18 Computer Integrated Production Scheduling

Within all activities of production management, *production scheduling* is a major part covering planning and control functions. By *production management* we mean all activities which are necessary to carry out production. The two main decisions to be taken in this field are production *planning* and production *control*. Production scheduling is a common activity of these two areas because scheduling is needed not only on the planning level as mainly treated in the preceding chapters but also on the control level. From the different aspects of production scheduling problems we can distinguish *predictive production scheduling* or *offline-planning* (*OFP*) and *reactive production scheduling* or online-control (*ONC*). Predictive production scheduling serves to provide guidance in achieving global coherence in the process of local decision making. Reactive production scheduling is concerned with revising predictive schedules when unexpected events force changes. OFP generates the requirements for ONC, and ONC creates feedback to OFP.

Problems of production scheduling can be modeled on the basis of distributed planning and control loops, where data from the actual manufacturing process are used. A further analysis of the problem shows that job release to, job traversing inside the manufacturing system and sequencing in front of the machines are the main issues, not only for production control but also for short term production planning.

In practice, scheduling problems arising in manufacturing systems are of discrete, distributed, dynamic and stochastic nature and turn out to be very complex. So, for the majority of practical scheduling purposes simple and rigid algorithms are not applicable, and the manufacturing staff has to play the role of the flexible problem solver. On the other hand, some kind of Decision Support Systems (*DSS*) has been developed to support solving these scheduling problems. There are different names for such systems among which "Graphical Gantt Chart System" and "Leitstand" are the most popular. Such a DSS is considered to be a shop floor scheduling system which can be regarded as a control post mainly designed for short term production scheduling. Many support systems of this type are commercially available today. A framework for this type of systems can be found in [EGS97].

Most of the existing shop floor production scheduling systems, however, have two major drawbacks. First, they do not have an integrated architecture for the solution process covering planning and control decisions, and second, they do not take sufficient advantage from the results of manufacturing scheduling theory. In the following, we concentrate on designing a system that tries to avoid

J. Blazewicz et al., *Handbook on Scheduling*, International Handbooks on Information Systems, https://doi.org/10.1007/978-3-319-99849-7_18

these drawbacks, i.e. we will introduce intelligence to the modeling and to the solution process of practical scheduling problems.

Later in this chapter we suggest a special DSS designed for *short term production scheduling* that works on the planning and on the control level. It makes appropriate use of scheduling theory, knowledge-based and simulation techniques. The DSS introduced will also be called "*Intelligent Production Scheduling System*" or IPS later.

This chapter is organized as follows. First we give a short idea about the environment of production scheduling from the perspective of problem solving in computer integrated manufacturing (Section 18.1). Based on this we suggest a reference model of production scheduling for enterprises (Section 18.2). Considering the requirements of a DSS for production scheduling we introduce an architecture for scheduling manufacturing processes (Section 18.3). It can be used either for an open interactive (Section 18.3.1) or a closed loop solution approach (Section 18.3.2). Based on all this we use an example of a flexible manufacturing cell to show how knowledge-based approaches and ideas relying on traditional scheduling theory can be integrated within an interactive approach (Section 18.3.3). Note that, in analogy, the discussion of all these issues can be applied to other scheduling areas than manufacturing.

18.1 Scheduling in Computer Integrated Manufacturing

The concept of *Computer Integrated Manufacturing* (CIM) is based on the idea of combining information flow from technical and business areas of a production company [Har73]. All steps of activities, ranging from customer orders to product and process design, master production planning, detailed capacity planning, predictive and reactive scheduling, manufacturing and, finally, delivery and service contribute to the overall information flow. Hence a sophisticated common database support is essential for the effectiveness of the CIM system. Usually, the database will be distributed among the components of CIM. To integrate all functions and data a powerful communication network is required. Examples of network architectures are hierarchical, client server, and loosely connected computer systems. Concepts of CIM are discussed in detail by e.g. Ranky [Ran86] and Scheer [Sch91].

We repeat briefly the main structure of CIM systems. The more technically oriented components are *Computer Aided Design* (CAD) and *Computer Aided Process Planning* (CAP), often comprised within *Computer Aided Engineering* (CAE), *Computer Aided Manufacturing* (CAM), and *Computer Aided Quality Control*(CAQ). More businesslike components are the *Production Planning System* (PPS) and the already mentioned IPS. The concept of CIM is depicted in Figure 18.1.1 where edges represent data flows in either directions. In CAD, de-

velopment and design of products is supported. This includes technical or physical calculations and drafting. CAP supports the preparation for manufacturing through process planning and the generation of programs for numeric controlled machines. Manufacturing and assembly of products are supported by CAM which is responsible for material and part transport, control of machines and transport systems, and for supervising the manufacturing process. Requirements for product quality and generation of quality review plans are delivered by CAQ. The objective of PPS is to take over all planning steps for customer orders in the sense of material requirements and resource planning. Within CIM, the IPS organizes the execution of all job- or task-oriented activities derived from customer orders.

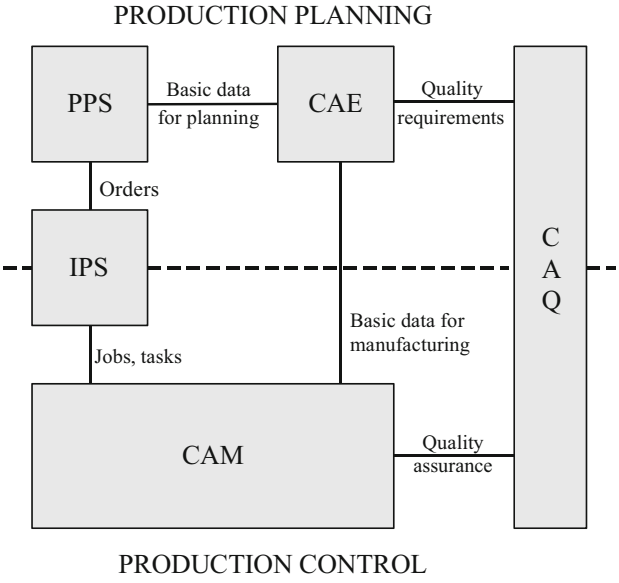


Figure 18.1.1 *The concept of CIM.*

Problems in production planning and control could theoretically be represented in a single model and then solved simultaneously. But even if all input data would be available and reliable this approach would not be applicable in general because of prohibitive computing times for finding a solution. Therefore a practical approach is to solve the problems of production planning and control sequentially using a hierarchical scheme. The closer the investigated problems are to the bottom of the hierarchy the shorter will be the time scale under consideration and the more detailed the needed information. Problems on the top of the hierarchy incorporate more aggregated data in connection with longer time scales. Decisions on higher levels serve as constraints on lower levels. Solutions for problems on lower levels give feedback to problem solutions on higher levels. The relationship between PPS, IPS and CAM can serve as an example for a hierarchy

which incorporates three levels of problem solving. It is of course obvious that a hierarchical solution approach cannot guarantee optimality. The number of levels to be introduced in the hierarchy depends on the problem under consideration, but for the type of applications discussed here a model with separated tactical (PPS), operational (IPS), and physical level (CAM) seems appropriate.

In production planning the material and resource requirements of the customer orders are analyzed, and production data such as ready times, due dates or deadlines, and resource assignments are determined. In this way, a midterm or tactical production plan based on a list of customer orders to be released for the next manufacturing period is generated. This list also shows the actual production requirements. The production plan for short term scheduling is the output of the production scheduling system IPS on an operational level. IPS is responsible for the assignment of jobs or tasks to machines, to transport facilities, and for the provision of additional resources needed in manufacturing, and thus organizes job and task release for execution. On a physical level CAM is responsible for the real time execution of the output of IPS. In that way, IPS represents an interface between PPS and CAM as shown in the survey presented in Figure 18.1.2. In detail, there are four major areas the IPS is responsible for [Sch89a].

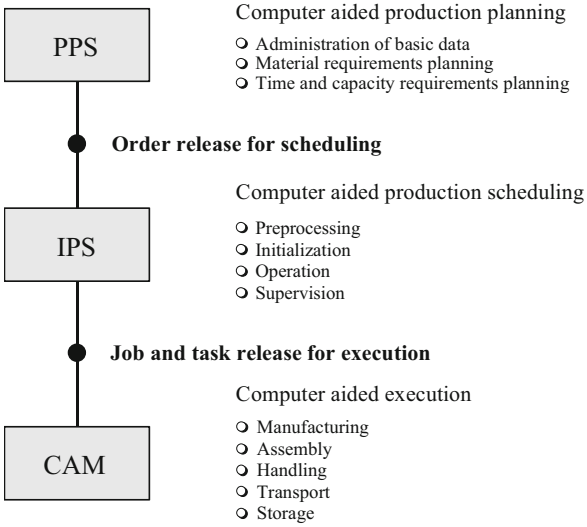


Figure 18.1.2 *Production planning, scheduling and execution.*

(1) *Preprocessing*: Examination of production prerequisites; the customer orders will only be released for manufacturing if all needed resources such as materials, tools, machines, pallets, and NC-programs are available.

(2) *System Initialization*: The manufacturing system or parts thereof have to be set up such that processing of released orders can be started. Depending on the

type of job, NC-programs have to be loaded, tools have to be mounted, and materials and equipment have to be made available at specific locations.

(3) *System Operation*: The main function of short term production scheduling is to decide about releasing jobs for entering the manufacturing system, how to traverse jobs inside the system, and how to sequence them in front of the machines in accordance with business objectives and production requirements.

(4) *System Supervision and Monitoring*: The current process data allow to check the progress of work continuously. The actual state of the system should always be observed, in order to be able to react quickly if deviations from a planned state are diagnosed.

Offline planning (OFP) is concerned with preprocessing, system initialization and system operation on a predictive level, while online control (ONC) is focused mainly on system operation on a reactive level and on system supervision and monitoring. Despite the fact that all these functions have to be performed by the IPS, following the purpose of this chapter we mainly concentrate on short term production scheduling on the predictive and the reactive level.

One of the basic necessities of CIM is an integrated database system. Although data are distributed among the various components of a CIM system, they should be logically centralized so that the whole system is virtually based on a single database. The advantage of such a concept would be redundancy avoidance which allows for easier maintenance of data and hence provides ways to assure consistency of data. This is a major requirement of the *database management system* (DBMS). The idea of an integrated data management within CIM is shown in Figure 18.1.3.

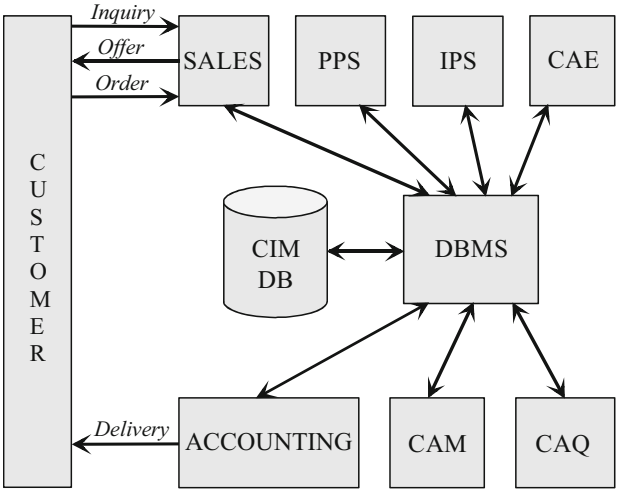


Figure 18.1.3 *CIM and the database.*

The computer architecture for CIM follows the hierarchical approach of problem solving which has already been discussed earlier in this section. The hierarchy can be represented as a tree structure that covers the following decision oriented levels of an enterprise: strategic planning, tactical planning, operational scheduling, and physical manufacturing. At each level a host computer is coordinating one or more computers on the next lower level; actions at each level are carried out independently, as long as the requirements coming from the supervising level are not violated. The output of each subordinated level meets the requirements for correspondingly higher levels and provides feedback to the host. The deeper the level of the tree is, the more detailed are the processed data and the shorter has to be the computing time; in higher levels, on the other hand, the data are more aggregated. Figure 18.1.4 shows a distributed computer architecture, where the boxes assigned to the three levels PPS, IPS and CAM represent computers or computer networks. The leaves of the tree represent physical manufacturing and are not further investigated.

