1 Knowledge-Based System Techniques in the Design, Implementation, and Validation of Resource Scheduling on the Shop Floor of Manufacturing Systems

Rahul De′
Rider University

1.1 Introduction

In this chapter we will consider the design, implementation, and validation issues of building knowledge-based systems for scheduling resources on the shop floor. Such systems are typically defined as decision support tools that support the human operators whose task it is to plan, schedule, execute, and control shop floor operations. The issues of decision support are reflected in the design of these systems, either directly supporting the cognitive goals of the schedulers or indirectly providing task-specific information germane to the decision situation. In either case, there appear to be a common set of data and processing needs that can be delineated as essential requirements of such systems.

Knowledge-based systems are used in diverse scheduling applications that include scheduling space shuttle repair, scheduling space telescope observations, scheduling shipbuilding, assigning personnel for projects, scheduling operating theaters at hospitals, retail logistics scheduling, military operations planning, etc. A large number of applications deal with allocating resources on the shop floor. Much research has focused on this latter problem given the tremendous complexity of the domain and the relative lack of traditional OR techniques that can be effectively applied. This chapters explores the problems of shop floor scheduling.

Software engineers choose to sharply distinguish between the design and implementation of systems. The former implies the logical and physical specifications of data structures and process (or objects) for the system, and the latter the realization of the design specifications in coded modules that are installed at the facility. Most methodologies, proposing normative system building methods, insist on keeping the two activities in distinct phases. The construction of knowledge-based systems, as is evident from the literature, follows an approach of iterating the design-implementation phases, where each iteration involves the construction of another layer of the system. In describing some of the systems in this paper, the design rationale will be examined first, followed by the implementation choices considered.

Validation of a knowledge-based system entails measuring its performance within its working environment. This is usually a difficult task in the domain of scheduling because it is difficult to obtain standards against which to measure the performance. Given that it is virtually impossible to obtain optimal solutions for most real scheduling situations, researchers have to rely on measuring changes in certain process characteristics specific to the particular factory. For example, in steel making, the parameter observed by a team of researchers to measure the performance of the scheduling system was the amount of reduction in the wait time of the molten metal (before pouring into casts). Other approaches to validation include measuring the quality of the schedules against those proposed by a human scheduler or measuring the improvement in the manufacturing processes via simulations. This chapter explores the different approaches.

1.2 Design of Knowledge-Based Scheduling Systems

The design of knowledge-based scheduling systems has been approached from many perspectives. The complex nature of the scheduling problem, and that of the domain, have ensured that researchers have had to continually seek newer and more innovative designs. A core theory for the representation of such designs has emerged and is used as a basis for many systems. However, the fact remains that there is much diversity in the final systems designed. This diversity includes variations in the manner of searching for solutions, the manner in which the system modules are linked up the kind of support provided to the scheduler the manner in which knowledge is extracted for the system, and the adaptability of the system to different tasks.

The knowledge in knowledge-based scheduling systems pertains to the rules, procedures, and heuristics used by the schedulers or workers to enable smooth functioning of the factory. The knowledge is usually obtained or extracted from the scheduler and is then included within the design of the system. It is worth pointing out that knowledge-based systems in manufacturing do not strictly follow the approach of expert systems where the expert's rules are the sole basis of reasoning within the system. To construct such systems, considerable effort is spent in extracting a set of rules that is as near complete as the expert's knowledge of the domain. Once available, it is encoded in a declarative form and used by the reasoning mechanism (the pattern matcher) of the system. The declarative rules extracted from the expert are a part of the total reasoning capabilities of the system. The knowledge base also contains knowledge about the factory, the processes, and other details that may be represented in many ways, not only in the declarative manner.

A number of reasons are provided by researchers as to why just the rules obtained from an expert scheduler are inadequate for constructing effective scheduling systems. First, it is not always possible for schedulers to articulate, and for knowledge engineers to capture, all of the complexities of the work they do (Kerr, 1992; May and Vargas, 1996). Their work involves assessing the state of materials and machines on the shop floor, assigning or reassigning duties to work centers, locating the source of problems in work flow, responding to the directives of higher management regarding manpower or work-in-process levels, not all of which can be captured as rules. Secondly, the domain of the shop floor is complex and consists of a number of interconnected parts for which the scheduler is not directly responsible and consequently does not control. Third, owing to political and cultural reasons, schedulers were not always willing to provide complete information to the system builders (McKay et al., 1995).

A core idea that many scheduling systems use is that of a constraint-based representation. Constraints are the restrictions of process times, process routings, or machine availability that are imposed on the resources of the plant along with the restrictions that originate from management directives, engineering concerns, operator preferences, etc. (Fox and Smith, 1984). The scheduler has to respond to all these constraints. Consequently, the task of scheduling revolves centrally around satisfying the constraints. All the constraints that can exist in a shop floor situation have to be represented within the system in order to complete the scheduling task. A constraint-based representation provides a formal method by which constraints can be represented and manipulated to create schedules.

Constraint-based representation forms the basis of many scheduling systems; however, a number of other important design issues that are relevant are:

- Constructive versus repair methods. In the constructive methods of schedule or plan construction, schedules are constructed by incrementally extending partial schedules. In the repair methods, schedules are completed by beginning with a complete, but possibly flawed, set of assignments and then iteratively modifying or repairing them. Algorithms designed for either of these methods rely on the underlying constraint representation to construct or modify the schedule.
- Predictive versus reactive scheduling. Predictive scheduling implies creating resource assignments for future periods of time, whereas reactive scheduling refers to adjusting the schedules to respond to current situations on the shop floor. Most systems have facilities to respond to both types of scheduling situations.
- Distributed scheduling systems versus "monolithic" scheduling systems. In the former, scheduling solutions are provided by a number of relatively independent modules that cooperate in different ways to arrive at a solution. In the latter, the system is designed as a large, comprehensive system consisting of a number of integrated modules. The level of aggregation in the system determines both its implementation strategy as well as the problem solving process that it uses.
- Cooperative problem solving versus independent machine-generated solutions. Cooperative problem solving systems are designed to work actively with the scheduler in supporting his or her goals and tasks. These are distinguished from systems that generate independent solutions with constraint or data input from the scheduler. In the former case, the solution generation proceeds with active input from the scheduler, but not in the latter.
- System knowledge based on one (or two) schedulers versus that based on input from many supervisors/workers on the shop floor. There are many implemented systems based on knowledge obtained from a senior scheduler in a factory. These systems tend to be smaller and focus on scheduling important (bottleneck) resources. Input from a large number of potential users is collected to build a comprehensive scheduling system that includes all activities on the shop floor.
- Inclusion of established operations research algorithms versus creating special implementations of general search algorithms. Many designers have tried to build their systems algorithms from operations research that can be applied at a very fine level of detail. For the situations in which they can be applied, these algorithms ensure that an optimal or near-optimal solution is obtained. However, in most cases, it is difficult to reduce the problem to the level in which such algorithms will work so designers usually end up modifying search algorithms to suit the problem situation.

- Generalized knowledge-based scheduling systems versus those specialized for a domain. Some designers have chosen to create designs that are general in scope and can provide scheduling solutions for a number of different domains. These systems provide knowledge representation and reasoning mechanisms, as in expert system "shells," which can be specialized for a domain. Special systems are designed for a particular domain, to solve the scheduling problems in that domain alone.
- Learning of scheduling heuristics versus static implementations. Some systems have been designed to learn scheduling heuristics and improve on their scheduling behavior. These systems are able to use their experience of arriving at schedules as a basis on which to build further heuristics. Most systems, however, tend to be static in that they use the problem solving methods already built in.
- Use of simulations for providing support. Some systems rely on intelligently simulating the processes on the shop floor to support the scheduler's decision making. The system enables the scheduler to 'visualize' the effect of the decisions over a given time horizon.

The following sections examine the issues highlighted above in greater detail. Examples of various systems reported in the literature are used to discuss the issues.

