and a dynamic part.

Representation mechanisms

In Chapter 3 two major methods for knowledge representation are presented: rule-based and object-based systems. There it is stated that object-based systems provide a method for representing knowledge in a structured way, whereas rule-based systems are very suitable for representing heuristic knowledge. The nature of the environmental knowledge suggests the use of an object-based system: environmental knowledge being knowledge with strong mutual relations. Especially frame-based systems seem promising: frames are related through IS-A relations, e.g. the frame 'ORDER X' IS-A 'ORDER' frame. Frames can inherit information from classes higher up the hierarchy. Hence, in the CQM problem elements of the dynamic knowledge can inherit the description that they must comply with from the frames, representing the class they belong to: the static knowledge as defined in the previous subsection. A **frame-based system** is therefore the best choice for representing the environmental knowledge.

Looking at the heuristic knowledge the choice for a **rule-based system** would be obvious. Three reasons support this:

- Rule bases can easily be incremented. Hence, it is easy to implement improved methods of sequencing.
- Rule-based systems can react to changing data without requiring detailed prescience about the flow of control. There is no need for exact knowledge of how the sequencing will be performed, only rules that react to situations.
- The methods used here can easily be transformed into condition-action rules.

Instead of making a compromise between these two methods: rules or frames, the two representations will be integrated. Having two major categories, it seems sensible to do so. Moreover, structuring the knowledge base this way is more understandable to the user.

A model for the environmental knowledge

Beside a representation method, a vital part of the framework is a model of the knowledge embedded in the knowledge base. In this section a model of the environmental knowledge will be presented. Due to the limited time for this study a model for the heuristic part of the knowledge base has not yet been developed.

Basic ideas

A vital idea for the environmental knowledge model is its independency of scheduling methods: it should not contain elements that result from, or depend on a scheduling method. Otherwise the model is brittle, i.e. it can only be applied to a small set of problems.
Originally the model was created for an environment as stated in the CQM problem. However, the model as defined here is more general in view of the application of the model for future problems. The environment described here is that of a production workshop.

For the construction of this model the following two concepts are used:

- System-design theory: using this theory a general structure for the model will be defined.
- Workshop model: constructing the model by using the structure of the workshop.

Systems-design theory

A result of the interviews conducted with OR scientists at the Centre for Quantitative Methods was the use of systems-design models for solving scheduling problems. A scheduling problem is visualized by them as follows (fig 5):

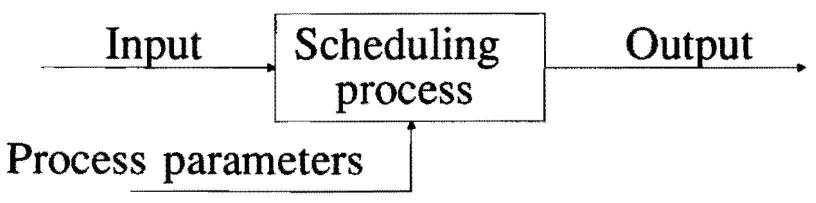


Figure 5: A systems design model of a scheduling problem.

A process is viewed as a black box, connected with the outside world through an information source: the input, and one information recipient: the output. Furthermore, the information governing the process comes from a second information source: the process--parameters.

Analogously, three classes of frames representing the classes of data are defined: input, output and process-parameters. The frame-classes form part of a meta model in which the model of the workshop has to fit.

Workshop model

The construction of a model for the workshop follows from the properties of the workshop: what entities form the workshop and what is the relation between them? In this subsection the properties of the workshop under consideration will be discussed. Also the entities to be represented in frames will be distinguished.

The general idea is that of a workshop in which a number of products are being manufactured on customer order. The model for such a workshop is based on the following assumptions:

- The workshop consists of a number of resources.
- The goal of the workshop is to manufacture products on customer orders. An order consists of the request for the

delivery of a number of products[1] belonging to one product type.

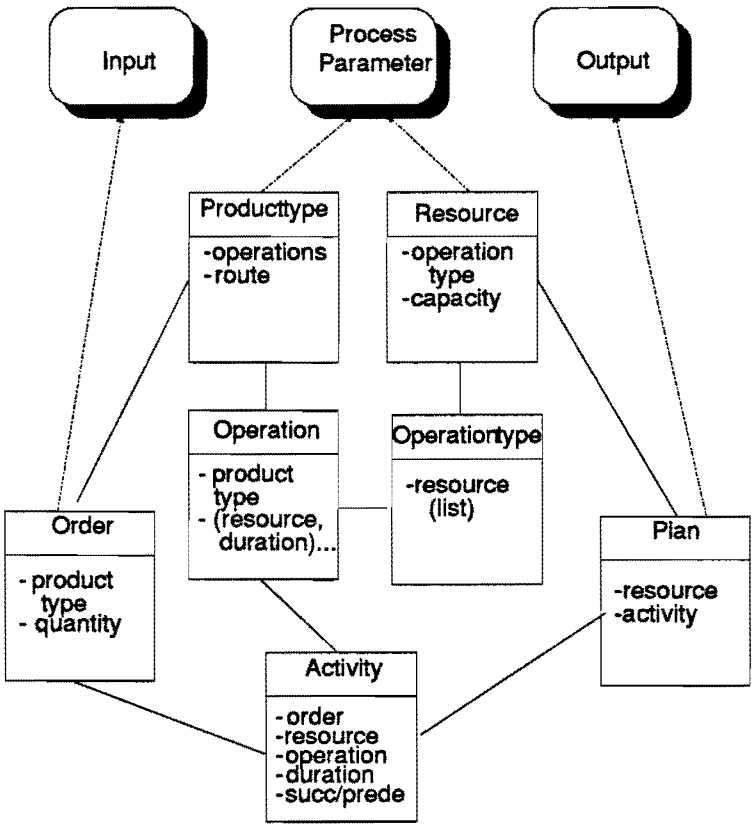
- Every product belongs to a product type.
- To obtain a product a sequence of operations has to be performed. Every product type has its own sequence of operations.
- An operation is the performance of a type of operation on a specific type of resources for a specific period of time (not necessarily the same duration for each resource).
- On each resource a number of operation types can be performed. An operation type describes operations possible on those resources: operation types are resource dependent, whereas operations describe a specific operation to be performed on a specific resource and are also product type dependent.
- To complete an order a sequence of operations has to be performed on a number of products. These are called activities. Activities are the 'jobs' to be scheduled.
- The result of the scheduling process is a plan. The plan consist of a GANTT chart of the resources and the planned activities.

In short, the process that is executed in the workshop can be defined by the following entities: **product type, resource, operation type, operation**.
The orders form the input of the workshop, whereas the plan represents the output. From the orders that have to be completed, activities are derived. Activities form the basic input of the workshop: they will be scheduled. A network representing this model is given in fig. 6.

Control strategy

In Chapter 3 it is noted that the choice for a control strategy depends on the knowledge representation. The prototype for the CQM problem employs two different representation methods: it could be regarded as having two knowledge bases: a frame base and a rule base. Control strategy in the knowledge-based system

1 *Note that in this report by product a finished product is meant.*



1. (resource, duration)... : A list of resources on which operations can be performed, together with matching durations.

2. succ./prede. : successor and/or predecessor activity.

Figure 6: The network representing the environmental knowledge.

acts on two levels: within the knowledge base and between the knowledge bases.

Within the knowledge base the choice for the separate knowledge bases is clear: an action-centred control in the rule base, and an object-centred control in the frame base (see page 23).

More difficult, however, is the interaction between the different bases. Should an action-centred control be used; rules accessing the frames and guiding the reasoning process, or an object-centred control; frames triggering rules, which alter frames. Or should a request-centred control be used e.g. a blackboard mechanism, with which the two knowledge sources communicate. There is not evidence yet to take a final decision on this matter.

A blackboard system seems most attractive for it is suitable for handling different knowledge sources, which could be useful in this case [6].

Conclusion

In this section the framework is presented of a knowledge-based planner for a job shop were products are being manufactured. The most important feature of this framework is that it consists of a knowledge base containing two major categories of knowledge: environmental knowledge and heuristic knowledge. The environmental knowledge represents the elements that define the workshop. These elements are represented as frames in a frame-based system. The heuristic knowledge represents methods used for scheduling. The methods are represented as rules.

A subject for which there is still no decision is the choice for a control strategy. Among the three methods a form of request-centred control, the blackboard mechanism, seems most promising.

7

ScheduleTalk: an implementation

Based on the framework defined in the previous chapter, a prototype planning system is developed for the CQM problem. In this chapter this prototype will be presented. The prototype is called ScheduleTalk: a Scheduler developed in SMALLTALK-80. The following subjects regarding the development of ScheduleTalk will be discussed in this chapter:

- The choice of a software tool and description of this tool.
- The functional description of the shell, on which ScheduleTalk will be built.
- The description of ScheduleTalk: ScheduleTalk is based on the framework presented in Chapter 6. In this section extensions and differences compared to the framework are discussed.

Choice of a software tool

In the previous chapter it is stated that a knowledge-based system used for job shop planning should use a hybrid representation mechanism: a frame-based mechanism complemented with rules. The tool to be chosen should, therefore, support a hybrid representation. In view of the limited time in which the prototype should be implemented, a special purpose tool is preferred to a general programming language: knowledge-based systems can be built in a much shorter time using a knowledge-based system shell than in a programming language. A knowledge-based system shell is a knowledge-based system with an empty knowledge base. When a tool had to be chosen, there was no shell available that supported a hybrid knowledge base. The only possibility left was to use a more general programming language.

The following languages were considered as possible tools:

- **PROLOG**: a logic-based language that tries to prove (PROLOG) relations, using backward chaining, unification and backtracking as needed. PROLOG has strong similarities to rule-based languages.
- **LISP**: Among the programming languages used for development of knowledge-based systems, LISP is the most popular, because it provides an easy and flexible method for manipulating symbols.
- **SMALLTALK-80**: in this object-oriented programming language, all elements are objects. The objects communicate by sending messages to each other.