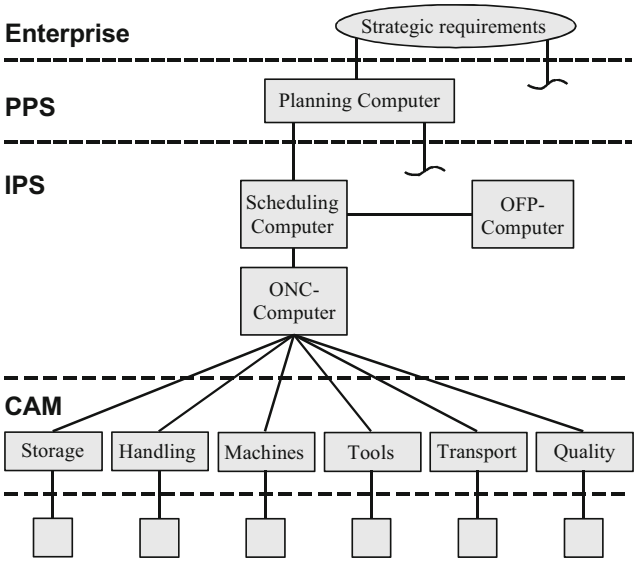


Figure 18.1.4 *Computer system in manufacturing.*

Apart from a vertical information flow, a horizontal exchange of data on the same level between different computers must be provided, especially in case of a distributed and global environment for production scheduling. Generally, different network architectures to meet these requirements may be thought of. Standard protocols and interfaces should be utilized to allow for the communication between computers from different vendors.

18.2 A Reference Model for Production Scheduling

In order to implement the solution approaches presented in the previous chapters within a framework of an IPS we need a basic description of the scheduling system. Here we introduce a modeling approach integrating declarative representation and algorithmic solution [Sch96]. Problem representation and problem solution are strongly interconnected, i.e. data structures and solution methods have to be designed interdependently [Wir76]. We will suggest a reference model for production scheduling and show how problem description and problem solution can be integrated. To achieve this we follow the object-oriented modeling paradigm.

Object-oriented modeling attempts to overcome the disadvantage of modeling data, functions, and communication, separately. The different phases of the modeling process are analysis, design, and programming. Analysis serves as the main representation formalism to characterize the requirements from the viewpoint of the application domain; design uses the results of analysis to obtain an implementation-oriented representation, and programming means translating this representation using some programming language into code. Comparing object-oriented modeling with traditional techniques its advantages lie in data abstraction, reusability and extensibility of the model, better software maintenance, and direct compatibility of the models of different phases of the software development process. Often it is also claimed that this approach is harmonizing the decentralization of organizations and their support by information systems. We will now develop an open object-oriented analysis model for production scheduling. In comparison to other models of this kind (see e.g. [RM93]) the model presented here is a one to one mapping of the classification scheme of deterministic scheduling theory introduced in Chapter 3 to models of information systems for production scheduling. Following this approach we hope to achieve a better transformation of theoretical results to practical applications.

A model built by object-oriented analysis consists of a set of objects communicating via messages which represent dynamic relations of pairs of them. Each object consists of attributes and methods here also called algorithms. Methods are invoked by messages and methods can also create messages themselves. Objects of the same type are classified using the concept of classes; with this concept inheritance of objects can be represented. The main static relations between pairs of objects are generalization/specialization and aggregation/decomposition.

Different methods for generating object-oriented models exist [DTLZ93], [WBJ90]. From a practical point of view the method should make it easy to develop and maintain a system; it should assist project management by defining deliverables and effective tool support should be available. Without loss of gen-

erality the object model for production scheduling which will be introduced here is based on the modeling approach called Object-Oriented Analysis or OOA suggested by [CY91]. It is easy to use, easy to understand, and fulfils most of the above mentioned criteria.

In Figure 18.2.1 the main classes and objects for production scheduling are represented using OOA notation. Relationships between classes or objects are represented by arcs and edges; edges with a semi-circle represent generalization/specialization relations, edges with triangles represent aggregation/decomposition, and arcs between objects represent communications by message passing. The arc direction indicates a transmitter/receiver relationship. The introduced classes, objects, attributes, methods, and relations are complete in the sense that applying the proposed model a production schedule can be generated; nevertheless it is easy to enlarge the model to represent additional business requirements.

Each customer order is translated into a manufacturing order, also called job, using process plans and bill of materials. Without loss of generality we want to assume that a job refers always to the manufacturing of one part where different tasks have to be carried out using different resources.

While in Figure 18.2.1 a graphical notation related to OOA is used, we will apply in the following a textual notation. We will denote the names of classes and objects by capital letters, the names of attributes by dashes, and the names of methods by brackets. In OOA notation relationships between classes or objects will be represented by arcs and edges; edges with a semi-circle represent generalization/specialization relations, edges with triangles represent aggregation, and arcs represent communications between objects by message passing. The direction of the arc indicates a transmitter-receiver relationship. The introduced classes, objects, attributes, methods, and relations are complete in the sense that applying the proposed model a production schedule can be generated; nevertheless it is easy to enlarge the model to represent additional business requirements.

The main classes of production scheduling are JOB, BOM (BILL_OF_MATERIALS), PP (PROCESS_PLAN), TASK, RESOURCE, and SCHEDULE. Additional classes are ORDER specialized to PURCHASING_ORDER and DISPATCH_ORDER and PLANNING specialized to STRATEGIC_P, TACTICAL_P, and OPERATIONAL_P. The class RESOURCE is a generalization of MACHINE, TOOL, and STAFF. Without loss of generality we concentrate the investigation here only on one type of resources which is MACHINE; all other types of resources could be modeled in the same manner. In order to find the attributes of the different classes and objects we use the classification scheme introduced in Chapter 3.

The objects of class BOM generate all components or parts to be produced for a customer order. With this the objects of class JOB will be generated. Each object of this class communicates with the corresponding objects of class PP which includes a list of the technological requirements to carry out some job. According to these requirements all objects of class TASK will be generated, which are necessary to process all jobs.

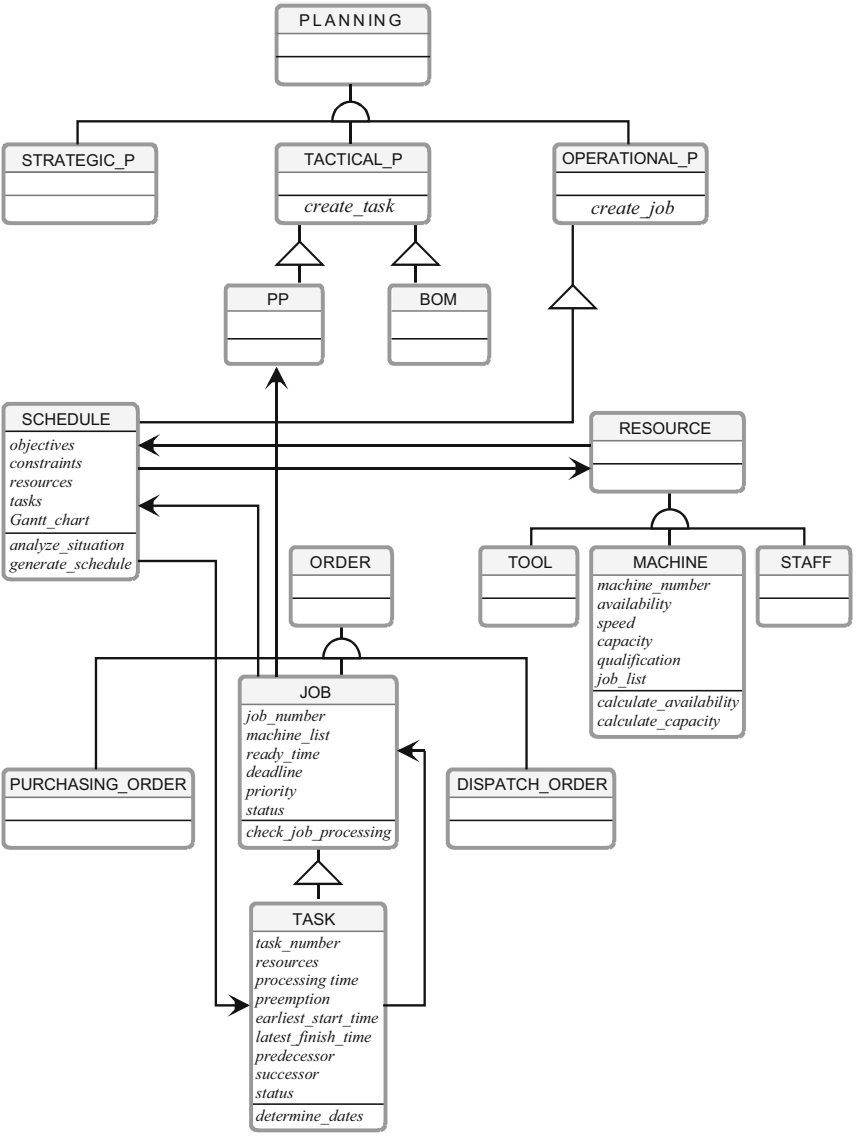


Figure 18.2.1 *Object-oriented analysis model for production scheduling.*

An object of class JOB is characterized by the attributes "job_number", "machines", "machine_list", "ready_time", "deadline", "completion_time", "flow_time", "priority", and "status". Some values of the attributes concerning time and priority considerations are determined by the earlier mentioned Production Planning System (PPS). The value of the attribute "machines" refers to these ma-

chines which have the qualification to carry out the corresponding job; after generating the final production schedule the value of "machine_list" refers to the ordered number of these machines to which the job is assigned. The value of the attribute "status" gives an answer to the question if the job is open, scheduled, or finished. The method used by JOB is here <check_job_processing> which has the objective to supervise the progress of processing the job. Communication between JOB and SCHEDULE results in determining the values of "machine_list", "completion_time", "flow_time" and "status".

Each object of class TASK contains structural attributes like "task_number", "resources", "processing_time", "completion_time", "finish_time", "preemption", "earliest_start_time", "latest_finish_time" and additional attributes like "predecessor", "successor", and "status". The values of the two attributes referring to earliest start and latest finish time are determined by the object-owned method <determine_dates>. The parameters for this method are acquired by communication with objects of the class JOB. Again the attribute "status" is required for analyzing the current state of processing of the task under consideration.