Constraint Representation

Constraints as a means of representing scheduling knowledge was first explored by Fox and Smith (1983, 1984). Their goal was to create a system to schedule a job shop. Analysis of the scheduling tasks revealed that "…the crux of the scheduling problem is the determination and satisfaction of a large variety of constraints." They found that the scheduler spent 10–20% of his time scheduling or determining assignments, while the balance of his time was spent communicating with other employees to determine additional 'constraints' that would affect the schedule. Thus, they concluded that the constraint-based representation would be most appropriate for building the scheduling system.

Constraints are built into the basic modeling units (schemas), as meta-information for any slot, slot-value or the schema itself. Schemas are frame-based (Minsky, 1975) knowledge representation units consisting of slots and their values, which may relate different schemas in a hierarchical inheritance network. Schemas are used to model the factory details from physical machine descriptions to organizational structures and relations (Fox and Smith, 1984). Schemas are created and updated for order definitions, lot definitions, work area definitions, plant organization, etc. Within the hierarchical organization of schemas, the lowest level consists of primitive concepts of states, objects, and actions, on top of which domain specific concepts are defined and related. Constraints in this hierarchy of schemas may define the duration of some activity, the due dates for some job, the precondition for some state, etc.

Constraints themselves are represented as schemas with certain properties. One property that is defined for all constraints is the relaxation that is possible for that constraint in a case of conflict with another. Constraint boundaries are relaxed, by a specified mechanism, to resolve conflicts. Relaxations are specified by discrete or continuous choice sets that have utility functions associated with them to assign preference amongst the alternatives. Constraints are also characterized by their relative importance or priority order. This measure is used to determine which of conflicting constraints may be relaxed. Another property is the relevance of the constraint which specifies the conditions under which the constraint should be applied. Interactions is another property that specifies the effects of variations of one constraint's values on others that are related to it, and the direction and extent of these variations. The generation of constraints may be dynamic, created during schedule construction, or they may be created along with the instantiation of schemas.

Constructive Approach

The constraint-based representation was used to build a scheduling system called ISIS. ISIS used the constraints in a sophisticated search procedure to construct schedules. The system conducts a hierarchical, constraint-directed search in the space of all possible schedules (Fox and Smith, 1984). Control is passed from the top to the lower levels and communication between the levels occurs via constraints. At each level, the system does three kinds of processing: pre-analysis (where bounds of the current level's search space are determined); search (where the actual assignment solution is sought); and post-analysis (where the results of the search are assessed). If the post-analysis finds the results acceptable, these are coded as constraints to be passed on to the next lower level. In case the post-analysis rejects the results, the search space is altered at the current or previous level and the control is transferred there. The search itself is conducted at four levels where level one selects an order to be scheduled based on its priority and due date, level two does a capacity analysis to determine availability of machines for this order, level three schedules the resources to satisfy the order, and level four finalizes the schedule by reserving the machines for the order. The control is returned to the top and another order is picked for assignment.

Using the same representation scheme, several researchers have tried different search procedures and architectures with which to solve the scheduling problem. Some of these have resulted in fielded systems while others have helped find faster and more efficient ways of solving the scheduling problem.

Iterative Repair

Iterative repair methods of scheduling are based on modifying a complete but possibly flawed set of assignments that are iteratively adjusted to improve the overall schedule. Zweben et al. (1994) describe a system, called GERRY, that is used to schedule operations for space shuttle flights. The system uses iterative repair with a stochastic approach that enables the construction of very good schedules. GERRY uses a general purpose scheduling algorithm that could be deployed in any situation. Its architecture is domain independent.

GERRY uses two important constraints known as the milestone and temporal constraints. Milestones are fixed metric times beyond which a task cannot be moved, whereas temporal constraints show the start-end times of tasks and their relationship to each other. The system never violates temporal constants. Resource constraints are classified as classes or pools, where a class represents a type and a pool represents the entire collection of a type. Capacity constraints restrict overuse or over allocation of resources.

State constraints depict certain attribute values that have to hold for a task to be accomplished. These are domain specific attributes related to the tasks. A data structure called a *history* is used to track the changes in attribute values, what caused the changes, and for what are they required. As the system runs, this data structure is constantly updated or read to maintain control over the changes to the task schedules. Reading and maintaining the data structure constitutes a major activity for the system.

The system also uses preemptive schedules where each task is associated with a calendar of legal work periods when it can be performed. If a task cannot be completed in one period, then this requires breaking it up into a set of subtasks to be allocated to different periods.

The iterative repair method proceeds with a complete but flawed schedule, which is iteratively modified until its quality is found to be satisfactory or until it reaches a time bound (for computation). The quality is measured by a cost function which is a weighted sum of penalties for constraint violations. The weights and penalties are non-zero values identified independently. In each iteration, constraints are repaired. Each repair causes local and global perturbations in the network which are resolved in later iterations. After each iteration, the system recomputes the evaluation (cost) function to compare the new schedule with the old. If the new schedule is better, it is accepted for the next iteration. If the new schedule is more costly than the old, then it is accepted with a certain probability. This technique is called *simulated annealing* and is used to avoid local minima. If the new schedule is rejected, then the old one is used to continue the iterations.

The results from tests performed with the system showed that in most cases the system converged and a solution was found. Even in the cases where the system timed out before finding a solution, the cost of the solution at the end was much lower than the initial cost.

Predictive and Reactive Scheduling

Predictive scheduling was the norm in planning and scheduling systems when computation was slower and more expensive. With some given input conditions, a plan or schedule was generated and shop floor managers had to rely on this output alone. Godin (1978) expressed the need for interactive scheduling systems: "Scheduling problems change so rapidly that the systems are not flexible or sophisticated enough to keep up with them." Systems worked more in the batch than in the interaction mode of operation.

The situation changed rapidly when interactivity was made central to scheduling systems, known as *reactive scheduling systems*. Design of reactive scheduling systems was centered around recognizing conflicts in the schedules arising from changes in the environment and modifying the schedules to resolve the conflicts (Ow et al., 1987). Conflict resolution in this fashion tended to affect the entire schedule, as the changes introduced in one part would spread to other parts. Research on reactive systems focused on devising algorithms that would handle such "ripple effects" efficiently (Szelke and Kerr, 1994).

One idea that remains constant throughout the work on reactive scheduling systems is that the predictive and reactive parts of scheduling cannot really be viewed separately. They are viewed as complementary functions for generating and revising schedules. An example of an implemented scheduling system, where the predictive and reactive parts are fully integrated, is that of the system for scheduling semiconductor wafer fabrication at Intel Corporation (Kempf, 1994).

The semiconductor manufacturing process at Intel is considered to be a linear flow with loops for rework. The process consists of hundreds of steps that take many weeks to complete. The factory consists of hundreds of processing resources, including machines and tools, and the machines exhibit a wide variety of characteristics, including cycle times from minutes to tens of hours, load sizes varying from one wafer to many, and multiple setup switches. In this situation, scheduling consists of assigning lots of wafers to active resources. The predictive part of scheduling produces a set of assignments of lots to specific resources in future times and the reactive part of scheduling attempts to realize this set of assignments in the light of unforeseen events on the shop floor.

Kempf argues that the idea of the predictive-reactive system was unavoidable because the predictive scheduler by itself would have been useless (given the complexity of the scheduling task and the number of unexpected events that occurred) and a reactive scheduler by itself would only find grossly sub optimal solutions. Thus, the predictive scheduler was designed to produce a 'stake in the ground' schedule that would be used until a change occurred and then the reactor would be used to recompute the schedule, as 'tethered to the stake.' Jobs were ordered according to priority and the higher priority ones were assigned by the predictor as fully as possible, leaving the lower priority ones for later assignment. The reactor focused on the changes made to the scheduled jobs.

Distributed Scheduling Systems

Distributed scheduling system architectures are constituted of a number of autonomous or independent modules (also referred to as agents) that act upon different parts of the scheduling problem to provide a comprehensive solution. A central coordinating mechanism controls the behavior of the modules to a certain extent; however, within the overall problem solving process, the modules act autonomously and opportunistically. These architectures are distinguished from monolithic, mostly hierarchic structures whose modules are integrated and act in a defined, structured manner. A number of possible types of distributed architectures have been proposed, some of which are described below.