These three are the most interesting among the programming languages available at the CAD Centre. SMALLTALK-80 was chosen for four reasons:

- Although SMALLTALK-80 is presented as a programming language, it is more. In SMALLTALK-80 an extensive user interface is defined in the language itself. However, it is not specially designed for knowledge base development: functions and data structures still need to be defined within SMALLTALK-80.
- SMALLTALK-80 is a language in which frames are easy to represent: frames can be represented as a class of object. Inheritance can be implemented by using message-passing.
- There is evidence of the use of object-oriented languages in other research centres, regarding the development of knowledge-based systems [5].
- It is likely that a knowledge-based system will also use conventional techniques as an aid for scheduling. Among conventional methods, event-driven simulation is a powerful aid. SMALLTALK-80 is also very suitable for the implementation of event-driven simulation [31].

SMALLTALK-80

SMALLTALK-80 is a programming environment based on objects. SMALLTALK-80 is the result of research at the XEROX Palo Alto Research Centre (XEROX PARC) to create a powerful information system in which the user can store, access and manipulate information so that the system can grow as the user's ideas grow.

The research for this system is mainly based on the following two areas [31]:

- A language of description (the programming language) which serves as an interface between the models in the human mind and those in the computer hardware.
- A language of interaction (the user-interface) which matches the human communication system.

As a result, of these research areas, SMALLTALK-80 consists of, beside a programming language, an extensive user interface.

Furthermore, it is based on just a few concepts: it consists of objects. Every object belongs to a class. Objects interact by sending messages to one another. The following subsections will present SMALLTALK-80 using the key words: object, class and message.

Objects

An object is the basic element of SMALLTALK-80 : every component in the system is an object. Numbers and characters are objects, as well as e.g. text-editors and files. An object consists of a private memory and a set of operations. Information belonging to an object can not be accessed by another object. The nature of an object's operations depends on the type of component it represents. Objects that represent numbers compute arithmetic functions, objects representing data structures store and retrieve information.

Messages

An important aspect of computer programs is that they consist of parts that exchange information. In SMALLTALK-80 information is transferred between objects, using messages. However, objects cannot access each other's private memory. A message in SMALLTALK-80 does not retrieve information but is only a request for information. The message specifies which operation should be carried out. The receiver of the message, however, determines how the operation will be performed. The set of messages to which an object can respond is called its interface with the rest of the system. Interaction with the rest of the system is only possible through the interface.

These two properties of SMALLTALK-80 mean that the implementation of one object is independent from other objects. It only depends on the messages to which it responds.

Classes and Instances

Another important concept behind SMALLTALK-80 is the classification of objects: every object in SMALLTALK-80 is an instance of a class. A class describes a set of objects that all represent the same kind of system component, e.g. each number is an instance of the class 'Number'.

Instances are the individual objects described by a class. Instances of a class have the same message interface, but have their own private memory: instances are similar in both their public and private properties. The private properties of an object are the set of so-called instance variables and the set of so-called methods. Instance variables contain the local information whereas the methods describe how to carry out the object's operations.

FrameTalk

ScheduleTalk is a knowledge-based system to be used for a scheduling problem. It comprises the knowledge governing the CQM problem embedded in an knowledge-based system. Prior to the implementation of ScheduleTalk an empty knowledge-based system, a shell, has to be created within SMALLTALK-80. This shell, called FrameTalk, will be presented in this section. A complete description of FrameTalk, together with the listing can be found in the appendix.

FrameTalk is an implementation of a frame-based system created in SMALLTALK-80. The concepts behind FrameTalk are derived from literature regarding frames ([7], [32]). However, these concepts are elaborated to form FrameTalk on the basis of my own ideas.

The detailed description of FrameTalk will be given by using the four key words: Class, Instance, Inheritance, Demons.

In FrameTalk two different types of frames can be distinguished: frame-classes and frame-instances.

A **frame-class** represents a specific type of object. The following properties of such a type are defined by the frame-class description:

- The attributes the type has (i.e. which slots).
- The common values for these attributes, if present.
- The way that the type of object reacts to changing values of its attributes.
- The way an object of that type acquires value for its slots.

Note that a frame-class can be a subclass of another frame-class. Therefore, it need not contain this information as it may be inherited from the superclass. A frame-class is a static object: during a consultation of the knowledge base, information stored in a frame-class does not alter.

Frame-instances are real object belonging to an object type. They form a set belonging to a frame-class. Frame-instances inherit the slots as well as the default values of these slots from the frame-class descriptions. Inheritance from a frame-class is not the only method for a frame-instance to acquire values for its slots: there are three other methods: first, a value for a slot can be inserted by the user. If a default value for that slot exists it will be replaced. If no default value exists and no value is inserted by the user, the chain of frame-classes is searched for a demon. The demon will be fired, resulting in a value for that slot.

The last method uses a network of related frames. Beside an inheritance network of classes there is also an inheritance network of related frame-instances. For Instance: a 'bicycle', inherits the value 'wheels' from its class 'vehicle', whereas it also can inherit values from the related frame-instance 'cycling', an instance of the class 'sports'. A frame-instance can have more than one frame-instance to which it is related. Searching for a value in the related frame-instances is the same as going through a search tree of frame-instances. Where the root of the tree is the frame's slot for which a value is required. When searching through the chain of frame-classes is not successful the tree of frame-instances is searched for a value. If none is found, the chain of frame-classes is searched for that frame.

A special role in FrameTalk is performed by demons. Beside suppliers of values for slots (ifNeeded-demons), there are also demons that will be fired when values of slots are changed (ifChanged-demons). An important concept behind FrameTalk is the nature of the demons. The shell on which ScheduleTalk should be built should have a hybrid character. So far FrameTalk only accommodates a frame-based system. The rule-based part consists of the demons: the demons in FrameTalk are rules that are fired. This creates the basis for a hybrid knowledge base.

In short FrameTalk can be depicted as follows:

- FrameTalk is a hybrid knowledge-based system shell containing frames whose slots can contain rules.
- Frames come in two types: frame-classes[1] and frame-instances.
- Frame-classes represent a type of object. They are static in their behaviour: they do not change during a consultation of the knowledge base. A frame-class can contain rules needed to perform part of the reasoning process.
- Frame-instances represent an object. Every frame-instance belongs to a frame-class. Frame-instances are dynamic; information in frame-instances changes during the reasoning process.
- A frame-instance inherits information from its frame-class and the frame-class' superclasses. A frame-instance can also inherit information from related frame-instances. FrameTalk allows multiple inheritance.

ScheduleTalk

In Chapter 6 a framework is presented for a knowledge-based planner in a job shop situation. This framework is designed in connection with the building of a prototype planning system for the CQM problem. This prototype is called ScheduleTalk. ScheduleTalk is based on the framework stated in Chapter 6. But as ScheduleTalk is devised for a specific planning problem, it differs slightly from the framework. In this section the essential differences between ScheduleTalk and the framework will be discussed. A detailed description of ScheduleTalk will be given in the appendix. ScheduleTalk is still under construction and so far only the first problem of the CQM problem has been solved. The results are also presented in this section.

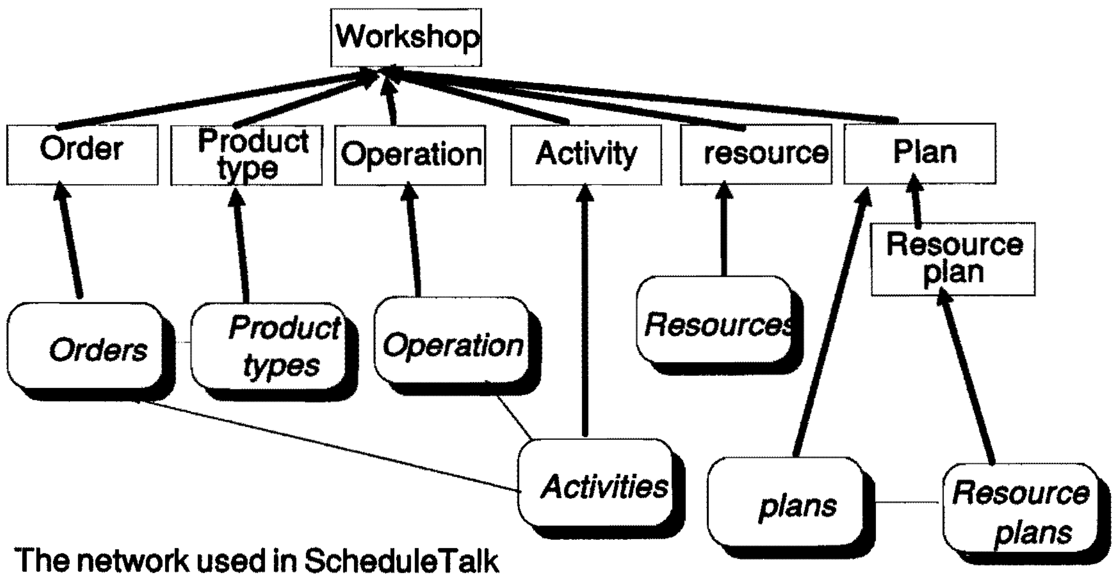
There are two fundamental differences between ScheduleTalk and the framework:

1 *Note: The classes and instances used in FrameTalk are not SMALLTALK-80 classes and instances.*