Objects of class MACHINE are described by the attributes "machine_number", "availability", "speed", "capacity", "qualification", and "job_list". The value of "qualification" is the set of tasks which can be carried out by the machine. The value of "job_list" is unknown at the beginning; after generating the schedule the value refers to the set of jobs and corresponding tasks to be processed by this machine. In the same sense the values of "availability" and "capacity" will be altered using the methods <calculate_availability> and <calculate_capacity>.

The task of the object SCHEDULE is to generate the final production schedule. In order to do this the actual manufacturing situation has to be analyzed in terms of objective function and constraints to be considered. This leads to the determination of the values for the attributes "objectives" and "constraints" using the method <analyze_situation>. The method <generate_schedule> is constructing the desired schedule. Calling this method the communication links to the objects of classes RESOURCE, JOB, and TASK respectively, are activated. To the attributes "resources" and "tasks" the input values for <generate_schedule> are assigned. The result of the method is a depiction of the production schedule which is assigned to the attribute "Gantt_chart". The required data concerning tasks and resources like machines, availability, speed, processing times etc. are available through the communication links to the objects of classes TASK and RESOURCE.

Example 18.2.1 The following example shows how an object-oriented model for production scheduling can be generated. When we refer to the objects of a particular class the first time we declare the name of the corresponding object, its attributes, and the value of the attributes. Later, we only declare the name of the object and the value of the attributes. All entries are abbreviated.

```
JOB1          "j_no"        J1;
              "machines"    P1, P2;
              "mach_list"   open;
              "ready"       0;
              "deadline"    open;
              "prio"        none;
              "stat"        open;
JOB2 (J2; P2; open; 0; open; none; open)
JOB3 (J3; P1; open; 2; open; none; open)
```

There are three jobs which have to be processed. No given sequence for J_1 exists but J_2 can only be processed on P_2 and J_3 can only be processed on P_1. The jobs can start to be processed at times 0 and 2; there is no deadline which has to be obeyed, all jobs have the same priority. The machine list and status of the jobs are open at the beginning; later they will assume the values of the permutation of the machines and scheduled, in_process, or finished, respectively.

```
TASK11        "t_no"        T11;
              "res"         P1, P2;
              "p_time"      3;
              "preempt"     no;
              "e_s_t"       0;
              "l_f_t"       open;
              "pre"         ∅;
              "suc"         T12, T13;
              "stat"        open;
TASK12 (T12; P1, P2; 13; no; 3; open; T11; ∅; open)
TASK13 (T13; P1, P2; 2; no; 3; open; T11; ∅; open)
TASK20 (T20; P2; 4; no; 0; open; ∅; ∅; open)
TASK31 (T31; P1; 2; no; 2; open; ∅; T32, T33, T34; open)
TASK32 (T32; P1; 4; no; 4; open; T31; ∅; open)
TASK33 (T33; P1; 4; no; 4; open; T31; ∅; open)
TASK34 (T34; P1; 2; no; 4; open; T31; ∅; open)
```

The three jobs consist of eight tasks; all tasks of job J_1 can processed on all machines, all other tasks are only allowed to be processed on machine P_2 or only on machine P_1. Processing times, precedence constraints and ready times are known, preemption is not allowed, and again deadlines do not exist. The status of the tasks is open at the beginning; later it will also assume the values scheduled, in_process, or finished.

```
MACHINE1      "m_no"        P1;
              "avail"       [0,∞);
              "speed"       1;
```

```
                "capac"        PC1;
                "qualif"       T11, T12, T13, T31, T32, T33, T34;
                "j_list"       open;
MACHINE2 (P2; [0, ∞); 1; PC2; T11, T12, T13, T20; open)
```

There are two machines available for processing. Both machines have the same speed. They are available throughout the planning horizon, capacity and qualification are known. The job list, i.e. the sequence the jobs are processed by the machines is not yet determined.

```
SCHEDULE        "object"       makespan;
                "constr"       open;
                "res"          P1, P2;
                "tasks"        T11, T12, T13, T31, T32, T33, T34;
                "Gantt_chart"  open;
```

The objective here is to minimize the makespan, i.e. to find a schedule where $\max\{C_i\}$ is minimized. Besides task and machine related constraints no other constraints have to be taken into account. All input data to generate the desired production schedule is given, the schedule itself is not yet known. Calling the method <generate_schedule> will result in a time oriented assignment of tasks to machines. Doing this the attributes will assume the following values.

```
JOB1 (J1; P1, P2; 0; 17; none; scheduled)
JOB2 (J2; P2; 0; 4; none; scheduled)
JOB3 (J3; P1; 3; 15; none; scheduled)

TASK11 (T11; P1; 3; no; 0; 3; ∅; T12, T13; scheduled)
TASK12 (T12; P2; 13; no; 4; 17; T11; ∅; scheduled)
TASK13 (T13; P1; 2; no; 15; 17; T11; ∅; scheduled)
TASK20 (T20; P2; 4; no; 0; 4; ∅; ∅; scheduled)
TASK31 (T31; P1; 2; no; 3; 5; ∅; T32, T33, T34; scheduled)
TASK32 (T32; P1; 4; no; 5; 9; T31; ∅; scheduled)
TASK33 (T33; P1; 4; no; 9; 13; T31; ∅; scheduled)
TASK34 (T34; P1; 2; no; 13; 15; T31; ∅; scheduled)
```

All jobs and the corresponding tasks are now scheduled; job J_1 will be processed on machines P_1 and P_2 within the time interval [0,17], job J_2 on machine P_2 in the interval [0,4] and job J_3 on machine P_1 in the interval [3,15].

MACHINE1 (P_1; [17,∞); 1; PC_1; T_{11}, T_{12}, T_{13}, T_{31}, T_{32}, T_{33}, T_{34}; T_{11}, T_{31}, T_{32}, T_{33}, T_{34}, T_{13})
MACHINE2 (P_2; [17,∞); 1; PC_2; T_{11}, T_{12}, T_{13}, T_{20}; T_{20}, T_{12})

The availability of machines P_1 and P_2 has now been changed. Machine P_1 is processing tasks T_{11}, T_{13}, and all tasks of job J_3, machine P_2 is processing tasks T_{20} and T_{12}. The processing sequence is also given.

SCHEDULE "object" makespan;
"constr" open;
"res" P_1, P_2;
"tasks" T_{11}, T_{12}, T_{13}, T_{31}, T_{32}, T_{33}, T_{34};
"Gantt_chart" generated;

The schedule has now been generated and is depicted by a Gantt chart shown in Figure 18.2.2. adaptation □

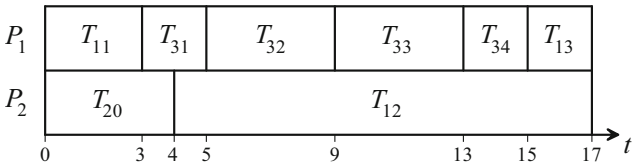


Figure 18.2.2 *Gantt chart for the example problem.*

Example 18.2.2 We now want to use the classical job shop scheduling problem as an example to show how the approach can be applied to dedicated models. Here we will concentrate especially on the interaction between problem representation and problem solution. The general job shop problem is treated in Chapter 8. The object model is characterized by the classes JOB, TASK, MACHINE and SCHEDULE. Investigating attributes of the objects we only concentrate on some selection of them. The class JOB can be described as follows.

JOB "j_no" J_j;
"machines " Permutation over P_i;
"mach_list" open;
"ready" 0;
"deadline" open;
"prio" none;
"stat" open;

As we are investigating a simple job shop problem each job is assigned to all machines following some pre-specified sequence, ready times for all jobs are zero; deadlines and priorities have not to be considered.

Each job consists of different tasks which are characterized by the machine where the task has to be processed and the corresponding processing time;

preemption is not allowed. Each task can be described by its predecessor or successor task. Input data for the algorithm are the values of the attributes "res", "p_time", "pre" and "suc". The values of "e_s_t" are not obligatory because they can be derived from the values of the attributes "pre" and "suc". With this the class TASK can be described as follows.

TASK	"t_no"	T_{ij};
	"res"	P_i;
	"p_time"	p_{ij};
	"preempt"	no;
	"e_s_t"	r_{ij};
	"l_f_t"	open;
	"pre"	T_{kj};
	"suc"	T_{lj};
	"stat"	open;
MACHINE	"m_no"	P_i;
	"avail"	$[0,\infty)$;
	"speed"	1;
	"capac"	PC_i;
	"qualif"	T_{ij};
	"j_list"	open;

All machines are continuously available in the planning period under consideration. The value of the attribute "capac" is not necessary to apply the algorithm, it is only introduced for completeness reasons.

SCHEDULE	"object"	makespan;
	"constr"	open;
	"res"	$P_1, \ldots, P_m$;
	"tasks"	T_{ij};
	"Gantt_chart"	open;
	<generate_schedule>	simulated annealing; □

The objective is again to find a production schedule which minimizes the maximum completion time. Additional information for describing the scheduling situation is not available. The input data for the algorithm are the available machines, the processing times of all jobs on all machines and the corresponding sequence of task assignment. After the application of an appropriate algorithm (compare to Chapter 8) the corresponding values describing the solution of the scheduling problem are assigned to the attributes and the Gantt chart will be generated.

We have shown using some examples that the object-oriented model can be used for representing scheduling problems which correspond to those investigated in the theory of scheduling. It is quite obvious that the model can be specified

to various individual problem settings. Thus we can use it as some reference for developing production scheduling systems.

18.3 IPS: An Intelligent Production Scheduling System

The problems of short term production scheduling are highly complex. This is not only caused by the inherent combinatorial complexity of the scheduling problem but also by the fact that input data are dynamic and rapidly changing. For example, new customer orders arrive, others are cancelled, or the availability of resources may change suddenly. This lack of stability requires permanent revisions, and previous solutions are due to continuous adaptations. Scheduling models for manufacturing processes must have the ability to partially predict the behavior of the entire shop, and, if necessary, to react quickly by revising the current schedule. Solution approaches to be applied in such an environment must have especially short computing times, i.e. time- and resource-consuming models and methods are not appropriate on an operational level of production scheduling.

All models and methods for these purposes so far developed and partially reviewed in the preceding chapters are either of descriptive or of constructive nature. Descriptive models give an answer to the question "*what happens if ...?*", whereas constructive models try to answer the question "*what has to happen so that ...?*". Constructive models are used to find best possible or at least feasible solutions; descriptive models are used to evaluate decision alternatives or solution proposals, and thus help to get a deeper insight into the problem characteristics. Examples of descriptive models for production scheduling are queuing networks on an analytical and discrete simulation on an empirical basis; constructive models might use combinatorial optimization techniques or knowledge of human domain experts.

For production scheduling problems one advantage of descriptive models is the possibility to understand more about the dynamics of the manufacturing system and its processes, whereas constructive models can be used to find solutions directly. Coupling both model types the advantages of each would be combined. The quality of a solution generated by constructive models could then be evaluated by descriptive ones. Using the results, the constructive models could be revised until an acceptable schedule is found. In many cases there is not enough knowledge available about the manufacturing system to build a constructive model from the scratch. In such situations descriptive models can be used to get a better understanding of the relevant problem parameters.

From another perspective there also exist approaches trying to change the system in order to fit into the scheduling model, others simplify the model in order to permit the use of a particular solution method. In the meantime more

model realism is postulated. Information technology should be used to model the problem without distortion and destruction. In particular it can be assumed that in practical settings there exists not only one scheduling problem all the time and there is not only one solution approach to each problem, but there are different problems at different points in time. On the other hand the analysis of the computational complexity of scheduling problems gives also hints how to simplify a manufacturing process if alternatives for processing exist.