Ow and Smith (1987) describe the use of multiple knowledge sources in the OPIS 0 scheduling system. In this first version of the OPIS family of systems, the emphasis was on opportunistic problem solving where a set of knowledge sources seek out solution opportunities in a problem situation. Knowledge sources are collections of heuristics that can bear on a particular aspect of the problem. These sources continuously monitor the solution process and wherever an opportunity arises for any one of them to act, they do so and provide their part of the solution to the global solution. The activities of the knowledge sources are controlled through a "blackboard architecture" (Hayes-Roth, 1985).

In OPIS 0, the scheduling problem was broken into two types of sub-problems: order-based and resource-based. In the order-based approach, a schedule would be created for a particular order and then refined, whereas in the resource-based approach, each resource or machine would be assigned according to availability after which the complete schedule would be built up. To manage the application of the knowledge sources, a hierarchical control structure was devised that consisted of manager knowledge sources. These would be responsible for decomposing the problem into two types of sub-problems: applying knowledge sources to them and collecting the partial schedules thus obtained. Depending on the nature of the problem, an order-based, or a resource-based solution, would be tried first.

Later implementations (Ow et al., 1987; Smith, 1987; Ow and Smith, 1987; Smith, 1994) of the OPIS architecture relaxed the rigid control of OPIS 0 and allowed for greater flexibility of applying the knowledge sources. The OPIS systems were tested for various problems and returned reasonable schedules.

The ReDS architecture (Hadavi et al., 1992; Hadavi, 1994) consists of small, independent modules (agents) based on a generic design that are recursively deployed for different purposes. Each module, called a *planning agent*, consists of a "scheduling gene" that has a predictive and a reactive component. The scheduling gene changes its state over time and beginning as a reactive agent, it evolves its predictive abilities. The agents are cooperative and perform tasks given to them by other agents and other inputs to the agents are feedback about their actions, responses from "higher" agents in the hierarchy, and information about the environment. The asynchronous inputs are used by the agents to decide on a course of action: their outputs. Two forms of feedback are provided to the agents: information about their model of the state of the world and about their reasoning. Within the architecture, agents respond to broadcast messages and provide their own inputs for a specific time horizon. There is no central controlling mechanism.

Each planning agent in ReDS consists of various modules which are as follows: 1. the SEQ module decides on sequencing and dispatching of jobs and deals with the details of lot combining, setups, machine loading, etc. 2. The DS or detailed scheduling module first considers the feasibility of scheduling the jobs before making specific assignments. It makes periodic checks on the progress of the orders, and if any are lagging, it reschedules them (makes inputs to the SEQ module). 3. The FA or feasibility analysis module makes "quick and dirty" analysis of the orders to determine release times. Release times are carefully considered so that when orders are released, they do not spend much time in queues. This is particularly important for semiconductor manufacturing for which ReDS is designed (Hadavi, 1994).

O-Plan2 is also an agent-based, distributed architecture that uses three agents to accomplish the planning, scheduling, and execution tasks (Currie and Tate, 1991; Tate et al., 1994). The first agent provides an interface to the user for accepting requests for scheduling tasks. The second agent is a planner that generates a plan to perform the specified task. The third agent is an execution system that monitors the execution of the planned activities. The planner responds to problems or failures in the execution of the plan, as reported by the execution system, by either replacing activities or re-planning from the start.

The main components of each agent O-Plan2 are: domain information describing the application and the tasks in the domain to the agent; plan state—the emerging plan of activities; knowledge sources—or plan modification operators that are the processing capabilities of the agent; support modules that assist the agent in its functioning; and, controller — the module that decides the order in which decisions are made. The planning proceeds in a least commitment manner with successive refinement or repair of the plan or schedule.

Burke and Prosser (1994) make a strong case for distributed scheduling arguing that a distributed problem solving architecture most closely resembles the structure of the enterprise to be scheduled. Most enterprises are distributed in nature, whether physically in terms of the location of productive resources or logically reflecting the organization structure. They propose a distributed asynchronous scheduler (DAS) that decomposes the scheduling problem and distributes it across a hierarchy of intelligent autonomous agents. Each agent has a defined role and communication path depending on its position in the hierarchy. Three types of agents are defined: the strategic agents are responsible for assigning tasks to lower agents and resolving conflicts; the tactical agents are responsible for delegation of tasks to individual resources; and the operational agents are responsible for the execution of tasks on individual resources. The agents are loosely coupled, in terms of communication between each other, but since they operate in a tightly coupled environment where a small change in a situation at one resource is rippled to other areas, they work in a reactive mode.

The system functions in a hierarchical manner by the strategic agents accepting inputs for new orders and assigning them to tactical agents and on to operational agents. The latter then try to include the tasks in their local schedule. On failure, this is communicated upward to the tactical agents that tries to rearrange the load balance. Upon failure of this, it goes up to the strategic agents to re-assign the resources. Agents keep track of their backtracking and use this knowledge to further guide the solution process.

Agent-based scheduling is an active area of research in scheduling systems. Sikora and Shaw (1997) describe an agent-based system for coordinating scheduling where each agent, though acting autonomously, depends on others to solve parts of the problem it cannot handle. Agents coordinate their activities by communicating their partial results to each other by using a "tradeoff function" to arrive at a common objective function.

Integration of design, process planning and scheduling within the same agent-based system is also an active area of investigation. Gu et al. (1997) describe a system that uses a bidding-based approach to coordinate the functions of process planning and scheduling. Each product is an agent that carries its production and due-date information and upon arriving at the manufacturing facility, its requirements are broadcast to other agents bound to resources. The agents bid for different tasks and if there is a conflict, a negotiation process is initiated. A resolution defines the process routing and resource assignment for the product. In Interrante and Rochowiak, 1994, a multi-agent system is used for assisting in dynamic scheduling and rescheduling where the focus is on concurrent engineering. Each agent represents a sub-system of the factory and mechanisms are designed for collaboration between them. The AARIA project (Parunak et al., 1997; Baker et al., 1997) uses agents to manage the entire process from ascertaining customer demand to final production. In an implemented prototype, customers can directly state their demands to the system which responds by providing a set of cost and due date schedules. Customers may choose the best cost alternative after which agents, through negotiation, create a production schedule for the product.

Cooperative Problem Solving Systems

The MacMerl system was designed to "understand and support scheduling from the perspective of the human scheduler" as opposed to implementing the expert's methods which is the approach taken by traditional expert systems (Prietula et al., 1991; Hsu et al., 1993; Prietula et al., 1994). The rationale for MacMerl's design was based on the fact that problem solving, by any intelligent agent, can be characterized as search conducted in a problem space of alternatives. The space is a representation of aspects of the task. The interactive scheduler was seen as one permitting the human and machine to be operating in coincident problem spaces. The idea was to find those decisions in which the human could be supported by the machine. To achieve this, a "backbone" system was designed based on the human scheduler's methods and supporting the achievement of goals with strong computational support. The design also had a cooperative approach for generating and reviewing schedules because the entire set of parameters to generate acceptable solutions could not be specified.

MacMerl was designed and implemented to solve the scheduling problems at a particular factory–one making windshields for automobiles. The basic production activities here were those of cutting the glass to the right size and then bending or shaping it. After this, further processes were done to attach the glass to different parts. Bending was the most critical activity where the glass had to be heated in an oven, called a lehr, and then pressed or simply shaped by gravity. Scheduling jobs for the lehr was the bottleneck activity at the plant. The scheduling task had hard constraints defining the quantity and type of glass to bend and the time in the lehr. It also had softer constraints, called preferences, that were used for determining preferences over schedules.

Given MacMerl's design philosophy of having mechanistic support states that assist task-specific human expertise, complemented by the flexible and judgmental scheduling knowledge of the human, the system was realized in three steps: first, identifying the expert's view of the scheduling problem, the expert's scheduling behavior, and the reasoning behind the expert's actions; second, defining a fundamental data structure to implement the scheduling knowledge and defining the processes (preprocessing and schedule generation); and third, interactions with the expert to review and revise the operators. After the third step, a set of operators were available that corresponded to specific goals of the scheduler.