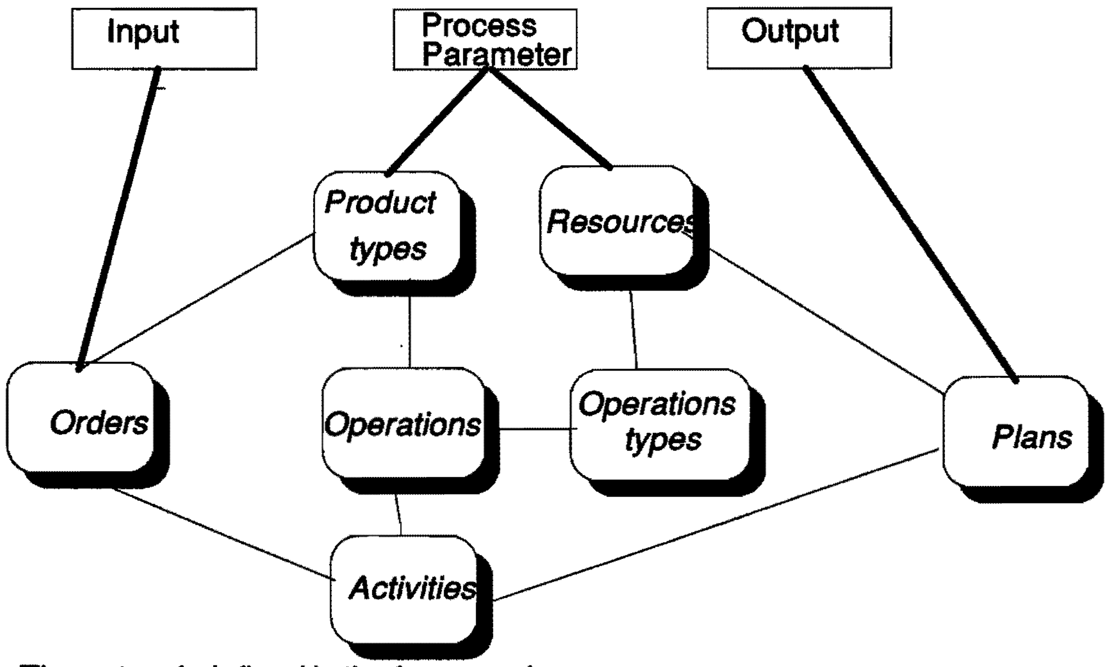
- **Knowledge representation**: the representation is adapted to the capabilities of FrameTalk. It has not yet been possible to implement the demons in FrameTalk as rules, therefore FrameTalk only supports frames whereas an important aspect of the framework is the use of a hybrid knowledge base. This allows demons to be represented as rules. In FrameTalk demons are SMALLTALK-methods and do not resemble rules. Using SMALLTALK-methods instead means that the construction of a schedule is identical to methods used in any other conventional planning system. Therefore the lack of rules diminishes the value of ScheduleTalk: based on ScheduleTalk it is difficult to make conclusions about the feasibility of knowledge-based systems for planning.
- The **knowledge model** resembles closely the knowledge model of the framework. A few differences are, however, obvious. These differences are best illustrated by the comparison in the following figure (fig. 7).

The following specific differences are introduced in the knowledge model resulting from the properties of the CQM problem.

- The framework network contains only the real elements that define the environment. The ScheduleTalk network contains beside these elements also a class description of the elements. This class description makes it possible to describe the behaviour of the elements. Although different to the network it is in agreement with the categorization of knowledge as discussed on page 42.
- Resources in the CQM problem can carry out only one type of operation. Hence, there is no need for representation of 'Operation type'.
- The frame 'Plan' in ScheduleTalk is expanded by adding a subclass: the 'Resource plan'. 'Plan' contains a plan for the total shop, whereas 'Resource plan' only contains a plan for one resource. 'Resource plan' is also more detailed than 'Plan'. 'Plan' in ScheduleTalk contains the sequence in which the orders are processed. The 'Resource plan' contains the sequence of orders, together with the matching start and finish times.



The network used in ScheduleTalk



The network defined in the framework

Figure 7: A comparison of two knowledge models.

Results

It has not yet been possible to solve both CQM problems using ScheduleTalk. Only for the first is a solution given. In this subsection this solution will be given by means of a GANTT chart created by ScheduleTalk. Both the input values and the complete solution are given in the appendix. Figure 8 represents a GANTT chart of six orders to be processed on two machines, A and B.

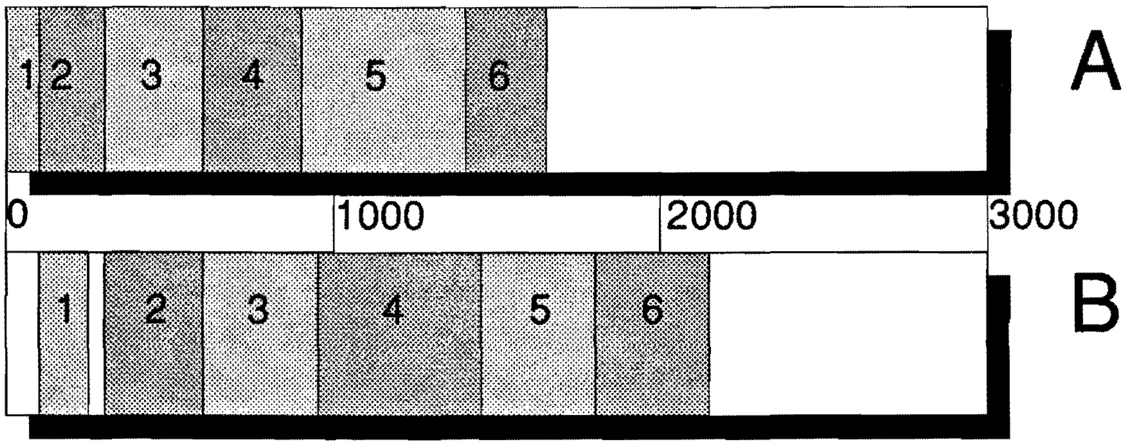


Figure 8: A GANTT chart, created by ScheduleTalk.

Conclusion

In this chapter a description of a planning system using knowledge-based technology is presented. This description contained the following subjects.

- The choice of a software tool;
- A description SMALLTALK-80, the chosen tool;
- The frame-based shell FrameTalk;
- The knowledge model ScheduleTalk.

The choice of a software tool is a vital aspect of knowledge-based system development. This project made this point very clear. To create a knowledge-based system it is important to choose a tool that accommodates the representation used for that system. Due to the lack of a custom hybrid shell, given the limited time for this project, too much time had to be spent on the construction of a shell. If a suitable tool had been available, far less time would have been spent on the creation of a frame-based shell. It then would have been possible to create a better knowledge model for the prototype, as well as test different models.

Nevertheless, SMALLTALK-80 is a suitable tool to develop a knowledge-based system for planning. SMALLTALK-80 is not designed for the development of knowledge-based systems but is designed for more general purposes. The general design makes it interesting for knowledge-based planning: it is also suitable for conventional techniques, e.g. simulation.

The implementation of SMALLTALK-80 used in this project, a version for IBM-PC, proved to be an excellent choice. The performance was sufficiently fast, moreover, it proved to be a system not sensitive to failures. However it is expected that for real problems SMALLTALK/V will be inadequate. The main reason for this is the limited amount of memory: an MS-DOS PC can only address 640 kilobytes. The limited memory space restricts the available number of objects and thus limits the number of frames.

For the prototype three subordinate goals were set.

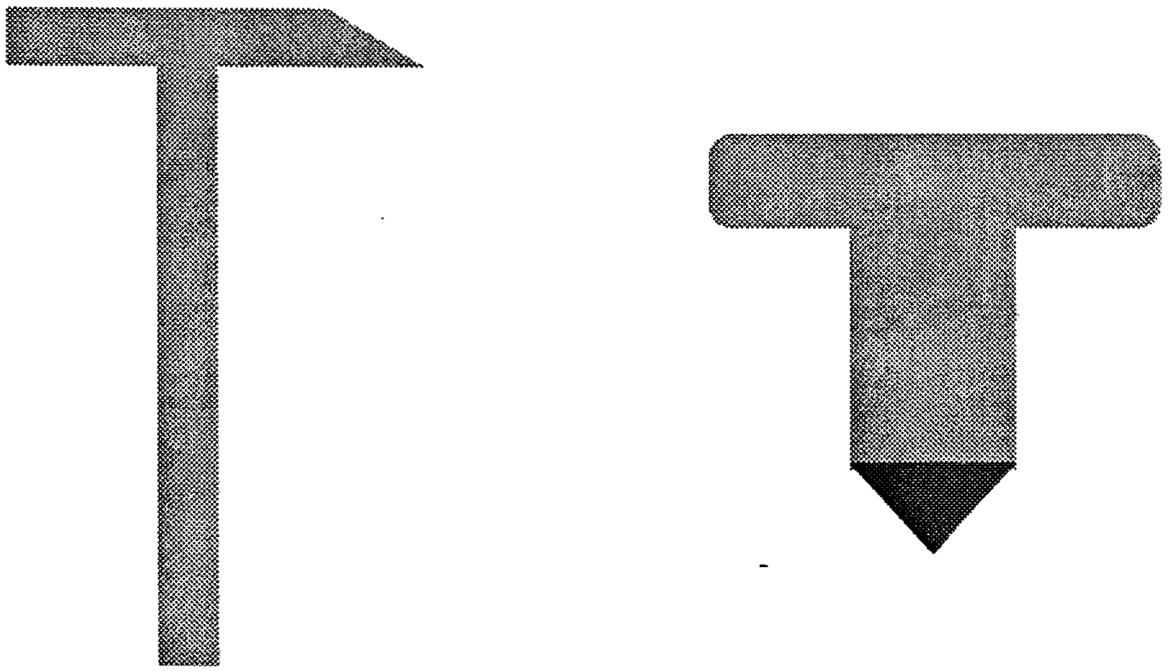
- The construction of a knowledge-based system that generates production plans and resource schedules for a limited class of problems. Within its limited scope the prototype should show improvements.
- Identification and documentation of the knowledge categories involved in the planning process.
- Devising and demonstrating representation mechanisms that will accommodate the previously identified categories of knowledge. This to help structure the knowledge which is the basis of the planning system.

It proved to be possible to create a knowledge-based planner. However, due to the restrictions mentioned above, it was not possible to test if knowledge-based systems show improvements regarding the problems stated on page 2. Nevertheless, the development of the prototype made it possible to define a framework for a knowledge-based planner. The answers to the last two questions are given by means of this framework (see Chapter 6).

The development of ScheduleTalk confirmed the importance of knowledge acquisition: it proved to be the most difficult task of this study.

PART III

Knowledge-based systems: tool or toy?



8

Conclusions

The main goal of this study is to examine the benefits knowledge-based systems can have for job shop planning. In particular, the goal was to obtain answers to the following questions:

- Is it feasible to implement a job shop planning system by means of knowledge-based technology?
- What are the specific demands for a knowledge-based system that is to be used in job shop planning?
- What should be the structure of the knowledge base?