Short term production scheduling is supported by shop floor information systems. Using data from an aggregated production plan a detailed decision is made in which sequence the jobs are released to the manufacturing system, how they traverse inside the system, and how they are sequenced in front of the machines. The level of shop floor scheduling is the last step in which action can be taken on business needs for manufacturing on a predictive and a reactive level.

One main difference between these two scheduling levels is the liability of the input data. For predictive scheduling input data are mainly based on expectations and assumptions. Unforeseen circumstances like rush orders, machine breakdowns, or absence of employees can only be considered statistically, if at all. This situation is different in reactive scheduling where actual data are available. If they are not in coincidence with the estimated data, situation-based revisions of previous decisions have to be made. Predictive scheduling has to go hand in hand with reactive scheduling.

Shop floor information systems available commercially today are predominately data administration systems. Moreover, they collect and monitor data available from machines and the shop floor. Mainly routine operations are carried out by the shop floor system; the production manager is supported by offering the preliminary tools necessary for the development of a paperless planning and control process. Additionally, some systems are also offering various scheduling strategies but with limited performance and without advice when to apply them. It can be concluded that the current shop floor information systems are good at data administration, but for the effective solution of production scheduling problems they are of very little help [MS92a, MS92b].

An intuitive job processing schedule, based solely upon the experience of skilled production managers, does not take advantage of the potential strengths of an integrated IPS. Thus, the development of an intelligent system which integrates planning and control within scheduling for the entire operation and supports effectively the shop floor management, becomes necessary. Such a system could perform all of the functions of the current shop floor scheduling systems and would also be able to generate good proposals for production schedules, which also take deviations from the normal routine into consideration. With the help of such concepts the problems involved in initializing and operating a manufacturing system should be resolved.

Practical approaches to production scheduling on the planning and control level must take also into account the dynamic and unpredictable environment of the shop floor. Due to business and technical considerations, most decisions must

be made before all the necessary information has been gathered. Production scheduling must be organized in advance. Predictive scheduling is the task of production planning and the basis for production control; where reactive scheduling has to be able to handle unexpected events. In such a situation, one attempt is to adapt to future developments using a chronological and functional hierarchy within the decision making steps of production scheduling. This helps to create a representation of the problem that considers all available information [Sch89a].

The chronological hierarchy leads to the separation of offline planning (OFP) and online control (ONC). Problems involved in production scheduling are further separated on a conceptual and a specific level in order to produce a functional hierarchy, too. The purpose of the chronological approach to prioritization is to be able to come to a decision through aggregated and detailed modeling, even if future information is unspecific or unavailable. Aside from fulfilling the functional needs of the organization, the basic concept behind the functional hierarchy is to get a better handle on the combinatorial difficulties that emerge from the attempt of simultaneously solving all problems arising in a manufacturing environment. The IPS should follow hierarchical concepts in both, the chronological and the functional aspect. The advantage of such a procedure consists not only in getting a problem-specific approach for investigation of the actual decision problem, but also in the representation of the decision making process within the manufacturing organization.

Models and methods for the hierarchically structured scheduling of production with its planning and control parts have been developed over the years and are highly advanced; see e.g. [KSW86, Kus86, Ste85, LGW86]. However, they lack integration in the sense of providing a concept, which encompasses the entire planning and control process of scheduling. With our proposal for an IPS we try to bring these methods and models one step closer to practical application. The rudimentary techniques of solving predictive scheduling problems presented here work on a closed Analysis-Construction-Evaluation loop (ACE loop). This loop has a feedback mechanism creating an IPS on the levels of OFP and ONC [Sch92]. An overview over the system is shown in Figure 18.3.1.

The OFP module consists of an analysis, a construction and an evaluation component. First, the problem instance is analyzed (A) in terms of objectives, constraints and further characteristics. In order to do this the first step for (A) is to describe the manufacturing environment with the scheduling situation as detailed as necessary. In a second step from this description a specific model has to be chosen from a set of scheduling models in the library of the system. The analysis component (A) can be based upon knowledge-based approaches, such as those used for problems like classification.

The problem analysis defines the parameters for the construction (C) phase. From the basic model obtained in (A), a solution for the scheduling problem is generated by (C) using some generic or specific algorithms. The result is a complete schedule that has then to be evaluated by (E). Here the question has to be answered if the solution can be implemented in the sense that manufacturing ac-

cording to the proposed solution meets business objectives and fulfils all constraints coming from the application. If the evaluation is satisfactory to the user, the proposed solution will be implemented. If not, the process will repeat itself until the proposed solution delivers a desirable outcome or no more improvements appear to be possible in reasonable time.

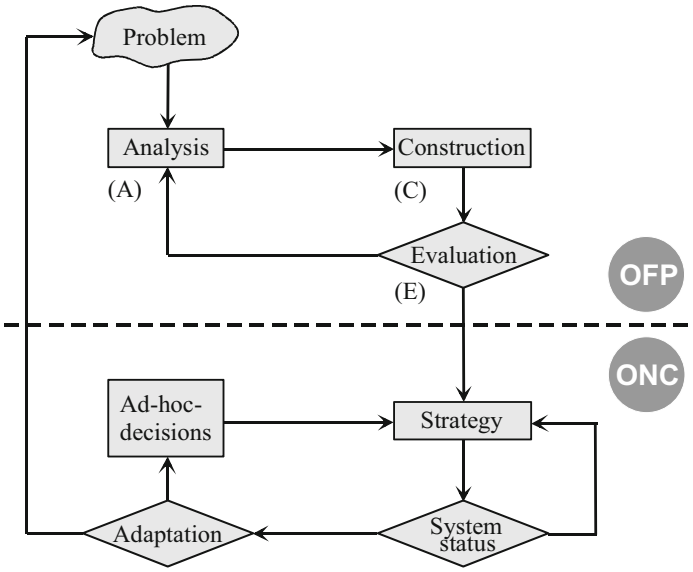


Figure 18.3.1 *Intelligent problem solving in manufacturing.*

The construction component (C) of the ACE loop generates solutions for OFP. It bases its solution upon exact and heuristic problem solving methods. Unfortunately, with this approach we only can solve static representations of quite general problems. The dynamics of the production process can at best be only approximately represented. In order to obtain the necessary answers for a dynamic process, the evaluation component (E) builds up descriptive models in the form of queuing networks at aggregated levels [BY86] or simulation on a specific level [Bul82, Ca86]. With these models one can evaluate the various outcomes and from this if necessary new requirements for problem solution are set up.

Having generated a feasible and satisfactory predictive schedule the ONC module will be called. This module takes the OFP schedule and translates its requirements to an ONC strategy, which will be followed as long as the scheduling problem on the shop floor remains within the setting investigated in the analysis phase of OFP. If temporary disturbances occur, a time dependent strategy in the form of an ad-hoc decision must be devised. If the interruption continues for such a long time that a new schedule needs to be generated, the system will return to the OFP module and seek for an alternative strategy on the basis of a new problem instance with new requirements and possibly different objectives within

the ACE loop. Again a new ONC strategy has to be found which will then be followed until again major disturbances occur.

As already mentioned, production scheduling problems are changing over time; a major activity of the problem analysis is to characterize the problem setting such that one or more scheduling problems can be modeled and the right method or a combination of methods for constructing a solution can be chosen from a library of scheduling methods or from knowledge sources coming from different disciplines. With this there are three things to be done; first the manufacturing situation has to be described, second the underlying problem has to be modeled and third an appropriate solution approach has to be chosen. From this point of view one approach is using expert knowledge to formulate and model the problem using the reference model presented in the preceding section, and then using "deep"-knowledge from the library to solve it.

The function of OFP is providing flexibility in the development and implementation of desirable production schedules. OFP applies algorithms which can either be selected from the library or may also be developed interactively on the basis of simulation runs using all components of the ACE loop. The main activity of the interaction of the three components of the loop is the resolution of conflicts between the suggested solution and the requirements coming from the decision maker. Whenever the evaluation of some schedule generated by (C) is not satisfactory then there exists at least some conflict between the requirements or business objectives of a problem solution and the schedule generated so far. Methods to detect and resolve these conflicts are discussed in the next section.

The search for a suitable strategy within ONC should not be limited to routine situations, rather it should also consider e.g. breakdowns and their predictable consequences. ONC takes into consideration the scheduling requirements coming from OFP and the current state of the manufacturing system. To that end, it makes the short term adjustments, which are necessary to handle failures in elements of the system, the introduction of new requirements for manufacturing like rush orders or the cancellation of jobs. An algorithmic reaction on this level of problem solving based on sophisticated combinatorial considerations is generally not possible because of prohibitive computing times of such an approach. Therefore, the competence of human problem solvers in reaching quality, real-time decisions is extremely important.

OFP and ONC require suitable diagnostic experience for high quality decision making. Schedules generated in the past should be recorded and evaluated, for the purpose of using this experience to find solutions for actual problems to be solved. Knowledge-based systems, which could be able to achieve the quality of "self-learning" in the sense of case-based reasoning [Sch98], can make a significant contribution along these lines.

Solution approaches for scheduling problems mainly come from the fields of Operations Research (*OR*) and Artificial Intelligence (*AI*). In contrast to OR-approaches to scheduling, which are focused on *optimization* and which were mainly covered in the preceding chapters, AI relies on *satisfaction*, i.e. it is suffi-

cient to generate solutions which are accepted by the decision maker. Disregarding the different paradigm of either disciplines the complexity status of the scheduling problems remains the same, as it can be shown that the decision variant of a problem is not easier than the corresponding optimization problem (see Section 2.2). Although the OR- and AI-based solution approaches are different, many efforts of either disciplines for investigating scheduling problems are similar; examples are the development of priority rules, the investigation of bottleneck resources and constraint-based scheduling. With priority scheduling as a job- or task-oriented approach, and with bottleneck scheduling as a resource-oriented one, two extremes for rule-based schedule generation exist.

Most of the solution techniques can be applied not only for predictive but also for reactive scheduling. Especially for the latter case priority rules concerning job release to the system and job traversing inside the system are very often used [BPH82, PI77]. Unfortunately, for most problem instances these rules do not deliver best possible solutions because they belong to the wide field of *heuristics*. Heuristics are trying to take advantage from special knowledge about the characteristics of the domain environment or problem description respectively and sometimes from analyzing the structure of known good solutions. Many AI-based approaches exist which use domain knowledge to solve predictive and reactive scheduling problems, especially when modeled as constraint-based scheduling.

OR approaches are built on numerical constraints, the AI approach is considering also non-numerical constraints distinguishing between *soft* and *hard constraints*. In this sense scheduling problems also can be considered as *constraint satisfaction problems* with respect to hard and soft constraints. Speaking of hard constraints we mean constraints which represent necessary conditions that must be obeyed. Among hard constraints are given precedence relations, routing conditions, resource availability, ready times, and setup times. In contrast to these, soft constraints such as desirable precedence constraints, due dates, work-in-process inventory, resource utilization, and the number of tool changes, represent rather *preferences* the decision maker wants to be considered. From an OR point of view they represent the aspect of optimization with respect to an objective function. Formulating these preferences as constraints too, will convert the optimization problem under consideration into a feasibility or a decision problem. In practical cases it turns out very often that it is less time consuming to decide on the feasibility of a solution than to give an answer to an optimization problem.