Numao (1994) proposes an architecture where the user, procedures, and rules are used cooperatively to solve the scheduling problem. The objective of the architecture is to collaborate with the user in finding a solution. The problem is studied in a steel producing OIC plant.

The steel-making process consists of three major steps: first is that the converter refines the pig iron into steel of the desired composition by blowing oxygen through the hot molten metal; second is that the ladle-refining adds alloy ingredients or removes impurities; and third is that a continuous caster casts the molten steel into slabs, blooms or billets. The objective of scheduling is to determine the sequence of operations from the converter to the caster for a charge. This requires determining the number of charges per day (based on the demand and due date information), the waiting time limitation given that the molten steel cannot wait for certain processes for a specified amount of time, and the order of the refinement processes for different qualities of steel. The scheduling tries to minimize the waiting time and maximize the number of charges per day.

Traditional combinatorial optimization techniques are shown to be NP-complete for this kind of problem so the authors explore cooperative scheduling. The emphasis here is on decision making and decision support rather than on constraint-satisfaction. The authors rely on the fact that it is easier for an expert to suggest what is wrong with a schedule and to suggest improvements than to "extract" his knowledge into a complete set of rules. Thus, the system criteria are that: it should produce a feasible solution, it should be interactive, it should provide on-line re-scheduling, and it should be modular and compatible with existing systems. The system essentially provides a feasible solution to the user who improves it interactively.

The architecture consists of three major components: a scheduling engine, a rule-base and an interface. The scheduling engine works as a general constraint satisfier to solve general primitive constraints. The rule-base then solves the domain dependent constraints. The user refines the solution via the interface. This process is iterated until a feasible solution is available.

The scheduling process has two steps: sub-scheduling and merging followed by interactive refinement. In the first step, a schedule is generated. Human experts determine the starting of each charge set, keeping in mind the final processes have to be completed together. They disregard machine conflicts. Using the constraints of waiting time limitations and continuous casting, the system resolves the conflicts of machine scheduling. Scheduling tasks are broken into subtasks and these are backward scheduled from the casting process. Overlaps are removed, loads are balanced, and the subschedules are merged. In the second step, the user interactively modifies the schedule. The user may create conflicts in the process which are then removed by the system.

The system described by Esquirol and others (Esquirol et al., 1997) uses a constraint-based approach to provide a set of solutions for scheduling problems, combining these with a cooperative human-machine problem solving framework. The domain is that of a flanged-element manufacturing workshop that includes: cutting-out metal sheet pieces, heat treatments; flanging on a hydraulic press, and manual finishing. Routings can differ for each job, or part cut out of the sheet, so the shop operates as a job shop.

Using a cognitive modeling approach, the authors wanted to integrate the humans in the decision process rather than simulate them. Cooperation in this case implies the sharing of goals by cooperative entities which provide complementary knowledge and skills. The system, called *Scoop*, enabled users to entirely construct the scheduling solution while it checked the consistency of the constraints used.

Constraint-based analysis is used to depict processing constraints and temporal constraints. This analysis shows infeasibilities to the users in the current set of decisions and a priority order of possible solutions is presented to them in different modes.

The architecture consists of two major modules: the Selection aid and the Placement aid. The Selection aid creates the subset of orders that have to be prioritized and presented to the user, while the Placement aid indicates where certain orders can be placed successfully. The users also see a graphical representation of the sheet that is being cut and flanged.

Knowledge Elicitation for Scheduling Systems

The manner in which knowledge was extracted from expert schedulers to implement in the different systems varied according to the design envisaged for the system. In some cases, it was sufficient to extract knowledge about the domain without finding out how the scheduler actually did the scheduling, while in other cases the latter was important also. For constructing the ISIS system, Fox (1994) interviewed experts to identify the scheduling knowledge required for building the system. They found that the most interesting and important knowledge related to constraints that bind the scheduling the problem. Fox identified five broad categories of constraints which included; organizational, physical, causal, availability, and preference constraints. Organizational constraints included: due date constraints restricting the lateness of orders, work-in-process limits, and physical constraints specifying characteristics of resources that limit functionality (such as the length of the milling machine's workbed). Causal constraints specified the preconditions for use of resources such as the precedence requirements (job has to be cleaned before it can be annealed) or resource requirements (requirements of tools, or trained operators for a job), while availability of resources constraint restricted the simultaneous use of any resource by multiple jobs. Preference constraints were "soft" constraints that indicated priorities or preferences for certain resources.

For extracting knowledge from the expert to construct MacMerl, Prietula and others went through several phases of analysis and study of the expert's problem solving behavior. Their objective was to not only understand the characteristics of the factory but also to have a trace of the expert's cognitive activities. To begin, they used protocol analysis as the expert developed schedules, direct questioning, post-task analysis, and extended apprenticeship to understand the problem solving methods, representation, and causal reasoning used by the scheduler. Detailed interviews were used to understand the scheduler's goals and heuristics. Combinations of parts, determined from the part characteristics that the scheduler used, were recorded to limit the search space. They also identified a general algorithm that the scheduler followed based on specific goals. These goals attempted to reduce setups, minimize stock-outs, and prioritized high-volume sales parts. A causal model of the expressed constraints and preferences was identified by continuously asking the expert to explain and justify all decisions.

A similar technique was used by May and Vargas (1996) and De' (to appear) to identify characteristics of the factory and also to elicit the process by which the scheduler constructed a mental simulation of the factory. This process was studied by extensive *in situ* protocol analysis, wherein the researchers observed and recorded the expert as he went about his daily tasks and talked about what he was doing. After spending several months acting as trainees, they were in a position to codify the problem solving behavior of the expert. May and Vargas also noted that the eventual model of behavior that emerged from their apprenticeship was different from the set of scheduling rules that the expert had mentioned to them at the beginning of the exercise (in the presence of management).

In Esquirol et al. (1997), the knowledge acquisition was done by using a modification of the personal construct theory where users were asked to identify their most important constraints on a grid. All current and potential users of the system were asked to participate and give their most important concerns. The analysis of the grid led to a design of the cooperative strategy including flexibility, support, and interactivity.

OR Algorithms in Systems

Shah et al. (1992) outline a system that provides meta-knowledge about the kinds of scheduling algorithms to use. They argue that researchers and practitioners waste effort searching the literature to seek out algorithms that will fit their particular scheduling situation. So they present a knowledge-based system, where the expert is a scheduling researcher. The expert takes input about the type of setup configuration of a particular shop floor situation, the type of constraints binding on the situation (such as due dates, delay costs, etc.), and the type of objective function to be used (such as minimize tardiness, or average completion time, etc.). The expert then gives, as output, a sequence of OR algorithms by which the problem can be solved. The system does not solve the problem itself.

Kempf (1994) and others designed their system to use published OR algorithms as part of their system. This was to be in the predictive component of their system which was designed to have an "importance-ordering" mechanism at a higher level to order tasks according to priority, not according to the time-order of tasks where priority is not regarded. Importance-ordering would enable a global view of the situation and not rely entirely on the local details. At a lower level, this component used a "multi-algorithm" approach where a number of algorithms from the published literature were used for given situations on the factory floor. (The reactive component was tied to the predictive part in that it made decisions that had been specified by the predictive component. In situations where the system could not react, the predictive component was restarted.)

Generalized Scheduling Systems

The constraint-based scheduling system designs discussed above are all attempts to realize a general representational framework within which any type of scheduling problem can be tackled. There are, however, attempts to take this idea one step further by designing entire architectures that can, with little modification, be adapted for various scheduling applications.

The ARPA-Rome knowledge-based planning and scheduling initiative (Fowler et al., 1995) was created based on the observation that although constraint-based frameworks have been successful in solving scheduling problems, they do not scale to larger problems very well. There are few that fully integrate the planning, scheduling and database components within a single architecture. At the system's (ARPI) core is a distributed network of graphics-based planning cells sharing a common reasoning infrastructure that enables continuous concurrent planning. Human users collaborate with the automated decision aids to rapidly generate large scale plans and schedules, with full use of all the background and relevant data. This initiative led to the creation of the common prototype environment (Burstein et al., 1995) consisting of a repository of software tools and a testbed for evaluation of planning and scheduling systems.