Summary of results

To answer these questions two tasks were performed. First, literature was consulted regarding knowledge-based planning. Secondly, a prototype was constructed for a stylized planning problem. Before answering the questions an overview of the results of these tasks will be given.

The literature study

Planning is a subject that has attracted great interest among researchers in the field of AI the last few years. The reason that it has acquired such interest is a result of the weaknesses of current planning systems to represent empirical knowledge. At present, OR techniques use abstract, mathematical techniques but are not able to use expert knowledge of human planners. This inability makes them rigid. Knowledge-based systems are designed to represent human expert knowledge ([33], [5]).

From the large amount of literature available, a number of aspects were significant. The essence of most of the systems is that they consist of a knowledge base founded on a model of the knowledge governing the problem domain. This knowledge consists of highly related entities. It is considered important to use a representation that allows such a model to be represented. The representation used is that of an object-based system.

Another vital aspect is the specific role of constraints. Constraints used by OR are rigid bounds of the solution space. Knowledge-based systems use flexible constraints that function as guides of the reasoning process.

The CQM problem

The CQM problem deals with planning in a job shop. For such an environment a framework for a knowledge-based planner is defined. Based on this framework a prototype planning system is developed. The framework is determined by the following concepts.

The knowledge that describes the environment is divided in two categories: an environmental and a heuristic category. The environmental category consists of entities like 'resource', 'operation', that define the job shop. The heuristic part contains the methods and rules used to create a schedule in this job shop.

The environmental knowledge consists of entities that are strongly related. This type of knowledge is best represented using an object-based system. The heuristic knowledge consist mainly of elements like rules of thumb, heuristic methods or mathematical techniques. For such knowledge a rule-based system presents the most suitable representation method. In view of this separation of knowledge categories it was decided to use a hybrid system which allows both rules and frames to be represented.

The second important concept is the model of the environmental knowledge. The model consists of a network of nodes, representing the entities that define the job shop. A knowledge-based planning system for job shops should use a knowledge base containing knowledge founded on that model.

Based on this framework the construction of a prototype has started. The prototype was called ScheduleTalk: a Scheduler developed in SMALLTALK-80. ScheduleTalk is still under construction, it is therefore not yet possible to compare the results of such a knowledge-based system with conventional systems. A significant problem that arose from the development process was the difficulty to acquire the knowledge in ScheduleTalk. A significant advantage was the ease to represent the knowledge in ScheduleTalk.

Answers to questions

Based on the results of the literature study and ScheduleTalk answers to the questions posed above will be formulated.

Feasibility

The most important questions in this stage is to give an answer to the following question:

- Is it feasible to implement a job shop planning system by means of knowledge-based technology.

It is now possible to give a provisional answer to this question: it seems feasible to use knowledge-based planning systems. The arguments will be discussed in the remainder of this subsection.
There are a number of factors that decide whether it is feasible to use knowledge-based planning systems. In general, feasibility can be determined by the following two measures:

- Does a knowledge-based planning system produce better schedules?
- How much effort does it cost to create a knowledge-based planning system?

Knowledge-based systems are likely to produce better schedules as they can enhance conventional techniques with methods developed by human planners through experience. Furthermore knowledge-based systems can deal with inexact information, allowing them to take inexact information from the shop floor into account, as well as give a measure of reliability to their plans. A significant advantage is also the representation of knowledge. Knowledge-based systems use programming languages that enable human knowledge to be represented similarly to the way humans think. Together with the explanation facility, users of the knowledge-based planning system (i.e. planners) can easily comprehend the planning process and can, if needed, easily alter it. A disadvantage could be that, for the moment, the computational effort might be greater than the effort needed by conventional systems.

The effort needed to create a knowledge-based system is not to be underestimated. Knowledge-based systems try to emulate human behaviour in problem solving. A compelling demand is the requirement to understand how humans solve, in this case, planning problems. A problem is that generally humans are not aware of the methods they use to solve problems: they know *how* to solve a problem but do not know *what* they do when they are solving a problem. Extracting this implicit knowledge is a task which is considered to be the bottleneck in the development of knowledge-based systems. A solution for the knowledge acquisition could well be the development of a knowledge model of job shop planning. Using such a model it might be found that knowledge-based systems can be built faster.

The experience gathered during the development of ScheduleTalk confirmed this: creating a model for the knowledge proved to be most difficult task. Regarding the universality of the proposed framework, it is to early to give an answer.

If ample time is spent on the acquisition of domain knowledge and the creation of a knowledge model, knowledge-based systems are an improvement on conventional techniques: they are enhanced by knowledge-based technology. This makes it feasible to implement planning systems using knowledge-based technology.

Structure and demands

In this subsection the two other questions will be discussed.

- What are the specific demands for a knowledge-based system that is to be used in job shop planning?
- What should be the structure of the knowledge base?

These questions will be considered together because they are related to one another. In the previous subsection it is mentioned that knowledge governing the planning domain contains many related items. This conclusion is supported by the experience with ScheduleTalk. Representing this knowledge requires a method that allows such relations to be encoded. A powerful method that is suitable is the frame-based approach ([4], [21], [22]). Furthermore, the planning techniques used can easily be mapped onto a rule-based system. The conclusion that can be drawn is that knowledge-based systems used for job shop planning should use a hybrid knowledge base, containing frames as well as rules. The structure to be used could well be similar to the one proposed in Chapter 6.

Future research

It is obvious that this project was too limited to give a decisive answer regarding the use of knowledge-based systems. To give a decisive answer it is therefore required to further research the capabilities and limitations of knowledge-based systems. In this section some subjects will be presented on which future research will give more clear conclusions regarding feasibility and architecture of knowledge-based systems.

Framework

In Chapter 6 a framework is presented for knowledge-based systems to be used for job shop planning. This framework is created in connection with a stylized planning problem. A possible subject of future research would be to test the validity of this framework in other more real environments. This research could improve this framework or suggest another framework. Furthermore, the application of knowledge-based systems in more real environments will give more decisive answers to the questions mentioned in Chapter 1.

Time, constraints, etc.

A number of important aspects have not been investigated. Among these elements are: Time representation, constraint representation. Generally in knowledge-based systems facts are either true or false (to a certain extent). In planning, facts have a limited validity. Representation of time is an essential element. Constraint representation is considered important while planners spend most of their time managing (i.e. adjusting) constraints.

Another subject worth researching is that of the control strategy. In Chapter 3 it is mentioned that three types of control strategy are used. An improvement to the framework would be the choice for a suitable strategy.

Conventional techniques

Finally, it would be interesting to learn how and which conventional techniques can be used within knowledge-based systems, or vice versa, how knowledge-based technology can improve conventional techniques.

References

[1] D.W. MacLeavey, S.L. Narasimhan
Production planning and Inventory Control
Allyn and Bacon, 1985

[2] W.M.J. Geraerds, J.W.M. Bertrand, J.C. Wortman
Syllabus Inleiding Productiebeheersing
Eindhoven University of Technology, 1984

[3] E.S. Buffa, W.H. Taubert
Production-inventory systems: planning and control
Richard D. Irwin INC., 1972

[4] M. Fox
Knowledge representation for decision support
in L.B. Methelie: Knowledge representation for decision support systems.
Elseviers, 1984

[5] T.J. Grant.
Lessons for O.R. from A.I.: A scheduling case study.
Journal of the Operations Research Society Vol 37 no 1, 41-57.

[6] D. Waterman
A guide to expert systems
Addison-Wesley,1986

[7] P.H. Winston
Artificial Intelligence
Addison-Wesley,1984

[8] A. Barr, E. Feigenbaum
Handbook of Artificial Intelligence

[9] E. Horwitt
Exploring Expert Systems
Business Computer Systems,3(4) pp. 48-57, 1985

[10] E.H. Shortliffe
Computer based medical consultation: MYCIN
Elsevier, 1976

[11] D. Hofstadter
Gödel, Escher, Bach: An eternal golden braid
Vintage, 1979

[12] S.M. Weiss, C. Kulikowski
A practical guide to designing Expert Systems
Rowman and Allanheld, 1984

[13] E.H. Shortliffe, B.G. Buchanan
A model for inexact reasoning in medicine
Mathematical Bioscience,vol. 23, 1975

[14] C.V. Negotia
Expert Systems and Fuzzy Systems
Benjamin/Cummings, 1985

[15] M. Fox
Observations on the role of constraints in problem solving
Proc. 6th Canadian Conference on Artificial Intelligence, 1977, 172-187

[16] A. Sathi, M. Fox, M. Greenberg
Representation of activity knowledge for project management
IEEE, transactions on Pattern Analysis and Machine Intelligence,7(5), 53 - 552

[17] T.J. Grant
Planning resource usage
Proc. 5th Alvey planning SIG workshop, March 1986

[18] P. Elleby, T.J. Grant
Knowledge based scheduling
in G. Mitra: Computer assisted decision making
Elseviers,1986

[19] A. Tate
A review of knowledge-based planning techniques
The Knowledge Engineering Review, vol 1, no. 2, 1985

[20] J.R. Slagle, H. Hamburger
An expert system for a resource allocation problem
Communications of the ACM,vol. 28, no. 9,pp. 994-1004

[21] J.C. Kunz, T. Bohura, H.J. Stelzner,R.E. Levitt
Contingent analysis for project management using multiple worlds
Proc. 1st National Conference on Applications of Artificial Intelligence in Engineering Problems

[22] I. Goldstein, B. Roberts
NUDGE, a knowledge based scheduling system
Proc. International Joint Conference on Artificial Intelligence, 1977,pp. 257-263

[23] T.J. Grant
An object-oriented approach to AI-planning and scheduling
to be published in G. Mitra: Expert Systems, Optimization and Process Control.

[24] P.C. Jackson, M.C. Maletz
Critical Path resource allocation using ART-viewpoints
Proc. 6th International Workshop on Expert systems and their Applications
Agence de l'Informatique, 1986,pp 405-415

[25] R.J. Brachman
On the epistemological status for representing knowledge
in N.V. Findler: Networks: representation and use of knowledge by computers
Academic press, 1979, pp. 3-50.