The *constraint satisfaction problem* (*CSP*) deals with the question of finding values for the variables of a set $\mathcal{X} = \{x_1, \cdots, x_n\}$ such that a given collection $\mathcal{C}$ of constraints $c_1, \cdots, c_m$ is satisfied. Each variable x_i is assigned a domain z_i which defines the set of values x_i may assume. Each constraint is a subset of the Cartesian product $z_1 \times z_2 \times \cdots \times z_n$ that specifies conditions on the values of the varia-

bles $x_1, \cdots, x_n$. A subset $\mathcal{Y} \subseteq z_1 \times z_2 \times \cdots \times z_n$ is called a *feasible solution* of the constraint satisfaction problem if $\mathcal{Y}$ meets all constraints of C, i.e. if $\mathcal{Y} \subseteq \bigcap_{j=1}^{n} c_j$.

The analysis of a constraint satisfaction problem either leads to feasible solutions or to the result that for a given constraint set no such solution exists. In the latter case *conflict resolution* techniques have to be applied. The question induced by a constraint satisfaction problem is an NP-complete problem [GJ79] and one of the traditional approaches to solve it is backtracking. In order to detect *unfeasibility* it is sometimes possible to avoid this computationally expensive approach by carrying out some preprocessing steps where conflicts between constraints are detected in advance.

Example 18.3.1 For illustration purposes consider the following example problem with $\mathcal{X} = \{x_1, x_2, x_3\}$, $z_1 = z_2 = z_3 = \{0, 1\}$, and $C = \{c_1, c_2, c_3\}$ representing the constraints

$$x_1 + x_2 = 1 \tag{18.3.1}$$

$$x_2 + x_3 = 1 \tag{18.3.2}$$

$$x_1 + x_3 = y \quad \text{for } y \in \{0, 2\}\,. \tag{18.3.3}$$

Feasible solutions for this example constraint satisfaction problem are given by $\mathcal{Y}_{11} = \{(0, 1, 0)\}$ and $\mathcal{Y}_{12} = \{(1, 0, 1)\}$. If a fourth constraint represented by

$$x_2 + x_3 = 0 \tag{18.3.4}$$

is added to C, conflicts arise between (18.3.2) and (18.3.4) and between (18.3.1), (18.3.3), and (18.3.4). From these we see that no feasible solution exists. Notice that no backtracking approach was needed to arrive at this result. □

To solve constraint satisfaction problems most AI scheduling systems construct a search tree and apply some search technique to find a feasible solution. A common technique to find feasible solutions quickly is constraint directed search. The fundamental philosophy uses a priori *consistency checking techniques* [DP88, Fre78, Mac77, Mon74]. The basic concept is to prune the search space before unfeasible combinations of variable values are generated. This technique is also known as *constraint propagation*.

Apart from the discussed focus on constraints, AI emphasizes the role of domain specific knowledge in decomposing the initial problem according to several perspectives like bottleneck resources, hierarchies of constraints, conflicting subsets of constraints, while ignoring less important details. Existing AI-based scheduling systems differentiate between *knowledge representation* (models) and *scheduling methodology* (algorithms). They focus rather on a particular application than on general problems. The scheduling knowledge refers to the manufacturing system itself, to constraints and to objectives or preferences. Possible rep-

resentation techniques are semantic networks (declarative knowledge), predicate logic (especially for constraints), production rules (procedural knowledge) and frames (all of it). Scheduling methodology used in AI is mainly based on production rules (operators), heuristic search (guides the application of operators), opportunistic reasoning (different views of problem solving, e.g. resource-based or job-based), hierarchical decomposition (sub-problem solution, abstraction and distributed problem solving), pattern matching (e.g. using the status of the manufacturing system and given objectives for the application of priority rules), constraint propagation, reinforcement or relaxation techniques.

In the next three sections we describe two approaches which use AI-based solution techniques to give answers to production scheduling problems. In Section 18.3.1 we demonstrate open loop interactive scheduling and in Section 18.3.2 we discuss some closed loop approaches using expert knowledge in the solution process of scheduling problems. In Section 18.3.3 we present an example for integrated problem solving combining OR- and AI-based solution approaches.

18.3.1 Interactive Scheduling

We now want to describe how a constraint-based approach can be used within the ACE-loop to solve predictive scheduling problems interactively. Following Schmidt [Sch89b], decomposable problems can be solved via a heuristic solution procedure based on a hierarchical "relax and enrich" strategy (*REST*) with look ahead capabilities. Using *REST* we start with a solution of some relaxed feasibility problem considering hard constraints only. Then we enrich the problem formulation step by step by introducing preferences from the decision maker. These preferences can be regarded as soft constraints. We can, however, not expect in general that these additional constraints can be met simultaneously, due to possible conflicts with hard constraints or with other preferences. In this case we have to analyze all the preferences by some *conflict detection* procedure. Having discovered conflicting preferences we must decide which of them should be omitted in order to *resolve contradictions*. This way a feasible and acceptable solution can be generated.

REST appears to be appealing in a production scheduling environment for several reasons. The separation of hard constraints from preferences increases scheduling flexibility. Especially, preferences very often change over time so that plan revisions are necessary. If *relaxation* and *enrichment* techniques are applied, only some preferences have to be altered locally while very often major parts of the present schedule satisfying hard constraints can be kept unchanged. A similar argument applies for acceptable partial schedules which may be conserved and the solution procedure can concentrate on the unsatisfactory parts of the schedule only.

This problem treatment can be incorporated into the earlier mentioned DSS framework for production scheduling which then includes an *algorithmic* module to solve the problem under the set of hard constraints, and a *knowledge-based* module to take over the part of conflict detection and implementation of consistent preferences. Without loss of generality and for demonstration purposes only we want to assume in the following that the acceptability of a solution is the greater the more preferences are incorporated into the final schedule. For simplicity reasons it is assumed that all preferences are of equal importance.

In this section we describe the basic ideas of *REST* quite generally and demonstrate its application using an example from precedence constrained scheduling. We start with a short discussion of the types of constraints we want to consider. Then we give an overview on how to detect conflicts between constraints and how to resolve them. Finally, we give a simple example and present the working features of the scheduling system based on *REST*.

Analyzing Conflicts

Given a set of tasks $\mathcal{T} = \{T_1, \cdots, T_n\}$, let us assume that preferences concern the order in which tasks are processed. Hence the set of preferences $\mathcal{PR}$ is defined as a subset of the Cartesian product, $\mathcal{T} \times \mathcal{T}$. *Conflicts* occur among contradictory constraints. We assume that the given hard constraints are not contradictory among themselves, and hence that and thus a feasible schedule that obeys all the hard constraints always exists. Obviously, conflicts can only be induced by the preferences. Then, two kinds of contradictions have to be taken into account: conflicts between the preferences and the hard constraints, and conflicts among preferences themselves. Following the strategy of *REST* we will not extract all of these conflicts in advance. We rather start with a feasible schedule and aim to add as many preferences as possible to the system.

The conflicting preferences are mainly originated from desired task orderings, time restrictions and limited resource availabilities. Consequently, we distinguish between logically conflicting preferences, time conflicting preferences, and resource conflicting preferences.

Logical conflicts between preferences occur if a set of preferred task orderings contains incompatible preferences. Logical conflicts can easily be detected by investigating the directed graph $G = (\mathcal{T}, \mathcal{PR})$. This analysis can be carried out by representing the set of preferences as a directed graph $G = (\mathcal{T}, \mathcal{LC})$ where $\mathcal{T}$ is the set of tasks and $\mathcal{LC} \subseteq \mathcal{T} \times \mathcal{T}$ represents the preferred processing orders among them.

Example 18.3.2 To illustrate the approach we investigate an example problem where a set $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$ of four tasks has to be scheduled. Let the pre-

ferred task orderings be given by $\mathcal{PR} = \{PR_1, PR_2, PR_3, PR_4, PR_5\}$ with $PR_1 = (T_1, T_2)$, $PR_2 = (T_2, T_3)$, $PR_3 = (T_3, T_2)$, $PR_4 = (T_3, T_4)$, and $PR_5 = (T_4, T_1)$.

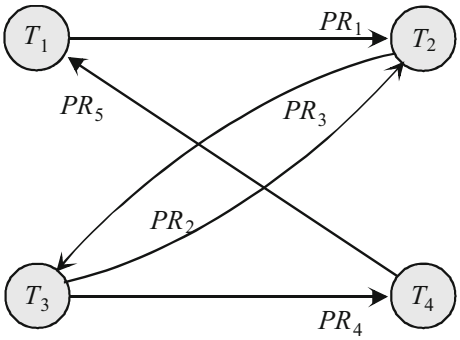


Figure 18.3.2 $G = (\mathcal{T}, \mathcal{PR})$ *representing preferences in Example* 18.3.2.

Logical conflicts in $G = (\mathcal{T}, \mathcal{PR})$ can be detected by finding all cycles of G (see Figure 18.3.2). From this we get two sets of conflicts, $\mathcal{LC}_1 = \{PR_2, PR_3\}$ and $\mathcal{LC}_2 = \{PR_1, PR_2, PR_4, PR_5\}$. □

Time conflicts occur if a set of preferences is not consistent with time restrictions following from the initial solution obtained on the basis of the hard constraints. To detect time conflicts we must explicitly check all time conditions between the tasks. Hard constraints implying earliest beginning times EB_j, latest beginning times LB_j and processing times p_j restrict the preferences that can be realized. So, if

$$EB_u + p_u > LB_v \tag{18.3.5}$$

for tasks T_u and T_v, the preference (T_u, T_v), would violate the time restrictions. More generally, suppose that for some $k \in \mathbb{N}$ and for tasks $T_{u_1}, \cdots, T_{u_k}$ and T_u there are preferences (T_{u_1}, T_{u_2}), $(T_{u_2}, T_{u_3}), \cdots, (T_{u_{k-1}}, T_{u_k})$ and (T_{u_k}, T_v). These preferences imply that the tasks should be processed in order $(T_{u_1}, \cdots, T_{u_k}, T_v)$. However, if this task sequence has the property

$$Z_{u_k} + p_{u_k} > LB_v \tag{18.3.6}$$

where

$$Z_{u_k} = \max\left\{EB_{u_k}, \max_l\{EB_{u_l} + \sum_{j=l}^{k-1} p_j\}\right\}$$

then obviously the given set of preferences is conflicting. If (18.3.6) is true the time constraint coming from the last task of the chain will be violated.

Example 18.3.2 - continued - To determine time conflicts, assume that each task T_j has a given earliest beginning time EB_j and a latest beginning time LB_j as specified together with processing times p_j in the following Table 18.3.1.

T_j	EB_j	LB_j	p_j
T_1	7	7	5
T_2	3	12	4
T_3	13	15	2
T_4	12	15	0

Table 18.3.1 *Time parameters for Example* 18.3.2.