Another generic framework for building practical scheduling systems is described by Sauer and Bruns (1997). The system would enhance the problem solving capabilities of human domain experts. The generic framework is based on two design principles–the combination of standard components (such as user interfaces), databases with knowledge-based concepts, (such as heuristic scheduling algorithms), and declarative knowledge representation and the explicit representation of scheduling knowledge for flexible reuse and adaptation.

The first principle provides support for predictive, reactive, and interactive scheduling. The user interface provides a graphical depiction of the scheduling situation, allowing the user to change priorities, resources, etc., while simultaneously checking for consistency. The knowledge-base contains the production management knowledge of the application domain—knowledge about products, resources, and solution schedules—in a relational form. It also contains rules that represent hard or soft constraints.

Algorithms for scheduling are used to create predictive schedules from given input conditions. Problem specific knowledge is used to guide the search, including heuristics from an expert scheduler. Reactive scheduling algorithms enable appropriate reactions to unexpected events by adjusting the schedules. The reactions can range from simple manipulations to complete rescheduling.

The second principle enables the reusability of algorithms that have been designed for highly specific scheduling situations. The framework separates predictive and reactive scheduling algorithms in underlying scheduling strategies (order-based or resource-based) and the selection rules to be used in these strategies. Predictive and reactive "skeletons" are used to separate the two approaches, leading to separate deployment and adaptability. The rules and strategies were determined from experts or from existing systems.

The framework was tested by building schedulers for three different applications–two in manufacturing and one in medicine. All three knowledge-based systems greatly reduced the time to create and maintain production schedules. (All were implemented in Prolog on Sun SparcStations.)

Learning in Scheduling Systems

The learning of scheduling heuristics has concerned researchers for many reasons (Aytug et al. 1994). It enables the learning of scheduling heuristics automatically, overcoming the problem of knowledge acquisition from experts. Learning methods also enable the systems to learn about the environment and improve on their performance through experience.

Piramuthu and others (Piramuthu et al., 1994; Piramuthu et al., 1993) describe a system that learns to apply scheduling heuristics in response to certain patterns in the environment. The system consists of two major components–a simulation module that generates training examples of various shop floor conditions, and an inductive learning module that learns the best scheduling rule (such as SPT or EDD) to apply to the given conditions. Over time, the system learns more decision rules. The system has a bi-level model, where the first level deals with part-release and the second level deals with dispatching at individual machines.

The system is designed to simulate a surface mount technology process consisting of various stages through which fourteen different part types are processed. The objective of the inductive learner is to learn rules of the form: "If $(b_{i_1} \geq a_{i_1} \geq c_{i_1})$ And $\cdots(b_{im} \geq a_{im} \geq c_{im})$ Then τ," where a_{ij} represents the ith level of the jth attribute and b_{ij}, c_{ij} define the range for a_{ij}. τ denotes the class (Piramuthu et al., 1994). The a_{ij}'s could represent parameters such as buffer content, machine utilization, or coefficient of variation of processing times. τ would be a part-release or dispatching rule.

A training example generator creates examples of shop floor patterns for the learning module of the system. The learning module passes the examples through a feature extraction preprocess and then through an inductive learning algorithm, which is a refinement of the ID3 algorithm (Quinlan, 1986). The decision rules obtained are used by a pattern-directed scheduling module. This module uses a smoothing function to remove the "chattering" that results from too many changes in the scheduling heuristic with changes in the input patterns. The authors use two options for smoothing–one has a constant threshold value and another that weighs current patterns more heavily. A critic module is used to evaluate the performance of the scheduler, in cases where performance tends to deteriorate (due to over-generalization, for instance). Results of running multiple simulations for many sets of patterns over the system showed its performance to be better by using the pattern-directed learning approach rather than by using the part-release or dispatch heuristics alone.

Simulations are also used to create training examples for a multi-dimensional classification algorithm in Chaturvedi and Nazareth's system (1994). The authors state the theory underlying conditional classification as an extension of simple classification where, in the former, the classification is performed once along one dimension and the output from this is used as priors in the second classification. This method is able to include more complexity of the shop floor scheduling phenomena than the simple classification.

In their experiments, the authors generate decision rules for a FMS shop floor using two output classification dimensions of production rate and utilization for machines. The shop floor is simulated and training and testing examples are obtained. The decision rules are able to accurately classify all the test examples. The training and test sets used are small, and the authors caution against over generalizing the results of their research.

Chen and Yih (1996) trained a neural net to identify the attributes required for developing a knowledge base. Their neural net could identify a set of relevant attributes, for a given problem situation, from a general pool of attributes. Their test case showed that the system could successfully select important attributes from a pool.

The CABINS system (Miyashita and Sycara, 1994; Miyashita et al., 1996) implements a methodology for learning a control level model for selection of heuristic repair actions based on experience. The scheduling of resources is based on constraint-directed scheduling methods. Case-based reasoning is used for the acquisition and flexible reuse of scheduling preferences and selection of repair actions. The case-base includes examples that collectively capture performance trade-offs under diverse repair situations. The case description captures the dependencies among the scheduling features, the repair context, and a suitable repair action. The dependencies in the case-base are used to dynamically adjust the search procedure. Users provide evaluations of the outcome of using cases to select search heuristics and these are later reused to select heuristics in similar situations. User preferences are reflected in the case-base in two ways: as preferences for selecting a repair tactic depending on the features of the repair context and as evaluation preferences for the repair outcome.

Experimental results with the system showed that the approach outperformed dispatch heuristics for similar problems and constraint-based scheduling using only static search procedures. It was useful in capturing user preferences not present in the scheduling model.

Zweben et al. (1992) describe a system that learns search control knowledge for a constraint-based scheduling system. The authors maintain that the efficiency of scheduling systems, based on constraints, is affected by resource contention. Their system learns the conditions under which chronic resource contention occurs and modifies its search control to avoid repeating those conditions. They modify the existing method of explanation-based learning to learn from multiple plausible explanations. Zhang and Dietterich (1995) use a neural net to learn a heuristic evaluation function (to control search), for the same problem of scheduling studied by Zweben. Their system uses the evaluation function for a one-step look-a head procedure. The results indicate that the performance of the system improves on the performance of the iterative repair method used by Zweben.

Simulation-Based Scheduling Support

Simulations allow users to see the effects of their decisions on the conditions on the shop floor. Systems that use heuristics to guide the simulation allow the users' expert heuristics to be encoded within simulation. The system by Jain et al. (1990) used deterministic, backward simulation to provide real-time support to shop floor personnel. Backward simulation enabled the system to simulate backward, from a given event (say the completion of a job), to the starting point. Heuristics obtained from an experienced scheduler were used to make decisions at crucial choice points which included part dispatching, machine selection, interval selection, and secondary resource constraints.

The SIMPSON system (May and Vargas 1996), also simulated the factory floor based on heuristics obtained from an experienced scheduler. The focus was on bottleneck resources where heuristics were used to 'feed the bottleneck' that would also ensure the desired utilization of other resources. In the same domain, the PLANOPTICON system, (De' 1996), also used a knowledge-based simulation approach to suggest setup changes for bottleneck machines. In both cases, the user could input hypothetical data to arrive at what-if scenarios.

1.3 Implementation Issues

In this section, we consider the issues related to the implementation and fielding of shop floor scheduling systems. Although the specific factory and organization determines in a large part the eventual implementation, there are a number of issues that are common to the implementation process. These issues include the access to transaction databases, the implementation of user interfaces, the use of scheduling horizons, and cultural issues.

The implementation issues are studied in reported instances of fielded systems. There are a fairly large number of fielded scheduling systems that have been reported in the literature. For this study, we focus on a few of these that have been implemented specifically in manufacturing settings and have been used for a significant portion of time for regular production scheduling. The systems studied are: LMS (Fordyce and Sullivan, 1994), ReDS (Hadavi, 1994; Hadavi et al., 1992), the systems at Intel (Kempf, 1994), the systems developed at Texas Instruments (Fargher and Smith, 1994), the DAS system (Lee et al., 1995), the MicroBOSS scheduler (Fowler et al., 1995), the high-grade steel-making scheduler (Dorn and Shams, 1996; Dorn and Slany, 1994), the cooperative scheduler for steel-making (Numao, 1994), the MacMerl scheduler (Prietula et al., 1994; Prietula et al., 1991; Hsu et al., 1993), the SIMPSON scheduler (May and Vargas, 1996), and the system at General Motors (Jain et al., 1990). Two other fielded systems that were not used to schedule the entire factory, but only specific resources, are those by Burke and Prosser (1994 and De', 1996).