[26] K.J. MacCallum
Description of ISIS - a system for job shop scheduling
University of Strathclyde, Dept. of Ship and Marine Technology

[27] B.C. Bruce
A model for temporal references and its applications in a question answering program
Artificial Intelligence, 3, 1,pp. 1-25.

[28] J.F. Allen, Koomens
Planning using a temporal world model
Proc. International Joint Conference on Artificial Intelligence, 1983, pp741-747

[29] P. Elleby
In defence of point-based temporal reasoning
Proc. 5th Alvey SIG workshop, 6-7 March 1986

[30] E.P.K. Tsang
Plan generation in a Temporal field
Proc. 7th European Conference on Artificial Intelligence, 1986

[31] A. Goldberg, D. Robson
SMALLTALK-80: the language and its implementation
Addison-Wesley, 1983

[32] P.H. Winston, B.K.P. Horn
LISP
Addison-Wesley, 1981

[33] Isenberg, Randolf
Comparison of BB1 and KEE for building a production planning expert system
Philips report

[34] A. Sen, G. Biswas
Decision support systems: an expert systems approach
Decision Support Systems, vol. 1 197-204, 1985,

[35] A.A. Assad, B. Golden
Expert systems, microcomputers and operation research
Computer and operations research, vol. 13,no.2/3

Appendix

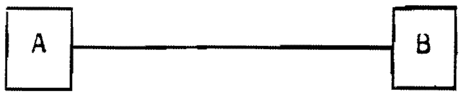
Twee voorbeelden van een schedulingsprobleem

In het volgende worden enkele aan de praktijk ontleende, maar sterk gestileerde schedulingsproblemen geschetst.
Het is niet de bedoeling ze in detail "op te lossen", maar meer om aan te geven hoe AI in de geschetste situaties van nut kan zijn en vooral hoe men vanuit AI het probleem denkt aan te pakken.

Geval A

In een afdeling staan twee machines A en B, waarop produkten gemaakt worden. Het volgende is aan de hand:

1. Produkten worden eerst op machine A bewerkt en vervolgens op machine B.



2. Voor de 10 verschillende typen die gemaakt kunnen worden gelden de volgende bewerkingstijden voor 1 produkt (in minuten):

Type	bewerkingstijd op A	bewerkingstijd op B
1	2	1
2	4	5
3	3	2
4	5	3
5	3	5
6	2	2
7	1	2
8	6	3
9	4	2
10	2	3

3. Aan het eind van elke week krijgt men te horen hoeveel produkten er van elk type gemaakt moeten worden.
   Bijvoorbeeld:

   Week 1: Type 2: 1 order van 25 stuks en 1 order van 75.
   Type 4: 1 order van 100 stuks.
   Type 10: 2 orders van elk 100 stuks en 1 order van 150 stuks.
   Overige types: 0.

   Week 2: Type 5: 1 order van 50 en 2 van 25 stuks.
   Type 7: 3 orders van 100 stuks en 1 van 200 stuks.
   Type 8: 1 order van 20 stuks en 1 van 80 stuks.
   Overige types: 0.

4. Pas als een order in zijn geheel is afgewerkt op machine A gaat hij naar machine B.

5. Omsteltijden van het ene naar het andere type zijn gelijk aan 0 minuten.

6. Max. capaciteit (incl. overwerk) mach. A: 3000 min./wk.
   Max. capaciteit (incl. overwerk) mach. B: 3000 min./wk.

7. Het is de bedoeling dat er zo wordt gepland dat het hele werkpakket voor een week zo snel mogelijk wordt afgewerkt (minimum time-span/ make-span).

8. Tussen machine A en B mogen orders liggen te wachten.

De planner gaat als volgt te werk:
Hij plant van "links naar rechts en van rechts naar links", d.w.z. hij plant de order die als eerste wordt uitgevoerd en neemt daarvoor de order die het snelst klaar is op machine A (zodat machine B snel aan de slag kan). Daarna plant hij de order die als laatste wordt uitgevoerd en neemt daarvoor de order die de kortste tijd vergt op machine B. Vervolgens plant hij de order die als tweede aan de beurt is op A, dan de order die voorlaatste zal zijn op B enzovoorts; steeds werkend van twee kanten en wel zo kiezend uit de nog niet geplande orders dat de machines geen of weinig stilstand hebben. Voorts worden eerst zoveel mogelijk ~~in~~ de langdurende orders gepland, zodat aan het eind er kortdurende orders resulteren waarmee het makkelijker is "gaten" te dichten. Als er tenslotte een plan ligt wordt gekeken of er door eenvoudige verwisselingen nog winst geboekt kan worden.

Geval B

Als geval A met als verschil dat:

- overgang van het ene produkt naar het ander een omsteltijd vergt van 10 minuten, zowel bij machine A als bij machine B;
- orders gesplitst mogen worden in deelorders. Een (deel-)order gaat in zijn geheel van A naar B. Er zijn nu 2 planners die elk hun eigen aanpak hebben;
- planner 1 handelt in grote lijnen als beschreven onder geval A;
- planner 2 kijkt eerst welke machine gegeven het werkpakket van een week de bottleneck is, d.w.z. de meeste minuten werk voor de kiezen krijgt. Voor die machine maakt hij een plan waarin orders voor eenzelfde type achter elkaar worden gepland.
  Vervolgens plant hij de andere machine en wel zo dat de bottleneck-machine "op zijn wenken wordt bediend", dat houdt in dat hij zorgt dat het plan voor de bottleneck-machine zo veel als mogelijk kan worden uitgevoerd, ook al moet de niet-bottleneck-machine daartoe wat vaker worden omgesteld (deelorders).

Afhankelijk van het pakket orders blijkt nu eens planner 1 en dan weer planner 2 als beste uit de bus te komen.

Framework frames

In this part of the appendix the frames that form the model described in chapter 6 will be presented. This description will contain the following information:

- The meaning of the frame.
- The attributes (properties) of the frame.
- The relations of the frame.

---

frame: **ORDER**

meaning: represents an amount of products ordered by a customer.

attributes: **- product type.**
- quantity.

relations: **ACTIVITY, PRODUCT TYPE**

---

frame: **PRODUCT-TYPE**

meaning: represents a type of object manufactured by the workshop.

attributes: **- name of the product type.**
- operations.
- route.

relations: **ORDER, OPERATION**

---

frame: **RESOURCE**

meaning: represents the work centers on which the operation are performed.

attributes: **- capacity.**
- operator.
- operation types.

relations: **PLAN, OPERATION TYPE**

---

frame: **OPERATION TYPE**

meaning: represents generic types of operations: i.e. describes what possible operations can be performed by this workshop

attributes: - **resource (list)**

relations: **OPERATION, RESOURCE**

---

frame: **OPERATION**

meaning: represents a single transformation process that is part of the process to manufacture one product.

attributes: - **product type.**
- **(resource, duration)...**

relations: **PRODUCT TYPE, OPERATION TYPE**

---

frame: **ACTIVITY**

meaning: represents a single transformation process that is part of the process to manufacture the products belonging to an order

attributes: - **order**
- **resource**
- **operation**
- **duration**
- **successor / predecessor.**

relations: **OPERATION, PLAN, ORDER**

---

frame: **PLAN**

meaning: represents the GANTT-chart of the resources with the planned activities.

attributes: - **GANTT chart.**

relations: **RESOURCE, ACTIVITY**

FrameTalk

implementation details and listing

FrameTalk is a frame-based shell, created in SMALLTALK/V, a SMALLTALK-80 implementation for IBM PC's and compatibles. In this case it is developed on a Genisys 'the Challenger': an IBM PC/AT clone.
An important reason for the choice of SMALLTALK/V was its the PROLOG extension. A PROLOG interpreter, defined within SMALLTALK/V, aids in the creation of a hybrid shell: Demons represented as PROLOG clauses resemble rules.
The implementation of FrameTalk proved to be a more time-consuming task than was to be expected. For this reason two simplifications had to be introduced to be able to complete the construction of FrameTalk in time:

- The user-interface of FrameTalk is rudimentary: only basic elements are implemented. In this form it is difficult for others to use it.
- Until now it has not been able to implement the demons using PROLOG. Although SMALLTALK/V is delivered with a PROLOG extension, due to the limited time the demons are carried out as SMALLTALK methods. These methods do not resemble rules. As the rules, needed for the CQM problem are practically algorithmic it was easy to implement these as SMALLTALK methods.

Prior to the presentation of FrameTalk, first an explanation of the terminology will be given. Both FrameTalk as well as SMALLTALK-80 have classes, however, these classes are identical. For the sake of clearness class names of SMALLTALK-80 will begin with a capital and are enclosed in quotes.

Programming in SMALLTALK-80 generally consist of the definition of classes and corresponding methods. In FrameTalk two types of frames are distinguished. Hence, two classes are defined within SMALLTALK/V defining the two types of frames: 'FrameClass' and 'Frame', subclass of 'FrameClass'. Based of the class definition the details of FrameTalk will be discussed. The definitions are as follows:

```
Object subclass: #FrameClass
  instanceVariableNames:
    'name class slots '
  classVariableNames: ''
  poolDictionaries:
    'Frames '

FrameClass subclass: #Frame
  instanceVariableNames:
    'frame '
  classVariableNames: ''
  poolDictionaries:
    'Frames '
```

The instance variables form the private memory of each instance of the classes: 'FrameClass'(i.e. the frame classes) and 'Frame' (i.e. the frame instances). As 'Frame' is a subclass of 'FrameClass' it also inherits the instance variables. The instance variables represent the following:

- **'name'** contains a string representing the name of the frame.
- **'class'** contains a pointer to the frameclass the frame is related to: its superclass. A frame can have only one superclass.
- The variable **'slots'** is the container of all slots. It is a dictionary where the keys are the names of the slots. The matching values are also dictionaries: the keys are the facets and the values are the values of that slot.
- the instance variable **'frame'** contains a set of pointers to other frames a frame instance is related to. This variable is takes care of the inheritance of values of slots from other frame instances.