To check time conflicts we have to investigate time compatibility of the preferences PR_i, $i = 1,\cdots,5$. Following (18.3.5), a first consistency investigation shows that each of the preferences, PR_3 and PR_5, is in conflict with the time constraints. The remaining preferences, PR_1, PR_2, and PR_4 would suggest execution of the tasks in order (T_1, T_2, T_3, T_4). To verify feasibility of this sequence we have to check all its subsequences against (18.3.6). The subsequences of length 2 are time compatible because the only time conflicting sequences would be (T_3, T_2) and (T_4, T_1). For the total sequence (T_1, T_2, T_3, T_4) we get $Z_3 = \max\{EB_3, EB_1 + p_1 + p_2, EB_2 + p_2\} = 15$ and $Z_3 + p_3 > LB_4$, thus the subset $\{PR_1, PR_2, PR_4\}$ of preferences creates a time conflict. Similarly the two subsequences of length 3 are tested: the result is that sequence (T_1, T_2, T_3) realized by preferences PR_1 and PR_2 establishes a time conflict, whereas (T_2, T_3, T_4) does not. So we end up with four time conflicting sets of preferences, $\mathcal{TC}_1 = \{PR_3\}$, $\mathcal{TC}_2 = \{PR_5\}$, $\mathcal{TC}_3 = \{PR_1, PR_2\}$, and $\mathcal{TC}_4 = \{PR_1, PR_2, PR_4\}$. □

If the implementation of some preference causes a resource demand at some time t such that it exceeds resource limits at this time, i.e.

$$\sum_{T_i \in \mathcal{T}_t} R_k(T_j) > m_k,\ k = 1,\cdots,s, \tag{18.3.7}$$

then a *resource conflict* occurs. Here $\mathcal{T}_t$ denotes the set of tasks being processed at time t, $R_k(T_j)$ the requirement of resource of type R_k of task T_j, and m_k the corresponding resource maximum supply.

Example 18.3.2 - continued - As to the *resource conflicts*, assume that $s = 1$, $m_1 = 1$, and $R_1(T_j) = 1$ for all $j = 1,\cdots,4$. Taking the given time constraints into account, we detect a conflict for PR_1 from (18.3.7) since T_2 cannot be processed in parallel with tasks T_3 and T_4. Thus an additional conflicting set $\mathcal{RC}_1 = \{PR_1\}$ has to be introduced. □

Coping with Conflicts

Let there be given a set $\mathcal{T}$ of tasks, and a set $\mathcal{PR}$ of preferences concerning the processing order of tasks. Assume that logical conflicting sets $\mathcal{LC}_1, \cdots, \mathcal{LC}_\lambda$, time conflicting sets $\mathcal{TC}_1, \cdots, \mathcal{TC}_\tau$, and resource conflicting sets $\mathcal{RC}_1, \cdots, \mathcal{RC}_\rho$ have been detected. We then want to find out if there is a solution schedule that meets all the restrictions coming from these conflicting sets. This means that we need to find a subset $\mathcal{PR}'$ of $\mathcal{PR}$ of maximal cardinality such that none of the conflicting sets $\mathcal{LC}_i$, $\mathcal{TC}_j$, $\mathcal{RC}_k$ contradicts $\mathcal{PR}'$, i.e. is contained in $\mathcal{PR}'$.

Let $\mathcal{LC} := \{\mathcal{LC}_1, \cdots, \mathcal{LC}_\lambda\}$ be the set of all logically conflicting sets; the set $\mathcal{TC}$ of time conflicting sets and the set $\mathcal{RC}$ of resource conflicting sets are defined analogously. Define $\mathcal{C} := \mathcal{LC} \cup \mathcal{TC} \cup \mathcal{RC}$, i.e. $\mathcal{C}$ contains all the conflicting sets of the system. The pair $\mathcal{IH} := (\mathcal{PR}, \mathcal{C})$ represents a *hypergraph* with *vertices* $\mathcal{PR}$ and *hyperedges* $\mathcal{C}$. Since $\mathcal{IH}$ describes all conflicts arising in the system we refer to $\mathcal{IH}$ as the *conflict hypergraph*.

Our aim is to find a suitable subset $\mathcal{PR}'$, i.e. one that does not contain any of the hyperedges. We notice that if $H_1 \subseteq H_2$ for hyperedges H_1 and H_2, we need not to consider H_2 since H_1 represents the more restrictive conflicting set. Observing this we can simplify the hypergraph by eliminating all hyperedges that are supersets of other hyperedges. The hypergraph then obtained is referred to as the *reduced conflict hypergraph*.

According to our *measure of acceptability* we are interested in the maximum number of preferences that can be accepted without loosing feasibility. This is justified if all preferences are of equal importance. If the preferences have different weights we might be interested in a subset of preferences of maximum total weight. All these practical questions result in NP-hard problems [GJ79].

To summarize the discussion we have to perform three steps to solve the problem.

Step 1: Detect all the logically, time, and resource conflicting sets.

Step 2: Build the reduced conflict hypergraph.

Step 3: Apply some conflict resolution algorithm.

Algorithm 18.3.3 *frame* ($\mathcal{IH} = (\mathcal{PR}, \mathcal{C})$);

```
begin
S := ∅;        -- initialization of the solution set
while PR ≠ ∅ do
  begin
  Reduce hypergraph (PR, C);
  Following some underlying heuristic, choose preference PR ∈ PR;   (18.3.8)
```

```
   PR := PR − {PR};
   if C ⊄ S ∪ {PR} for all C ∈ C then S := S ∪ {PR};
-- PR is accepted if the temporal solution set does not contain any conflicting preferences
   for all C ∈ C do C := C ∩ (PR × PR);
        -- the hypergraph is restricted to the new (i.e. smaller) set of vertices
   end;
end;
```

The algorithm is called *frame* because it has to be put in concrete form by introducing some specific heuristics in line (18.3.8). Based on the conflict hypergraph $\mathcal{H} = (\mathcal{PR}, \mathcal{C})$ heuristic strategies can easily be defined. We also mention that if preferences are of different importance their weight should be considered in the definition of the heuristics in (18.3.8).

In the following we give a simple example of how priority driven heuristics can be defined. Each time (18.3.8) is performed, the algorithm chooses a preference of highest priority. In order to gain better adaptability we allow that priorities are re-computed before the next choice is taken. This kind of dynamics is important in cases where the priority values are computed from the hypergraph structure, because as the hypergraph gets smaller step by step its structure changes during the execution of the algorithm, too.

Heuristic *DELTA-decreasing* (δ_{dec}): Let δ: $\mathcal{PR} \to \mathbb{N}^0$ be the *degree* that assigns - in analogy to the notion of degree in graphs - each vertex $PR \in \mathcal{PR}$ the number of incident hyperedges, i.e. the number of hyperedges containing vertex *PR*. The heuristic δ_{dec} then arranges the preferences in order of non-increasing degree. This strategy follows the observation that the larger the degree of a preference is, the more subsets of conflicting preferences exist; thus such a preference has less chance to occur in the solution set. To increase this chance we give such preference a higher priority.

Heuristic *DELTA-increasing* (δ_{inc}): Define $\delta_{inc} := -\delta_{dec}$. This way preferences of lower degree get higher priority. This heuristic was chosen for comparison against the δ_{dec} strategy.

Heuristic *GAMMA-increasing* (γ_{inc}): Define γ: $\mathcal{PR} \to \mathbb{N}^0$ as follows: For $PR \in \mathcal{PR}$, let $\gamma(PR)$ be the number of vertices that do not have any common hyperedge with *PR*. The heuristic γ_{inc} then arranges the preferences in order of non-decreasing cardinalities. The idea behind this strategy is that a preference with small γ-value has less chance to be selected to the solution set. To increase this chance we give such preference a higher priority.

Heuristic *GAMMA-decreasing* (γ_{dec}): Define $\gamma_{dec} := -\gamma_{inc}$. This heuristic was chosen for comparison against the γ_{inc} strategy.

The above heuristics have been compared by empirical evaluation [ES93]. There it turned out that *DELTA-decreasing*, *GAMMA-increasing* behave considerably better than *DELTA-increasing* and *GAMMA-decreasing*.

Example 18.3.2 - continued - Summarizing all conflicts we get the conflict hypergraph $\mathcal{H} := (\mathcal{PR}, \mathcal{C})$ where the set $\mathcal{C}$ contains the hyperedges

$\{PR_2, PR_3\}$
$\{PR_1, PR_2, PR_4, PR_5\}$ (logically conflicting sets)

$\{PR_3\}$
$\{PR_5\}$
$\{PR_1, PR_2\}$
$\{PR_1, PR_2, PR_4\}$ (time conflicting sets)

$\{PR_1\}$ (resource conflicting set).

Figure 18.3.3 shows the hypergraph where encircled vertices are hyperedges.

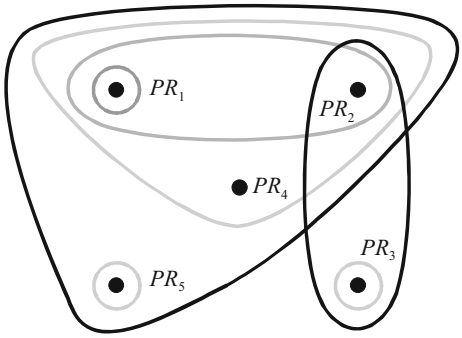


Figure 18.3.3 *$\mathcal{H} = (\mathcal{PR}, \mathcal{C})$ representing conflicts of the example problem.*

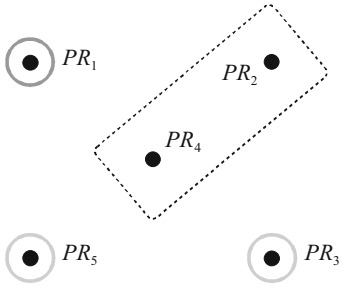


Figure 18.3.4 *Reduced hypergraph representing conflicts of the example problem.*

Since each of the hyperedges of cardinality > 1 contains a conflicting set of cardinality one, the reduced hypergraph has only the three hyperedges $\{PR_1\}$, $\{PR_3\}$ and $\{PR_5\}$, see Figure 18.3.4. A subset of maximal cardinality that is not in conflict with any of the conflicting sets is $\{PR_2, PR_4\}$. Each of the above algorithms finds this solution as can easily be verified. □

Below we present a more complex example where not only the heuristics can be nontrivially applied; the example also demonstrates the main idea behind an interactive schedule generation.

Working Features of an Interactive Scheduling System

The above described approach of *REST* with conflict detection mechanisms can be integrated into a DSS [EGS97]. Its general outline is shown in Figure 18.3.5.