Transaction Databases

Transaction databases are crucial for the implementation of any realistic scheduling system. These databases log the transactions occurring on the shop floor related to starting of processing of jobs, status of machines, completion of jobs, operator assignment, setup changes, etc. Scheduling systems rely on these to obtain current status reports from the shop floor.

In all the reported instances of scheduling system implementations, the developers have first had to create or deploy interface software to retrieve the transaction data from the management information systems. Depending on the currency of the data, or the frequency at which shop floor status reports were available, the scheduling system could update its records and respond to the situation. All the developers note that culturally this data access was the first and possibly the most important hurdle to overcome to have a viable implemented system.

The common issues related to transaction databases are:

- Building interfaces with the MIS databases to obtain the relevant data. This involves building a working relationship with the MIS department to identify the nature of the data, its manner of capture, and its representation.
- Ensuring that the data is entered correctly and in a timely manner. In most cases, this requires re-setting the data entry patterns and re-training employees to do this.
- Collecting distributed data stored in diverse forms (i.e., on different types of application software), and translating it into a form understandable to the scheduling system.
- Building additional data collection mechanisms if the existing system is not collecting data on all the steps in the manufacturing process.

User Interfaces

Constructing attractive and user-friendly interfaces was a deliberate implementation tactic employed by some developers. The motive was to get the user, in some cases, the expert involved in the development process as a provider of scheduling heuristics and system requirements. Some important issues with regard to user interfaces are:

- The interfaces have to show timely and useful information to the users. To build the LMS system, Fordyce and Sullivan built interfaces that showed the shop floor in a graphical manner and also had signals to indicate problem situations or warnings. These interfaces helped to build support for their scheduling system.
- The interfaces should be as close as possible to the current forms or screens used by the users. May and Vargas designed input screens and output reports for the user of SIMPSON that were identical to what he was used to on paper and preferred. For the MacMerl system, the developers made the interface used by the scheduler as user-friendly as possible.
- The schedule items depicted on the screen should be easily manipulable (such as being mouse-sensitive) or modifiable by the user. Numao used such an approach to allow the user to make on-fly-changes to input conditions and see the resulting effects.

Scheduling Horizons

Implementations of scheduling systems, particularly with hierarchical architectures, necessitate the explicit use of horizons. The horizons are time boundaries within which decisions are located. They are usually pre-determined and the system is designed to limit the schedules assignments up till those points.

In the LMS system, the lowest level of the hierarchy, called the dispatch scheduling tier, has a horizon of a few hours to a few weeks. Here are where the pertainings to decisions running test lots, prioritizing late lots, determining maintenance downtime, determining downstream needs, etc., are made. A short horizon was needed for most decisions at this level because of the dynamic nature of the shop floor where the validity of decisions degenerated quickly.

Kempf distinguishes the predictive and reactive modules of the system (Kempf 1994) on the basis of the scheduling horizons they address. The predictive part would work on a horizon of a few shifts to allocate capacity between production, maintenance, etc., whereas the reactive part would work on a horizon of a few minutes to respond to occurrences on the shop floor.

In the ReDS system (Hadavi), the authors had to build a system corresponding to a particular scheduling horizon, primarily because it was so desired by the management. This was a predictive system which would refresh the schedule after every time horizon with new data from the shop floor. The horizon was set by the frequency by which the new data was available.

For setting explicit scheduling horizons, De' (1993) argues that horizon parameters have to be established whose values are determined from shop floor parameters. Long or short horizon values are determined from parameters such as the number of products. For different and persisting shop-floor conditions, different horizon lengths may be required.

Cultural Issues

Cultural and political issues play as much of an important role in the deployment and eventual success of a scheduling system as do the technological issues. These issues arise from the community of people who are concerned with the manufacturing process and whose cooperation and support is needed for any system implementation. The incidents below show the cultural reasons why systems may be supported or opposed, regardless of the merit of the systems themselves.

For the LMS project, Fordyce and Sullivan (Fordyce and Sullivan, 1994) recalled two incidents that facilitated the system's implementation. The first was a call by a supervisor to be provided a real-time monitoring system that would warn of any impending deviations from the plan and the second was a problem observed in the simulation systems. A review of the manufacturing systems revealed several shortcomings, chief among them was insufficient integration between real-time data, knowledge bases, and tactical guidelines. To build support for their eventual scheduling systems, the authors first built a real-time monitoring system that would enable shop supervisors and engineers to observe events on the shop floor through desktop terminals. Users were weaned on the systems, after which they demanded more features such as alerts and rule bases, that further propelled the development of the system.

In the second phase, the system design shifted from a reactive mode to a more proactive mode. This was achieved in a time of booming business, when greater throughput and planning was required from the existing systems. Using a mix of knowledge engineering and operational analysis, the authors were able to implement a set of dispatch heuristics that improved throughput and profitability. Following this success, the value of knowledge-based systems was firmly established and, along with a renewed interest by IBM in participating in the manufacturing marketplace, further development and implementation of such systems was assured.

At Intel, (Kempf, 1994), the scheduling research group had to market their ideas to two different sets of people–the managers and the users on the shop floor. The first group had to be persuaded to support the project, given that the further any senior executive was from the shop floor, the less concerned he or she was with scheduling problems. This group was persuaded through promises and assurances of greater responsiveness, better resource utilization, and integration with automated tools. The second group was cognizant of the scope of the problems but remained skeptical of any systemic solution for various reasons–the dynamic and changing nature of the shop floor, the complexity of the problem, the adherence to manual procedures, and the history of failed attempts to solve the problems. This latter group was persuaded to support the development effort by involving them with the development process. Resistance to change remained, however, because changing work processes for even one person meant changing the grading, reporting, and training of possibly an entire group working with the individual.

The first modules of the system implemented on the shop floor were the user interface and the data interface. Users were asked to comment on the design of the interface, and in a situation where personnel had a very wide degree of familiarity with graphical interfaces, the suggestions were diverse and many. The system builders could not cope with them. The database interface suffered from inconsistencies and inaccuracies in data being logged in on the shop floor. The scheduling system could not proceed without an accurate snapshot of the shop floor; thus, the developers had to persuade those in charge of collecting data to improve their system and practices.

In the ReDS system (Hadavi, 1994), the developers first implemented a research prototype that they used to show management the possibilities of such a system and gain support for it. Users were aware of such a prototype but when they saw it, they were disappointed with its limited capabilities. The developers tried to deploy the prototype itself, which led to further annoyance and friction. The biggest drawback was that there were no means of collecting shop floor data for input to the system. The developers had to start another implementation from scratch. This time, they collaborated with the MIS personnel to obtain useful and timely data from the shop floor system. Eventually there were no objections to the introduction of the system; however, some users expressed doubts as to its usefulness over the older simulation-based system.

The eventual introduction of the system brought many changes in the practices on the shop floor. Management now had a means by which to directly track the progress of jobs on the shop floor. Any delays would be immediately visible and liable to questioning. This introduced a form of *panopticism* (Zuboff, 1988) that forced shop floor personnel to be alert at all times.

1.4 Validation Issues

Testing implemented systems involves the twin tasks of verification and validation. Simply stated “verification is building the system right, validation is building the right system” (O’Keefe et al., 1987). Validation evaluates the system in the context for which it is designed. Knowledge-based systems are often validated by seeking acceptable performance levels for specified criteria (O’Keefe and Preece, 1996).

Validation of implemented scheduling systems is very difficult and few developers do so in any detailed manner. The reasons for this difficulty are the lack of standards by which to measure the performance of the system (Gary et al., 1995). Real-life shop floors are so complex that generating a theoretical model and solving one to arrive at an optimal standard is impossible. Scheduling researchers try to validate the system by various means. Measuring its performance on specific goal parameters is the one most widely used. There are, however, many ways by which knowledge-based systems in general can be validated (O’Keefe and O’Leary, 1993). Internal functioning of the systems are verified by unit and system testing.