Each value belonging to a frame has a facet. This facet indicate what type of value the value is. FrameTalk can distinguish the following facets:

- **'default'**: a value inherited from a superclass.
- **'value'**: values added in another way. The 'value' facet of a frameclass is always empty. Frame classes contain only 'default' values or 'demon' values .
- **'IfNeeded'**: a demon value that points to a methods that is carried out when a value for a slot is needed.
- **'IfChanged'**: a demon value that points to a method that is carried out when a value of a slot is changed.

The methods belonging to 'FrameClass' are the basic dynamic elements of FrameTalk: they are the procedures that access and manipulate the values of slots. The basis of these procedures are the ideas of Winston and Horn [32] as formulated in their book on LISP.
The essence of these methods is how to way they find and/or react to values in slots. Finally a description will be given of the procedure to obtain a value for a slot and what the result of such a value is:

- If no value is present in the slot first the frameclass is consulted in a 'default' value exists. If no such 'default' exist the frameclass is searched for an 'IfNeeded demon', if found the matching method is executed and the resulting value is stored in the target frame.
- If the previous is not successful it will be repeated with the superclass of the frame class, until either a value is found or the top node is reached.
- Finally the related frame instances are searched for a value. The search is a breadth-first search through a fictitious tree of related frames. For each related frame, if no value is found, beside itself also its frame classes are examined.
- If a value is found and stored, the previous procedure is repeated in search for a 'IfChanged demon'.