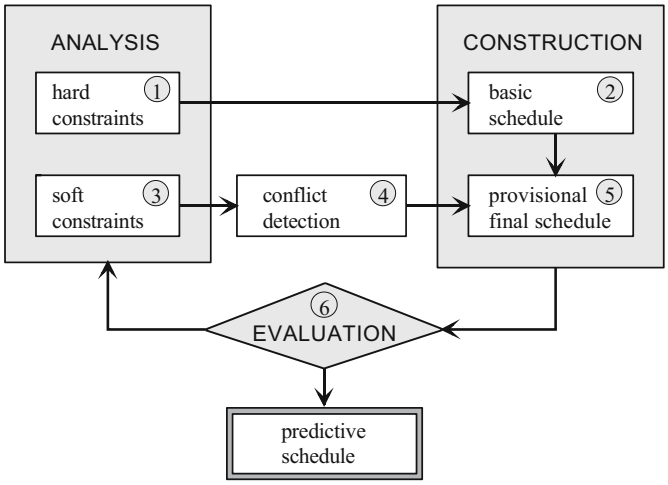


Figure 18.3.5 *A DSS for the REST-approach.*

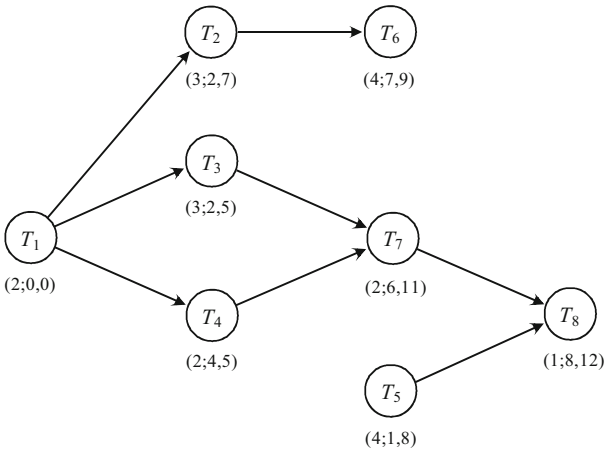
The DSS consists of four major modules: problem analysis, schedule generation, conflict detection and evaluation. Their working features can be organized by incorporating six phases. The first phase starts with some problem analysis investigating the hard constraints which have to be taken into account for any problem solution. Then, in the second phase a first feasible solution (basic schedule) is generated by applying some scheduling algorithm. The third phase takes over the part of analyzing the set of preferences of task constraints. In the fourth phase their interaction with the results of the basic schedule is clarified via the conflict detection module. In the fifth phase a compatible subset of soft constraints according to the objectives of the decision maker is determined, from which a revised schedule is generated. In the last phase the revised evaluated. If the evalua-

tion is satisfactory a solution for the predictive scheduling problem is found; if not, the schedule has to be revised by considering new constraints from the decision maker. The loop stops as soon as a satisfactory solution has been found.

The DSS can be extended to handle a dynamic environment. Whenever hard constraints have to be revised or the set of preferences is changing we can apply this approach on a rolling basis.

Example 18.3.4 To demonstrate the working feature of the scheduling system consider an extended example. Let there be given a set of tasks $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$, and hard constraints as shown in Figure 18.3.6(a). Processing times and earliest and latest beginning times are given as triples (p_j, EB_j, LB_j) next to the task nodes. In addition, concurrent task execution is restricted by two types of resources and resource requirements of the tasks are $\boldsymbol{R}(T_1) = [2, 0]$, $\boldsymbol{R}(T_2) = [2, 4]$, $\boldsymbol{R}(T_3) = [0, 1]$, $\boldsymbol{R}(T_4) = [4, 2]$, $\boldsymbol{R}(T_5) = [1, 0]$, $\boldsymbol{R}(T_6) = [2, 5]$, $\boldsymbol{R}(T_7) = [3, 0]$, $\boldsymbol{R}(T_8) = [0, 1]$. The total resource supply is $\boldsymbol{m} = [5, 5]$.

(a)



(b)

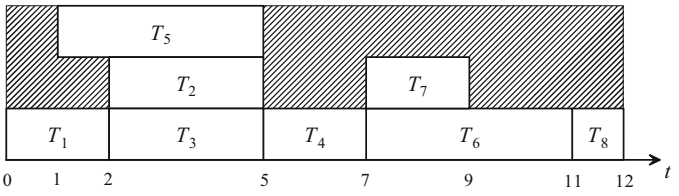


Figure 18.3.6 *Illustration of Example* 18.3.4 :
(a) *hard constraints,*
(b) *a basic schedule.*

Having analyzed the hard constraints we generate a feasible basic schedule by applying some scheduling algorithm. The result is shown in Figure 18.3.6(b).

Feasibility of the schedule is gained by assigning a starting time s_j to each task such that $EB_j \le s_j \le LB_j$ and the resource constraints are met.

Describing the problem in terms of the constraint satisfaction problem, the variables refer to the starting times of the tasks, their domains to the intervals of corresponding earliest and latest beginning times and the constraints to the set of preferences. Let the set of preferences be given by $\mathcal{PR} = \{PR_1, \cdots, PR_7\}$ with $PR_1 = (T_3, T_4)$, $PR_2 = (T_2, T_3)$, $PR_3 = (T_4, T_3)$, $PR_4 = (T_7, T_5)$, $PR_5 = (T_5, T_2)$, $PR_6 = (T_5, T_6)$, and $PR_7 = (T_4, T_5)$ (see Figure 18.3.7). Notice that the basic schedule of Figure 18.3.6(b) realizes just two of the preferences.

Analyzing conflicts we start with the detection of logical conflicts. From the cycles of the graph in Figure 18.3.7 we get the logically conflicting sets $\mathcal{LC}_1 = \{PR_1, PR_3\}$ and $\mathcal{LC}_2 = \{PR_1, PR_2, PR_5, PR_7\}$.

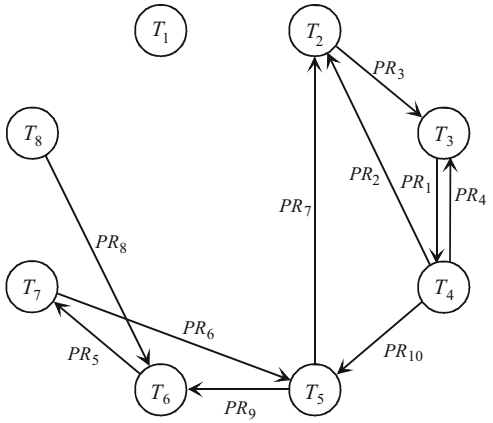


Figure 18.3.7 *$G = (\mathcal{T}, \mathcal{PR})$ representing preferences in Example* 18.3.4.

Task sequence	Time conflicting set of preferences
(T_4, T_3)	$\mathcal{TC}_1 = \{PR_3\}$
(T_2, T_3, T_4)	$\mathcal{TC}_2 = \{PR_1, PR_2\}$
(T_7, T_5, T_6)	$\mathcal{TC}_3 = \{PR_4, PR_6\}$
(T_2, T_3, T_4, T_5)	$\mathcal{TC}_4 = \{PR_1, PR_2, PR_7\}$
(T_3, T_4, T_5, T_2)	$\mathcal{TC}_5 = \{PR_1, PR_5, PR_7\}$
(T_4, T_5, T_2, T_3)	$\mathcal{TC}_6 = \{PR_2, PR_5, PR_7\}$
(T_5, T_2, T_3, T_4)	$\mathcal{TC}_7 = \{PR_1, PR_2, PR_5\}$

Table 18.3.2 *Subsets of preferences being in time conflict.*

For the analysis of time constraints we start with task sequences of length 2. We see that there is only one such conflict, $\mathcal{TC}_1$. Next, task sequences of length greater than 2 are checked. Table 18.3.2 summarizes non-feasible task sequences and their corresponding time conflicting subsets of preferences.

So far we found 2 logically conflicting sets and 7 time conflicting sets of preferences. In order to get the reduced hypergraph, all sets that already contain a conflicting set must be eliminated. Hence there remain 5 conflicting sets of preferences, $\{PR_3\}$ $\{PR_1, PR_2\}$, $\{PR_4, PR_6\}$, $\{PR_1, PR_5, PR_7\}$ and $\{PR_2, PR_5, PR_7\}$. The corresponing hypergraph is sketched in Figure 18.3.8.

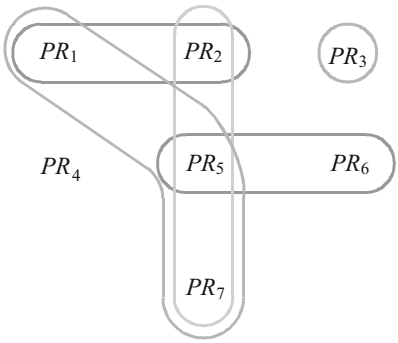


Figure 18.3.8 $G_c = (\mathcal{PR}, \mathcal{E})$ *representing logical and time conflicts of Example* 18.3.4.

We did, however, not consider the resource constraints so far. To detect resource conflicts we had to find all combinations of tasks which cannot be scheduled simultaneously because of resource conflicts. Since in general the number of these sets increases exponentially with the number of tasks, we follow another strategy: First create a schedule without considering resource conflicts, then check for resource conflicts and introduce additional precedence constraints between tasks being in resource conflict. In this manner we proceed until a feasible solution is found.

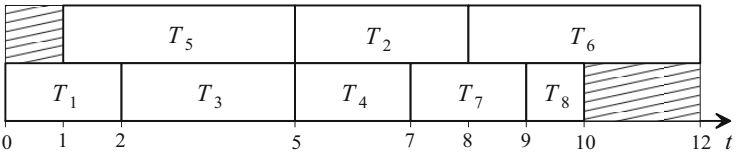


Figure 18.3.9 *Schedule for Example* 18.3.4 *without considering resource conflicts.*

To construct a first schedule, we aim to find a set of non-conflicting preferences of maximum cardinality, i.e. a maximum set that does not contain any of the hy-

peredges of the above hypergraph. For complexity reasons we content ourselves with an approximate solution and apply algorithm *frame*. Heuristics GAMMA-increasing, for example, selects the subset $\{PR_1, PR_5, PR_6\}$ and we result in the schedule presented in Figure 18.3.9. Remember that we assumed for simplicity reasons that all preferences are equally weighted.

The schedule of Figure 18.3.9 shows two resource conflicts, for T_2, T_4 and for T_6, T_8. Hence T_2 and T_4 (and analogously T_6 and T_8) cannot be processed simultaneously, and we have to choose an order for these tasks. This way we end up with two additional hard constraints in the precedence graph shown in figure 18.3.10(a). Continuing the analysis of constraints we result in a schedule that realizes the preferences PR_5 and PR_6 (Figure 18.3.10(b)). □

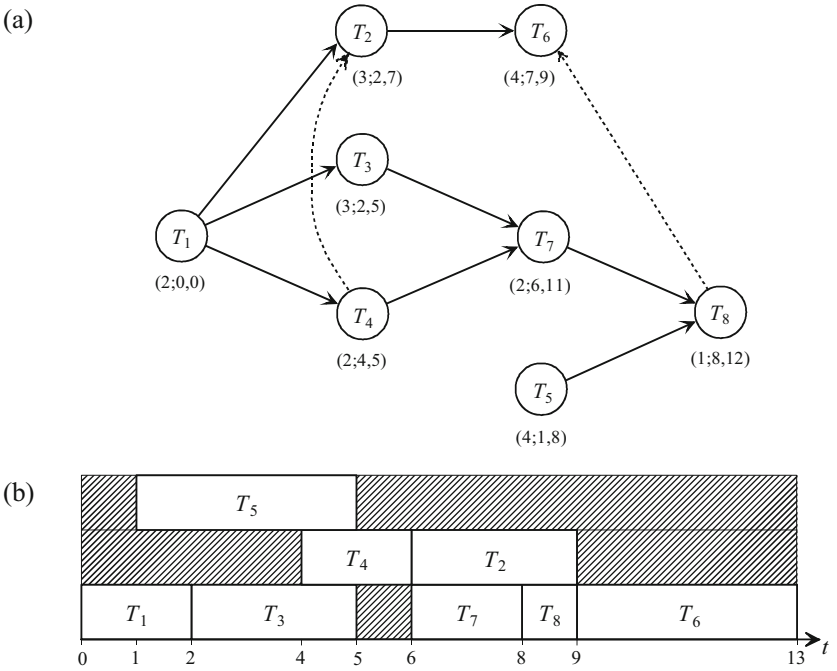


Figure 18.3.10 *Final schedule for Example* 18.3.4 :
(a) *precedence graph of Figure* 18.3.6(a) *with additional hard constraints* (T_4, T_2) and (T_8, T_6),
(b) *a corresponding schedule.*

We can now summarize our approach by the following algorithm:

Algorithm 18.3.5 *for interactive scheduling.*
`begin`
Initialize 'Basic Schedule'**;**
Collect preferences**;**

Detect conflicts;
while conflicts exist **do** Apply *frame*;
Generate final schedule;
end;

Reactive Scheduling

Now assume that the predictive schedule has been implemented and some unforeseen disturbance occurs. In this case within the ACE loop (compare Figure 18.3.1) reactive scheduling is concerned with revising predictive schedules as unexpected events force changes. Now we want to present some ideas how to interactively model parts of the reactive scheduling process using fuzzy logic. The idea is to apply this approach for monitoring and diagnosing purposes only. Based on the corresponding observations detailed reactive scheduling actions can be taken by the decision maker [Sch94].