Validation Against System Objectives or Goals

In most cases, support for the development and implementation of large-scale and expensive factory scheduling systems is sought from management on conditions of meeting specified goals. These goals usually pertain to improvements in productivity, throughput, tardiness, utilization, cost savings, etc. Subsequent validation of the system then relies on showing that the goals have been met.

- Sadeh’s MicroBoss scheduler (Fowler et al., 1995) showed a 55 to 60% improvement in lead times, for actual load conditions, for scheduling over 1000 part types at a Raytheon manufacturing facility. The system reduced average tardiness by 14 to 16% and inventories by 20 to 30%.
- IBM’s LMS was able to improve productivity by about 35% and avoided a capital outlay of $10 million.
- The ReDS system improved productivity and cycle times by a “considerable amount.” The developers contend that, due to the many changes on the shop floor, it is hard to assign any particular amount of productivity increase to the system alone.
- Numao (Numao, 1994) shows that their scheduling system, used for scheduling steel-making, enabled a reduction in scheduling time from 3 hours to 30 minutes. This made real-time scheduling and re-scheduling possible. It also improved the quality of the schedule by reducing the average waiting time for the charge, molten iron, from 16 minutes to 8 minutes. This resulted in a saving in energy costs totalling about $1 million a year.
- Dorn and Shams (1996) report no direct impact on the production process of their expert system implemented for supporting steel-making; however, they do mention a number of qualitative benefits. The system eliminated a lot of paperwork and reduced weekly planning times. It enabled the schedulers to seek better sequencing through what-if analysis. It supported a deterministic decision framework that improved overall production quality.
- For the DAS scheduler, Lee et al., (1995) estimate an annual benefit of about $4 million to Daewoo Shipbuilding Company. This estimate is arrived at by adding a production productivity improvement of 30% and a planning productivity improvement of 50%.

System developers warn against ascribing the improvements in certain criteria to the systems performance alone. Many changes take place on the shop floor, as well as the tasks performed by personnel, when a large factory-wide scheduling system is installed. Simply the access to control information by management affects the performance of shop floor personnel. The requirement for entering the data accurately and punctually, the ability to see the status of upstream and downstream machines, and the facility of having tasks prepared by an automation device, adds to the changes in the shop floor practices. Thus, when changes in productivity occur, whether positive or negative, these cannot be simply credited or blamed on the new system.

Validation Against Manual Scheduler's Criteria

Often the scheduling expert, on whose knowledge the expert system has been based, becomes the judge for the performance of the system. The scheduler verifies that the schedules produced for particular situations are appropriate and in accordance with what he would have done. Though this approach has certain merits, it is often criticized on grounds that there is no assurance that the scheduler's solution is near the optimal one. Schedulers simply do not rely on the shop-floor situation information available to the system to make their schedules but actively influence the various parameters (such as demand and machine capacity) (McKay et al., 1995). Schedulers may not be able to critique large scale schedules produced by the system, and the problems identified by the schedulers may not be entirely significant (Kempf et al., 1991a).

For situations where only a few resources are being scheduled by the system, the method of using the scheduler's judgement is adequate. De', (1996) used a historical record to compare the system's output against what was actually scheduled on the shop floor. The historical record reflected a combination of the schedulers decisions and contingency moves made by shift supervisors and workers. The system performed reasonably well by this measure. While implementing the MacMerl system, Prietula et al. (1994) constantly asked the scheduler about the quality of the schedules and the user interface. The system was fine tuned until the schedules it produced resembled those of the scheduler, only they were more consistent and exhaustive.

Validation With Simulations

A simulation model of the factory processes makes available a fairly large set of test cases that can be used to validate the scheduling system. Such a method is deployed by Fargher and Smith, (1994) where plan validation was performed by comparing the output from the system with simulation results, both using data from a type of factory modeled by the system. Close agreement between the system and the simulation suggested that the performance of the system was adequate.

1.5 Conclusions

In 1991, Kempf and others (Kempf et al., 1991b), while reviewing the state of knowledge-based scheduling systems, stated that despite there being a large amount of research targeted at this problem, there were few fielded systems being used in daily manufacturing practice. Ten years later, the current situation is different since there are a fair number of fielded systems that have survived the uncertainties of the shop floor and are being used daily.

The research in designs of scheduling systems continues unabated. Of the design issues discussed in this chapter, the two that are receiving the most attention, and are likely to continue to do so, are those of distributed scheduling and learning in scheduling systems. With the widespread integration of information technology within all aspects of the manufacturing processes and the organization as a whole (through enterprise-wide integration systems), distributed agent-based systems have emerged as the most appropriate technology with which scheduling can be performed. Agent programs are designed to be autonomous, relatively independent of the users and other modules for their functioning, and can reside on various machines from which data can be directly obtained for scheduling purposes. Agents can also act as monitors, informing about the performance of certain resources and as negotiators, using utility-based or game-theoretic models to resolve assignment conflicts.

Learning in scheduling systems addresses the single bottleneck issue facing developing knowledge-based systems–that of extracting knowledge. Learning systems try to obtain, from a set of training examples, a set of generalized rules for scheduling. This task is especially difficult because of the tremendous complexity of any shop floor system for which generalizations are being sought. A few systems have obtained promising results but it is clear that much more research needs to be done in this area.

Another design area that is gaining importance is that of generalized scheduling systems. These systems will, in some measure, obviate the need for extensive knowledge elicitation from schedulers and users. They are being designed with the aim of having entire production systems built around them in order to simplify the product design, production planning, and scheduling tasks. In such situations, users would learn to use the scheduling system while they are learning to use the production system. Also, the scheduling system will be composed of distributed, autonomous agents that have adaptive capabilities, which will further reduce the need for knowledge gathering from shop floor personnel.

It was evident from the review of the implemented systems that the cultural and political issues were very important for the final acceptance of the scheduling system. Once the users were actively involved with the activities of providing specifications and testing the software, a certain measure of commitment and acceptance was obtained from them. The lesson to be learned here has been described in detail before, in the discipline of software engineering, where developers are urged to include the users and the sponsors of the system within the development life cycle.

References

Baker, A., Parunak, H., and Erol, K. (1997). Manufacturing over the internet and into your living room: Perspectives from the aaria project. Technical report, Dept. of Electrical and Computer Engineering and Computer Science, University of Cincinnati.

Burke, P. and Prosser, P. (1994). The distributed asynchronous scheduler. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 11, pages 309–340. Morgan Kaufman Publishers.

Burstein, M., Schantz, R., Bienkowski, M., desJardins, M. E., and Smith, S. (1995). The common prototyping environment: A framework for software technology integration, evaluation, and transition. *IEEE Expert,* 10(1):17–26.

Chaturvedi, A. and Nazareth, D. (1994). Investigating the effectiveness of conditional classification: An application to manufacturing scheduling. *IEEE Transactions on Engineering Management,* 41(2): 183–193.

Chen, C. and Yih, Y. (1996). Identifying attributes for knowledge-based development in dynamic scheduling environment. *International Journal of Production Research,* 34(6):1739–1755.

Currie, K. and Tate, A. (1991). o-plan: the open planning architecture. *Artificial Intelligence,* 52:49–86.

De', R. (1993). *Empirical Estimation of Operational Planning Horizons: A Study in a Manufacturing Domain.* Ph.D. thesis, University of Pittsburgh.

De', R. (1996). A knowledge-based system for scheduling setup changes: An implementation and validation. *Expert Systems With Applications,* 10(1):63–74.

De', R. (To appear). An implementation of a system using heuristics to support decisions about shop floor setup changes. In Yu, G., editor, *Industrial Applications of Combinatorial Optimization.* Kluwer Academic Publishers.

Dorn, J. and Shams, R. (1996). Scheduling high-grade steelmaking. *IEEE Expert,* 11(1):28–35.