When a value is found two possible facets may be assigned to it. If a value is inherited from its superclass the facet 'default' is assigned to it. In all other situations it is assigned the facet 'value'. A complete listing can be found in the following pages.

```
Object subclass: #FrameClass
  instanceVariableNames:
    'name class slots '
  classVariableNames: ''
  poolDictionaries:
    'Frames ' !

!FrameClass class methods !

create: aString class: aFrame slots: aSetofSlots
    "creates a frameclass with slots.
    adds it to the set FrameClasses"

    (aString exists) isNil
                     ifTrue: [ FRAMECLASSES add:( (self new) name: aString
                                                              class: (aFrame frame)
                                                              slots: aSetofSlots)].
    ^aString!

frames
    "answer the set of frameclasses"

    ^FRAMECLASSES collect: [:frame | frame name]!

ppFrames
    "pretty prints all frames to the file frames.pp"
    |output|
    output := Disk newFile: 'frames.pp'.

    output nextPutAll: 'FRAMECLASSES';cr.
    FrameClass frames do: [:frame|(frame frame) printFrame: output].
    output cr;nextPutAll: 'FRAMES';cr.
    Frame frames do: [:frame|(frame frame) printFrame: output].
    output close.
    ^output! !


!FrameClass methods !

actCreateDemon: aSymbol for: anObject
    "ifAdded demon that creates the activities which form part of the
    production process"
    |list obname predecessor actname|

    list := (anObject find: 'operationSequence') asArrayOfSubstrings.
    obname := anObject name.
    predecessor := 'store'.

    list do:
        [:operation| actname := (obname,operation) asString.
                     Frame create: actname
                           class: ('Activity')
                           frame: obname.self halt.
                     actname frame slot: 'operation'
                          value: operation;
                          slot: 'order'
                          value: obname;
                          slot: 'predecessor'
                          value: predecessor.
```

```
                        predecessor := actname].
    ^'(created activities)'!

addSlot: aString
    "add an empty slot to a Frame "

    slots at: aString
          put: Dictionary new.
    ^aString!

askDemon: aString for: anObject
    "prompt the user for a value of aslot(aString) for self
    the result is ALLWAYS a string"

    ^Prompter prompt:('What is the value of slot "',aString,'" for Frame "',(anObject name),'"')
              default:''!

compile
    "find all information for a frame"

    self slots keys do: [:slot|self find: slot].
    self printFrame.!

find: aString
    "searches beside the class tree also the frame tree"
    |aValue |

   aValue := self slot: aString for: self.
   (Inherits := FrameCollection new) addAllLast: self frame .

  ^aValue notNil
          ifTrue: [aValue]
          ifFalse: [self slot: aString
                        value: ((self inheritsFrom) find: aString
                                                    for: self)].!

find: aString for: anObject
    "answers the value of aString. searches the class tree"
    |aValue |

    aValue := self slot: aString for: anObject.

    ^aValue notNil
            ifTrue: [aValue]
            ifFalse: [(self inheritsFrom) find: aString
                                          for: anObject].!

frame
    "stops inheritance"
    ^nil!

frameClass
    "answers the frame that is the
    superclass of self"

    ^class!

ifChanged: aString for: anObject
    "follow the inheritance chain in search for ifAdded demons
    to execute"
    |slotdict selector|

    slotdict := ((self slots) at: aString
                              ifAbsent: []).
    slotdict isNil
        ifFalse: [selector := slotdict at: 'IfChanged'
                                       ifAbsent: []]
        ifTrue: [selector := nil].

    selector notNil
        ifTrue: [self perform: selector with: aString
                                        with: anObject]
        ifFalse: [(self inheritsFrom) ifChanged: aString
                                  for: anObject].
    ^self!

inheritsFrom
    " follows the inheritance: first follow the chain
    through classes, then the chain through frames, and so on"
    |temp |
    temp := self frameClass.

    ^temp isNil
        ifTrue: [Inherits frame]
        ifFalse: [temp].!

multiplyDemon:aSymbol for: anObject
    "computes the duration of an activity by multiplying the number of products
    with the duration of the operation"
    |t1 t2|

    t1 := (((anObject find: 'operation')frame) find: 'duration') asInteger.
    t2 := (((anObject find: 'order') frame) find: 'quantity') asInteger.

    ^(t1 * t2) printString.!

name
    "answer the name of the object"

    ^name!

name: aString class: aFrame slots: aSet
    "bind the InstanceVariables name, class and slots"

    name := aString.
    class := aFrame .
    slots := (Dictionary new).

    aSet do: [:slot | slots at: slot
                            put: (Dictionary new)].!

opSequenceDemon: aSymbol for: anObject
    "asks for the sequence of operation for a certain productType
    ifNeeded demon for ProductType"
    |aString temp|

    aString := Prompter prompt: ('What is the operation sequence for "',(anObject name))
              default:''.
    ^aString asArrayOfSubstrings
            .!
```

```
perform: aString with: aValue
    " Implementation of perform that tests if aString is not nil
    else returns nil"

    ^aString isNil
        ifFalse: [self perform: (aString asSymbol) withArguments: (Array with: aValue)]
        ifTrue: []!

perform: aString with: aValue1 with: aValue2
    " Implementation of perform that tests if aString is not nil
    else returns nil"

    ^aString isNil
        ifFalse: [self perform: (aString asSymbol) withArguments: (Array with: aValue1
                                                                   with: aValue2)]
    ifTrue: []!

printFrame
    "pretty prints a frame"
    |dict slotlist facets|


    Transcript nextPutAll: '-------------------------------------';cr;
         nextPutAll: 'name =>';tab;
         nextPutAll: (self name);cr;cr;
         nextPutAll: 'superclass =>';tab.
    self frameClass notNil
              ifTrue: [Transcript nextPutAll: (self frameClass name);cr;cr]
              ifFalse: [].
         Transcript nextPutAll: 'slot            facet       value';cr;cr.

    slotlist := self slots keys.
    slotlist do: [:key|Transcript nextPutAll: key.
                       dict := (self slots) at: key.
                       facets := dict keys.
                       facets do: [:facet|Transcript tab;tab;nextPutAll: facet;
                                        tab;nextPutAll: (dict at: facet);cr].
                       Transcript cr;cr].
    Transcript nextPutAll: '-------------------------------------';cr.!

printFrame: aStream
    "pretty prints a frame"
    |dict slotlist facets|


    aStream nextPutAll: '-------------------------------------';cr;
         nextPutAll: 'name =>';tab;
         nextPutAll: (self name);cr;cr;
         nextPutAll: 'superclass =>';tab.
    self frameClass notNil
              ifTrue: [aStream nextPutAll: (self frameClass name);cr;cr]
              ifFalse: [].
         aStream nextPutAll: 'slot            facet       value';cr;cr.

    slotlist := self slots keys.
    slotlist do: [:key|aStream nextPutAll: key.
                       dict := (self slots) at: key.
                       facets := dict keys.
                       facets do: [:facet|aStream tab;tab;nextPutAll: facet;
```

```
                                                                      tab;nextPutAll: (dict at: facet);cr].
                                        aStream cr;cr].
     aStream nextPutAll: '------------------------------------';cr.!

reasonDemon: aString for: anObject
     "heuristic that constructs a sequence"
     |l acta actb ord sa sb resa resb|
     l := ((Frame allInstances) collect: [:frame| ((frame frameClass) = ('Activity' frame)
                                                         and: [(frame find: 'week') = (anObject find: 'week')])
                                                         ifTrue: [frame]]) asSet.

     sa := (l collect: [:frame | (frame find: 'resource') = 'A' ifTrue: [frame]])
              asSortedCollection: [:a :b | ((a find: 'duration') asInteger) <= ((b find: 'duration') asInteger)].

     sb := (l collect: [:frame | (frame find: 'resource') = 'B' ifTrue: [frame]])
              asSortedCollection: [:a :b |((a find: 'duration') asInteger) <= ((b find: 'duration') asInteger)].

     resa := OrderedCollection new.
     resb := OrderedCollection new.

     [sa size >=1]
     whileTrue:
          [acta := sa removeFirst.
           ord := acta find: 'order'.
           actb := sb detect: [:act |( act find: 'order') = ord].
           resa add: (OrderedCollection with: acta with: actb).
           sb remove: actb.
           sb size >=0
              ifTrue:
                   [actb := sb removeFirst.
                    ord := actb find: 'order'.
                    acta := sa detect: [:act | (act find: 'order') = ord].
                    resb add: (OrderedCollection with: acta with: actb).
                    sa remove: acta]].
       [resb size >0]
          whileTrue: [resa addLast: (resb removeLast)].
     ^resa!

removeFrame
     " removes a frame of class aClass from the list of frames 'Frames'"

     ^FRAMECLASSES remove: self.!

removeSlot: aString
     " remove the slot aString from frame self"

     (self slots) removeKey: aString.
     ^self slots.!

resPlanDemon: aString for: anObject
     "defines the plans for the different resources"
     |l w alist blist act1 act2 tas tae tbs tbe starttimeA starttimeB aName bName|

     l := (anObject find: 'sequence').
     w := anObject find: 'week'.
     alist := Set new.
     blist := Set new.
     aName := ('Aplan',w).
     bName := ('Bplan',w).
     Frame create: aName
```

```
            class: 'ResourcePlan';
            create: bName
            class: 'ResourcePlan'.

      starttimeA :=0.
      starttimeB :=0.

      1 do:
        [:actlist| act1 := actlist at: 1.
                   act2 := actlist at: 2.
                   tas := starttimeA.
                   tae := tas + ((act1 find: 'duration') asInteger).
                   starttimeA := tae.
                   tbs := starttimeB max: tae.
                   tbe := tbs + ((act2 find: 'duration') asInteger).
                   starttimeB := tbe.
                   act1 := (OrderedCollection with: (act1 name))
                                       add: tas;
                                       add: tae.
                   alist add: act1.
                   act2 := (OrderedCollection with: (act2 name))
                                       add: tbs;
                                       add: tbe.
                   blist add: act2].

      aName frame slot: 'actlist'
                       value: alist.
      bName frame slot: 'actlist'
                       value: blist.!

slot: aString
      "answer the value of a slot. Does not add values calculated by ifNeeded
      to the slot. Only for Frameclasses"
      |aSlot anObject|

      anObject := self.
      aSlot := (self slots) at: aString
                           ifAbsent: [].

      ^aSlot at: 'value'
             ifAbsent:
             [aSlot at: 'default'
                    ifAbsent:
                    [self perform: (aSlot at: 'IfNeeded'
                                          ifAbsent: [])
                          with: aString
                          with: anObject]]!

slot: aString1 facet: aString2 value: aValue
      " adds a value."
      |dict|
      dict := (self slots) at: aString1.
      ^dict at: aString2
            put: aValue.!

slot: aString for: anObject
      "answer the value of a slot. Does not add values calculated by ifNeeded
      to the slot. Only for Frameclasses"
      |aSlot|
    aSlot := (self slots) at: aString
                          ifAbsent: [].

    ^aSlot notNil
    ifTrue: [aSlot at: 'value'
                   ifAbsent:
                   [aSlot at: 'default'
                          ifAbsent:
                          [self perform: (aSlot at: 'IfNeeded'
                                                ifAbsent: [])
                                with: aString
                                with: anObject]]]
    ifFalse: []!

slot: aString value: aValue
    "adds aValue to slot aString. executes ifChanged demons"
    |aSlot t2|

    aSlot := ((self slots) at: aString
                           ifAbsent: []).

    aSlot notNil
        ifTrue: [aSlot  at: 'value'
                        put: aValue.

    (Inherits := FrameCollection new) addAllLast: self frame .
    t2 := Inherits.
    self ifChanged: aString for: self].
    ^aValue!

slots
    "answer the name of the slots"

    ^slots! !
```

```
FrameClass subclass: #Frame
  instanceVariableNames:
    'frame '
  classVariableNames: ''
  poolDictionaries:
    'Frames ' !

!Frame class methods !

create: aString class: aFrame
    "create a frame called aString of class aFrame.
    Add slots to the frame given by the class description"

    (aString exists) isNil
                    ifTrue: [FRAMES add:( (self new) name: aString
                                                     class: (aFrame frame))].
    ^(aString,' of class: ', aFrame)!

create: aString class: aFrame frame: aStringFrame
    "create a frame called aString of class aFrame.
    Add slots to the frame given by the class description"

    (aString exists) isNil
                    ifTrue: [FRAMES add:( (self new) name: aString
                                                     class: (aFrame frame)
                                                     frame: (aStringFrame frame))].
    ^(aString,' of class: ', aFrame)!

frames
    "answer the set of frameclasses"

    ^FRAMES collect: [:frame | frame name]!

framesOfClass: aString
    "answer the set of frameclasses"

    ^FRAMES collect: [:frame|frame frameClass name = aString
                              ifTrue: [frame name]]! !


!Frame methods !

frame
    "answers the set of related frames"

    ^frame!

frame: aString
    "bind the related frame to the variable 'frame' "

    frame addLast: (aString frame).
    ^aString!

name: aString class: aFrame
    "instantiate the vars name, class and slots"

    frame := FrameCollection new.

    self name: aString
```

```
          class: aFrame
          slots: (aFrame slots) keys!

name: aString class: aFrame frame: aString2
    "instantiate the vars name, class, frame and slots"

    frame := FrameCollection with: (aString2 frame).

    self name: aString
         class: aFrame
         slots: (aFrame slots) keys!

printFrame
    "pretty prints a frame"
    |dict slotlist facets|


    Transcript nextPutAll: '------------------------------------';cr;
         nextPutAll: 'name =>';tab;
         nextPutAll: (self name);cr;cr;
         nextPutAll: 'superclass =>';tab;
         nextPutAll: ((self frameClass) name);cr;cr;
         nextPutAll: 'frames =>';tab.
         self frame notNil
            ifTrue: [self frame do: [:frame|Transcript nextPutAll: ((frame name),', ')]]
            ifFalse: [].
         Transcript cr;cr;
         nextPutAll: 'slot            facet        value';cr;cr.

    slotlist := self slots keys.
    slotlist do: [:key|Transcript nextPutAll: key.
                       dict := (self slots) at: key.
                       facets := dict keys.
                       facets do: [:facet|Transcript tab;tab;
                                          nextPutAll: facet;tab;
                                          nextPutAll:((dict at: facet) printString); cr].
                                  Transcript cr;cr].
     Transcript nextPutAll: '------------------------------------';cr.!

printFrame: aStream
    "pretty prints a frame"
    |dict slotlist facets|


    aStream nextPutAll: '------------------------------------';cr;
         nextPutAll: 'name =>';tab;
         nextPutAll: (self name);cr;cr;
         nextPutAll: 'superclass =>';tab;
         nextPutAll: ((self frameClass) name);cr;cr;
         nextPutAll: 'frames =>';tab.
         self frame notNil
            ifTrue: [self frame do: [:frame|aStream nextPutAll: ((frame name),', ')]]
            ifFalse: [].
         aStream cr;cr;
         nextPutAll: 'slot            facet        value';cr;cr.

    slotlist := self slots keys.
    slotlist do: [:key|aStream nextPutAll: key.
                       dict := (self slots) at: key.
                       facets := dict keys.
```

```
                        facets do: [:facet|aStream tab;tab;
                                                 nextPutAll: facet;tab;
                                                 nextPutAll: ((dict at: facet) printString); cr].
                        aStream cr;cr].
     aStream nextPutAll: '-------------------------------------';cr.!

removeFrame
    " removes a frame of class aClass from the list of frames 'Frames'"

    FRAMES remove: self.!

removeSlot: aString
    " remove the slot aString from frame self"

    (self slots) removeKey: aString.
    ^self slots.!

slot: aString for: anObject
    "answer the value of a slot.Adds values calculated by ifNeede
    to the slot."
    |aSlot aValue|

    aSlot := (self slots) at: aString
                          ifAbsent: [].
    ^aSlot notNil
     ifTrue: [^aSlot at: 'value'
                 ifAbsent:
                     [aSlot at: 'default'
                            ifAbsent:
                            [aValue := self perform: (aSlot at: 'IfNeeded'
                                                        ifAbsent: [])
                                              with:aString
                                              with: anObject.
                             ^aValue notNil
                             ifTrue: [aSlot at: aString
                                              put: aValue]
                             ifFalse: []]]]
    ifFalse: []! !
```

```
OrderedCollection variableSubclass: #FrameCollection
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!FrameCollection class methods ! !


!FrameCollection methods !

find: aString for: anObject
    "searches a set of frames for values of the slot: aString"
    |aFrame aNewSet aValue|

    self size > 0
    ifTrue: [aFrame := self removeFirst.
             aNewSet := self.
             ^aFrame notNil
                ifTrue: [aValue := aFrame find: aString]
                ifFalse: [aNewSet find: aString for: anObject]]
    ifFalse: [aValue := nil].

    ^aValue notNil
           ifTrue: [aValue]
           ifFalse: [aNewSet find: aString for: anObject]!

frame
    "answers the first element of a FrameCollection: a Frame"
    | temp t2|
    self size = 0
        ifFalse: [temp := self removeFirst.
                    (temp frame) notNil
                    ifTrue: [Inherits addAllLast: (temp frame copy)]]
        ifTrue: [^nil].

    ^temp!

ifChanged: aString for: anObject
    "searches a set of frames for values of the slot: aString"
    |aFrame aNewSet aValue|

    self size > 0
    ifTrue: [aFrame := self removeFirst.
             aNewSet := self .
             ^aFrame notNil
                ifTrue: [aValue := aFrame find: aString]
                ifFalse: [aNewSet ifChanged: aString for: anObject]]
    ifFalse: [aValue := nil].

    ^aValue notNil
           ifTrue: [aValue]
           ifFalse: [aNewSet ifChanged: aString for: anObject]! !
```

```
FRAMECLASSES
-------------------------------------
name =>   ProductType

superclass =>  Workshop

slot                facet     value

operationSequence   IfNeeded  askDemon:for:


-------------------------------------
-------------------------------------
name =>   Activity

superclass =>  Workshop

slot                facet     value

predecessor

week

operation

order

duration       IfNeeded  multiplyDemon:for:


-------------------------------------
-------------------------------------
name =>   ResourcePlan

superclass =>  WeekPlan

slot            facet       value

actlist

resource

-------------------------------------
-------------------------------------
name =>   WeekPlan

superclass =>  Plan

slot            facet       value

sequence       IfChanged resPlanDemon:for:
               IfNeeded  reasonDemon:for:


week           IfNeeded  askDemon:for:


-------------------------------------
-------------------------------------
name =>   Operation
```

```
superclass =>  Workshop

slot            facet       value

duration       IfNeeded  askDemon:for:


resource       IfNeeded  askDemon:for:


-------------------------------------
-------------------------------------
name =>   Order

superclass =>  Workshop

slot            facet       value

productType         IfChanged actCreateDemon:for:
                    IfNeeded  askDemon:for:


quantity            IfNeeded  askDemon:for:


week                IfNeeded  askDemon:for:


-------------------------------------
-------------------------------------
name =>   Plan

superclass =>  Workshop

slot            facet       value

activity

resource

-------------------------------------
-------------------------------------
name =>   Resource

superclass =>  Workshop

slot            facet       value

capacity       IfNeeded  askDemon:for:


-------------------------------------
-------------------------------------
name =>   Workshop

superclass =>  slot             facet       value

-------------------------------------
```

```
FRAMES
-------------------------------------
name =>   Order3

superclass =>  Order

frames => P4,

slot              facet       value

productType           value     'P4'


quantity              value     '100'


week                  value     '1'


-------------------------------------
-------------------------------------
name =>   Order2

superclass =>  Order

frames => P2,

slot              facet       value

productType           value     'P2'


quantity              value     '75'


week                  value     '1'


-------------------------------------
-------------------------------------
name =>   P8

superclass =>  ProductType

frames =>

slot              facet       value

operationSequence         value     'Opa6 Opb4'


-------------------------------------
-------------------------------------
name =>   Order6Opb4

superclass =>  Activity

frames => Order6, Opb4,

slot              facet       value
```

```
predecessor         value     'Order6Opal'

week           value     '1'

operation      value     'Opb4'

order          value     'Order6'

duration       value     '450'

--------------------------------------
--------------------------------------
name =>   Order3Opa4

superclass =>  Activity

frames => Order3, Opa4,

slot             facet       value

predecessor         value     'store'

week           value     '1'

operation           value     'Opa4'

order               value     'Order3'

duration            value     '500'

--------------------------------------
--------------------------------------
name =>   Bplan1

superclass =>  ResourcePlan

frames =>

slot             facet       value

resource

actlist        value     Set(OrderedCollection('Order5Opb4' 1725
2025 ) OrderedCollection('Order2Opb2' 600 975 )
OrderedCollection('Order4Opb4' 300 600 )
OrderedCollection('Order6Opb4' 975 1425 )
OrderedCollection('Order1Opb2' 100 225 )
OrderedCollection('Order3Opb4' 1425 1725 ) )


-------------------------------------
-------------------------------------
name =>   Opa6

superclass =>  Operation

frames =>

slot             facet       value

duration       value     '6'


resource       value     'A'


-------------------------------------
-------------------------------------
name =>   P3

superclass =>  ProductType

frames =>

slot             facet       value

operationSequence        value      'Opa3 Opb3'


-------------------------------------
-------------------------------------
name =>   Order2Opa2

superclass =>  Activity

frames => Order2, Opa2,

slot             facet       value

predecessor          value     'store'


week      value     '1'


operation      value     'Opa2'
```

```
order           value     'Order2'

duration        value     '300'

-------------------------------------
-------------------------------------
name =>   Order1Opa2

superclass =>  Activity

frames => Order1, Opa2,

slot              facet       value

predecessor          value     'store'

week      value     '1'

operation       value     'Opa2'

order           value     'Order1'

duration        value     '100'

-------------------------------------
-------------------------------------
name =>   Opa2

superclass =>  Operation

frames =>

slot              facet       value

duration        value     '4'

resource        value     'A'

-------------------------------------
-------------------------------------
name =>   Opa5

superclass =>  Operation
```

```
frames =>

slot             facet        value

duration       value     '1'


resource       value     'A'


-------------------------------------
-------------------------------------
name =>   Order4

superclass =>  Order

frames => P10,

slot             facet        value

productType          value     'P10'


quantity       value     '100'


week      value     '1'


-------------------------------------
-------------------------------------
name =>   P7

superclass =>  ProductType

frames =>

slot             facet        value

operationSequence        value     'Opa5 Opb3'


-------------------------------------
-------------------------------------
name =>   Order6

superclass =>  Order

frames => Opa1, Order6,

slot             facet        value

productType          value     'P10'
```

```
quantity        value      '150'

week        value      '1'

-------------------------------------
-------------------------------------
name =>    Opal

superclass =>  Operation

frames =>

slot               facet        value

duration        value      '2'

resource        value      'A'

-------------------------------------
-------------------------------------
name =>    Opa4

superclass =>  Operation

frames =>

slot               facet        value

duration        value      '5'

resource        value      'A'

-------------------------------------
-------------------------------------
name =>    Order5

superclass =>  Order

frames =>

slot               facet        value

productType          value      'P10'

quantity        value      '100'
```

```
week      value     '1'


-------------------------------------
-------------------------------------
name =>   P2

superclass =>  ProductType

frames =>

slot             facet       value

operationSequence        value     'Opa2 Opb2'


-------------------------------------
-------------------------------------
name =>   Opa3

superclass =>  Operation

frames =>

slot             facet       value

duration       value     '3'


resource       value     'A'


-------------------------------------
-------------------------------------
name =>   Aplan1

superclass =>  ResourcePlan

frames =>

slot             facet       value

resource

actlist        value     Set(OrderedCollection('Order2Opa2' 300
600 ) OrderedCollection('Order4Opa1' 100 300 )
OrderedCollection('Order6Opa1' 600 900 )
OrderedCollection('Order1Opa2' 0 100 )
OrderedCollection('Order3Opa4' 900 1400 )
OrderedCollection('Order5Opa1' 1400 1600 ) )


-------------------------------------
```

```
-------------------------------------
name =>   Order1

superclass =>  Order

frames => P2,

slot              facet        value

productType          value     'P2'


quantity       value     '25'


week      value     '1'


-------------------------------------
-------------------------------------
name =>   Opb4

superclass =>  Operation

frames =>

slot              facet        value

duration       value     '3'


resource       value     'B'


-------------------------------------
-------------------------------------
name =>   WeekPlan1

superclass =>  WeekPlan

frames =>

slot              facet        value

sequence       value     OrderedCollection(OrderedCollection(a
Frame a Frame ) OrderedCollection(a Frame a Frame )
OrderedCollection(a Frame a Frame ) OrderedCollection(a Frame a
Frame ) OrderedCollection(a Frame a Frame ) OrderedCollection(a
Frame a Frame ) )


week      value     '1'
```

```
-------------------------------------
-------------------------------------
name =>    P6

superclass =>  ProductType

frames =>

slot              facet        value

operationSequence        value      'Opal Opb3'


-------------------------------------
-------------------------------------
name =>    Order3Opb4

superclass =>  Activity

frames => Order3, Opb4,

slot              facet        value

predecessor          value      'Order3Opa4'


week        value      '1'


operation       value      'Opb4'


order           value      'Order3'


duration        value      '300'


-------------------------------------
-------------------------------------
name =>    Order5Opb4

superclass =>  Activity

frames => Order5, Opb4,

slot              facet        value

predecessor          value      'Order5Opal'


week        value      '1'
```

```
operation      value     'Opb4'

order          value     'Order5'

duration       value     '300'

------------------------------------
------------------------------------
name =>   Order4Opb4

superclass =>  Activity

frames => Order4, Opb4,

slot              facet       value

predecessor           value      'Order4Opa1'

week       value      '1'

operation      value     'Opb4'

order          value     'Order4'

duration       value     '300'

------------------------------------
------------------------------------
name =>   P1

superclass =>  ProductType

frames =>

slot              facet       value

operationSequence        value     'Opa1 Opb1'

------------------------------------
------------------------------------
name =>   Order2Opb2

superclass =>  Activity

frames => Order2, Opb2,
slot                facet        value

predecessor            value      'Order2Opa2'


week       value      '1'


operation       value      'Opb2'


order           value      'Order2'


duration        value      '375'

-------------------------------------
-------------------------------------
name =>    Order1Opb2

superclass =>  Activity

frames => Order1, Opb2,

slot                facet        value

predecessor            value      'Order1Opa2'


week       value      '1'


operation       value      'Opb2'


order           value      'Order1'


duration        value      '125'

-------------------------------------
-------------------------------------
name =>    Opb3

superclass =>  Operation

frames =>

slot                facet        value

duration        value      '2'
```

```
resource        value     'B'


-------------------------------------
-------------------------------------
name =>    P5

superclass =>  ProductType

frames =>

slot            facet       value

operationSequence        value     'Opa3 Opb2'


-------------------------------------
-------------------------------------
name =>    Order4Opal

superclass =>  Activity

frames => Order4, Opal,

slot            facet       value

predecessor          value     'store'


week      value     '1'


operation      value     'Opal'


order          value     'Order4'


duration       value     '200'


-------------------------------------
-------------------------------------
name =>    Order6Opal

superclass =>  Activity

frames => Opal, Order6,

slot            facet       value

predecessor          value     'store'
```

```
week        value      '1'

operation        value      'Opa1'

order            value      'Order6'

duration         value      '300'

-------------------------------------
-------------------------------------
name =>    Order5Opa1

superclass =>  Activity

frames => Order5, Opa1,

slot              facet        value

predecessor          value     'store'

week        value      '1'

operation        value      'Opa1'

order            value      'Order5'

duration         value      '200'

-------------------------------------
-------------------------------------
name =>    Opb2

superclass =>  Operation

frames =>

slot              facet        value

duration         value      '5'

resource         value      'B'
```

```
-------------------------------------
-------------------------------------
name =>   P9

superclass =>  ProductType

frames =>

slot            facet       value

operationSequence        value     'Opa2 Opb3'


-------------------------------------
-------------------------------------
name =>   Opb1

superclass =>  Operation

frames =>

slot            facet       value

duration       value     '1'


resource       value     'B'


-------------------------------------
-------------------------------------
name =>   P10

superclass =>  ProductType

frames =>

slot            facet       value

operationSequence        value     'Opa1 Opb4'


-------------------------------------
-------------------------------------
name =>   P4

superclass =>  ProductType

frames =>

slot            facet       value

operationSequence        value     'Opa4 Opb4'
```