Dorn, J. and Slany, W. (1994). A flow shop with compatibility constraints in a steelmaking plant. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 22, pages 629–654. Morgan Kaufman Publishers.

Esquirol, P., Lopez, P., Haudot, L., and Sicard, M. (1997). Constraint-oriented cooperative scheduling for aircraft manufacturing. *IEEE Expert,* 12(1):32–39.

Fargher, H. E. and Smith, R. A. (1994). Planning in a flexible semiconductor manufacturing environment. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 19, pages 545–580. Morgan Kaufman Publishers.

Fordyce, K. and Sullivan, G. (1994). Logistics management system (Ims): Integrating decision technologies for dispatch, scheduling in semiconductor manufacturing. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 17, pages 473–516. Morgan Kaufmann Publishers.

Fowler, N., Cross, S., and Owens, C. (1995). Guest editor's introduction: The arpa-rome knowledge-based planning and scheduling initiative. *IEEE Expert,* 10(1):4–9.

Fox, M. and Smith, S. F. (1984). ISIS — a knowledge-based system for factory scheduling. *Expert Systems,* 1(1):25–49.

Fox, M. S. (1983). *Constraint-Directed Search: A Case Study of Job Shop Scheduling.* Ph.D. thesis, Carnegie-Mellon University.

Gary, K., Uzsoy, R., Smith, S., and Kempf, K. (1995). Measuring the quality of manufacturing schedules. In Brown, D. and Scherer, W., editors, *Intelligent Scheduling Systems.* Kluwer, Boston.

Godin, V. (1978). Interactive scheduling: Historical survey and state of the art. *AIIE Transactions,* 10(3):331–337.

Gu, P., Balasubramanian, S., and Norrie, D. (1997). Bidding-based process planning and scheduling in a multi-agent system. *Computers and Industrial Engineering,* 32(2):477–496.

Hadavi, K., Hsu, W., Chen, T., and Lee, C. (1992). An architecture for real-time distributed scheduling. *AI Magazine,* 13(3):46–56.

Hadavi, K. C. (1994). A real time production scheduling system from conception to practice. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 20, pages 581–604. Morgan Kaufman Publishers.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence,* 26:251–321.

Hsu, W. L., Prietula, M. J., and Ow, P. S. (1993). A mixed-initiative scheduling workbench: Integrating ai, or, and hci. *Decision Support Systems,* 9(3):245–257.

Interrante, L. D. and Rochowiak, D. M. (1994). Active rescheduling and collaboration in dynamic manufacturing systems. *Concurrent Engineering: Research and Applications,* 2(2):97–105.

Jain, S., Barber, K., and Osterfeld, D. (1990). Expert simulation for on-line scheduling. *Communications* of the ACM, 33(10):55–60.

Kempf, K., Pape, C. L., Smith, S., and Fox, B. (1991a). Issues in the design of ai-based schedulers: A workshop report. *AI Magazine,* 11(5):37–46.

Kempf, K., Russell, B., Sidhu, S., and Barrett, S. (1991b). Ai-based schedulers in manufacturing practice. *AI Magazine,* 11(5):46–55.

Kempf, K. G. (1994). Intelligently scheduling semiconductor wafer fabrication. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 18, pages 517–544. Morgan Kaufmann Publishers.

Kerr, R. (1992). Expert systems in production scheduling: Lessons from a failed implementation. *Journal of Systems Software,* 19:123–130.

Lee, J., Lee, K., Hong, J., Kim, W., Kim, E., Choi, S., Kim, H., Yang, O., and Choi, H. (1995). Das: Intelligent scheduling systems for shipbuilding. *AI Magazine,* 16(4):94.

May, J. H. and Vargas, L. G. (1996). Simpson: An intelligent assistant for short-term manufacturing scheduling. *European Journal of Operational Research,* 88:269–286.

McKay, K., Buzacott, J., and Safayeni, F. (1995). 'Common sense' realities of planning and scheduling in printed circuit board production. *International Journal of Production Research,* 33:1587–1603.

Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision,* pages 211–277. McGraw-Hill, New York.

Miyashita, K. and Sycara, K. (1994). Adaptive case-based control of schedule revision. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 10, pages 291–308. Morgan Kaufman Publishers.

Miyashita, K., Sycara, K., and Mizoguchi, R. (1996). Modeling ill-structured optimization tasks through cases. *Decision Support Systems,* 17:345–364.

Numao, M. (1994). Development of a cooperative scheduling system for the steel-making process. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 21, pages 607–628. Morgan Kaufman Publishers.

O'Keefe, R., Balci, O., and Smith, E. (1987). Validating expert system performace. *IEEE Expert,* 2(4):81–90.

O'Keefe, R. and O'Leary, D. (1993). A review and survey of expert system verification and validation. *Artificial Intelligence Review,* 7(1):3–42.

O'Keefe, R. and Preece, A. (1996). The development, validation, and implementation of knowledge-based systems. *European Journal of Operations Research,* 92:458–473.

Ow, P., Smith, S. F., and Thiriez, A. (1987). Reactive plan revision. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* pages 77–82, San Mateo, CA. Morgan Kaufmann.

Ow, P. S. and Smith, S. F. (1987). Two design principles for knowledge-based systems. *Decision Sciences,* 18(3):430–447.

Parunak, H., Baker, A., and Clark, S. (1997). The aaria agent architecture: An example of requirements driven agent-based system design. In *Proceedings of the First International Conference on Autonomous Agents* (*ICAA '97*), Marina del Rey, CA.

Piramuthu, S., Raman, N., and Shaw, M. (1994). Learning-based scheduling in a flexible manufacturing flow line. *IEEE Transactions on Engineering Management,* 41(2):172–182.

Piramuthu, S., Raman, N., Shaw, M., and Park, S. (1993). Integration of simulation modeling and inductive learning in an adaptive decision support system. *Decision Support Systems,* 9:127–142.

Prietula, M., Hsu, W.-L., Ow, P., and Thompson, G. (1994). Macmerl: Mixed-initiative scheduling with coincident problem spaces. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 23, pages 655–682. Morgan Kaufman Publishers.

Prietula, M. J., Hsu, W. L., and Ow, P. S. (1991). A coincident problem space perspective to scheduling support. In *Proceedings of the Fourth International Symposium on Artificial Intelligence,* Cancun, Mexico.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning,* 1:81–106.

Sauer, J. and Bruns, R. (1997). Knowledge-based scheduling systems in industry and medicine. *IEEE Expert,* 12(1):24–31.

Shah, V., Madey, G., and Mehrez, A. (1992). A methodology for knowledge based scheduling decision support. *Omega, International Journal of Management Science,* 20(5/6):679–703.

Sikora, R. and Shaw, M. J. (1997). Coordination mechanisms for multi-agent manufacturing systems: Applications to integrated manufacturing scheduling. *IEEE Transactions on Engineering Management,* 44(2):175–187.

Smith, S. (1987). A constraint-based framework for reactive management of factory schedules. In *Proceedings of the International Congress on Expert Systems and Leading Edge in Production Planning and Control,* Charleston, SC.

Smith, S. F. (1994). Opis: A methodology and architecture for reactive scheduling. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 2, pages 29–66. Morgan Kaufmann Publishers.

Szelke, E. and Kerr, R. M. (1994). Knowledge-based reactive scheduling. *Production Planning & Control,* 5(2):124–145.

Tate, A., Drabble, B., and Kirby, R. (1994). O-plan2: An open architecture for command, planning, and control. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 7, pages 213–239. Morgan Kaufmann Publishers.

Zhang, W. and Dietterich, T. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.*

Zuboff, S. (1988). *In the Age of the Smart Machine: The Future of Work and Power.* Basic Books, Inc.

Zweben, M., Daunn, B., Davis, E., and Deale, M. (1994). Scheduling and rescheduling with iterative repair. In Zweben, M. and Fox, M. S., editors, *Intelligent Scheduling,* chapter 8, pages 241–255. Morgan Kaufmann Publishers.

Zweben, M., Davis, E., Daun, B., Drascher, E., Deale, M., and Eskey, M. (1992). Learning to improve constraint-based scheduling. *Artificial Intelligence,* 58:271–296.