An algorithmic reaction on the reactive level of problem solving based on sophisticated combinatorial considerations is generally not possible because of prohibitive computing times; therefore, the competence of human problem solvers in reaching quality, real-time decisions is extremely important. The human problem solver should be supported by advanced modeling techniques. In order to achieve this we suggest the application of fuzzy logic because it allows to represent the vague, qualitative view of the human scheduler most conveniently. The underlying theory of fuzzy sets [Zad65] concentrates on modeling vagueness due to common sense interpretation, data aggregation and vague relations. Examples for common sense interpretation in terms of a human production scheduler are e.g. 'long queues of jobs' or 'high machine processing speed'. Data aggregation is used in expressions like 'skilled worker' or 'difficult situation' and vague relations are represented by terms like 'not much more urgent than' or 'rather equal'.

Reactive scheduling requires suitable diagnostic support for quick decision making. This is intended with our approach modeling reactive scheduling by fuzzy logic. The two main components of the model we use are (1) linguistic variables [Zad73] and (2) fuzzy rules or better decision tables [Kan86]. A linguistic variable L can be represented by the tuple $L = (X, U, f)$ where set X represents the feasible values of L, set U represents the domain of L, and f is the membership function of L which assigns to each element $x \in X$ a fuzzy set $A(x)=\{u, f_x(u)\}$ where $f_x(u) \in [0, 1]$.

A decision table (DT) consists of a set of conditions (if-part) and a set of actions (then-part). In case of multi conditions or multi actions conditions or actions respectively have to be connected by operators. If all conditions have only one precise value we speak of *deterministic* DT. In case we use fuzzy variables for representing conditions or actions we also can build non-deterministic DT. These tables are very much alike of how humans think. In order to represent the interaction of linguistic variables in DT we have to introduce set-theoretic opera-

tions to find the resulting membership function. The most common operations are union, intersection and complement. In case of an union we have $f_C(u) = \max\{f_A(u), f_B(u)\}$, in case of an intersection $f_D(u) = \min\{f_A(u), f_B(u)\}$, and in case of the complement A° of A we have $f_{A^\circ}(u) = 1 - f_A(u)$. To understand the approach of modeling reactive scheduling by fuzzy logic better consider the following scenario.

There are queues of jobs in front of machines on the shop floor. For each job J_j the number of jobs N_j waiting ahead in the queue, its due date d_j and its slack time $s_j = d_j - t$ are known where t is the current time. Processing times of the jobs are subject to disturbances. Due date and machine assignment of the jobs are determined by predictive scheduling. The objective is to diagnose critical jobs, i.e. jobs which are about to miss their due dates in order to reschedule them. N_j and s_j are the linguistic input variables and "becomes critical" is the output variable of the DT. Membership functions for the individual values of the variables are determined by a knowledge acquisition procedure which will not be described here. The following DT shown in Table 18.3.3 represents the fuzzy rule system.

AND	Small	Medium	Great
Few	soon	later	not to see
Some	now	later	not to see
Many	now	soon	not to see
Very	now	soon	later

Table 18.3.3 *Decision table for fuzzy rule system.*

The rows represent the values of the variable N_j and the columns represent the values of the variable s_j. Both variables are connected by an AND-operator in any rule. With the above Table 18.3.3 twelve rules are represented. Each element of the table gives one value of the output variable "becomes critical" depending on the rule. To find these results the membership functions of the input variables are merged by intersection operations to a new membership function describing the output variable of each rule. The resulting fuzzy sets of all rules are then combined by operations which are applied for the union of sets. This procedure was tested to be most favorable from an empirical point of view.

As a result the decision maker on the shop floor gets the information which jobs have to be rescheduled now, soon, later, or probably not at all. From this two possibilities arise; either a complete new predictive schedule has to be generated or local ad-hoc decisions can be taken on the reactive scheduling level. Control decisions based on this fuzzy modeling approach and their consequences should be recorded and evaluated, for the purpose of using these past decisions to find better solutions to current problems. Fuzzy case-based reasoning systems,

which should be able to achieve the quality of a self-learning system, could make a significant contribution along these lines.

We have implemented our approach of modeling reactive scheduling by fuzzy logic as a demonstration prototype. Two screens serve as the user interface. On the first screen the jobs waiting in a machine queue and the slack time of the job under investigation are shown. An example problem is shown in Figure 18.3.11.

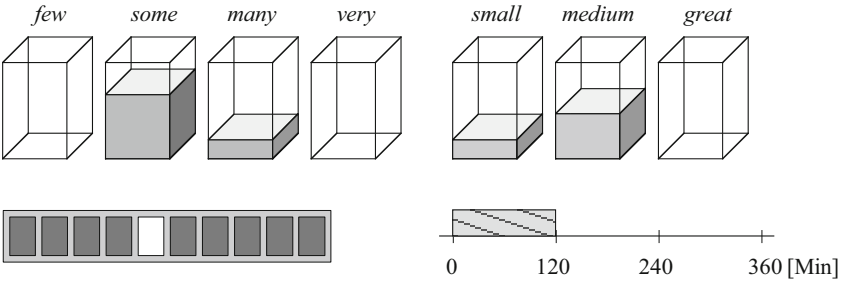


Figure 18.3.11 *Interface of fuzzy scheduler.*

There are ten jobs waiting in a queue to be processed by some machine P_i, the job under consideration J_j is shown by a white rectangle. Above the queue the different fuzzy sets concerning the linguistic variable N_j and the values of the corresponding membership functions are represented. The slack time s_j of job J_j is currently 130 minutes; again fuzzy sets and membership functions of this linguistic variable are represented above the scale.

Applying the rules of the DT results in a representation which is shown in Figure 18.3.12. The result of the first part of inference shows that for job J_j the output variable "becomes critical" is related to some positive values for "soon" and for "later" shown by white segments. From this it is concluded by the second part of inference that this job has to be checked again before it is about to be rescheduled.

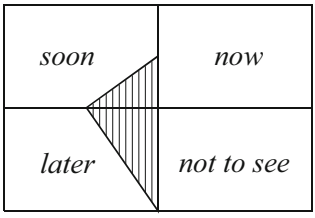


Figure 18.3.12 *Result of inference.*

18.3.2 Knowledge-based Scheduling

Expert Systems are special kinds of knowledge-based systems. Expert systems are designed to support problem modeling and solving with the intention to simulate the capabilities of domain experts, e.g. problem understanding, problem solving, explaining the solution, knowledge acquisition, and knowledge restructuring. Most expert systems use two types of knowledge: *descriptive knowledge* or *facts*, and *procedural knowledge* or knowledge about the *semantics* behind facts. The architecture of expert systems is mainly based on a *closed loop solution approach*. This consists of the four components *storing knowledge*, *knowledge acquisition*, *explanation of results*, and *problem solution*. In the following we will concentrate on such a closed loop problem solution processes.
There is an important difference between expert systems and conventional problem solving systems. In most expert systems the model of the problem description and basic elements of problem solving are stored in a knowledge base. The complete solution process is carried out by some inference module interacting with the knowledge base. Conventional systems do not have this kind of separated structure; they are rather a mixture of both parts in one program.

In order to implement an expert system one needs three types of *models*: a model of the domain, a model of the elementary steps to be taken, and a model of inference that defines the sequence of elementary steps in the process of problem solution. The domain is represented using descriptive knowledge about objects, their attributes and their relations as introduced in Section 18.2. In production scheduling for example, objects are machines, jobs, tasks or tools, attributes are machine states, job and task characteristics or tool setup times, and relations could be the subsumption of machines to machine types or tasks to jobs. The model of elementary steps uses production rules or other representations of procedural knowledge. For if-then rules there exists a unique input-output description. The model of inference uses combinations or sets of elementary steps to represent the solution process where a given start state is transformed to a desired goal state. This latter type of knowledge can also be knowledge of domain experts or domain independent knowledge. The goal of the expert system approach is mainly to improve the modeling part of the solution process to get closer to reality.

To give a better understanding of this view we refer to an example given by Kanet and Adelsberger [KA87]: "... consider a simple scheduling situation in which there is a single machine to process jobs that arrive at different points in time within the planning period. The objective might be to find a schedule which minimizes mean tardiness. An algorithmic approach might entertain simplifying the formulation by first assuming all jobs to be immediately available for processing. This simplified problem would then be solved and perhaps some heuristic used to alter the solution so that the original assumption of dynamic arrivals is back in tack. The approach looks at reformulation as a means to 'divide et impera'. On the other hand a reformulative approach may ... seek to find a 'richer'

problem formulation. For example the question might be asked 'is working overtime a viable alternative?', or 'does there exist another machine that can accomplish this task?', or 'is there a subset of orders that are less critical than others?', and so on."

On the other hand systems for production scheduling should not only replicate the expert's schedule but extend the capabilities by doing more problem solving. In order to achieve this AI systems separate the scheduling model from a general solution procedure. In [Fox90] the shop floor scheduling model described uses terms from AI. It is considered to be time based planning where tasks or jobs must be selected, sequenced, and assigned to resources and time intervals for execution. Another view is that of a multi agent planning problem, where each task or job represents a separate agent for which a schedule is to be created; the agents are uncooperative, i.e. each is attempting to maximize its own goals. It is also claimed that expert systems appear inappropriate for the purpose of problem solution especially for two reasons: (1) problems like production scheduling tend to be so complex that they are beyond the cognitive capabilities of the human scheduler, and (2) even if the problem is relatively easy, factory environments change often enough so that any expertise built up over time becomes obsolete very quickly.

We believe that it is nevertheless possible to apply an expert system approach for the solution of production scheduling problems but with a different perspective on problem solving. Though, as already stated, expert systems are not appropriate for solving combinatorial search problems, they are quite reasonable for the *analysis* of models and their solutions. In this way expert systems can be used for building or selecting models for scheduling problems. An appropriate solution procedure can be selected for the model, and then the expert system can again support the evaluation of the solution.

The scheduling systems reviewed next are not expert systems in their purest sense and thus we will use the more general term *knowledge-based system*. ISIS [SFO86, Fox87, FS84], OPIS [SPP+90] and CORTES [FS90] are a family of systems with the goal of modeling knowledge of the manufacturing environment using mainly constraints to support *constraint guided search*; knowledge about constraints is used in the attempt to decrease the underlying search space. The systems are designed for both, predictive and reactive scheduling.

ISIS-1 uses pure constraint guided search, but was not very successful in solving practical scheduling problems. ISIS-2 uses a more sophisticated search technique. Search is divided into the four phases job selection, time analysis, resource analysis, and resource assignment. Each phase consists in turn of the three sub-phases pre-search analysis (model construction), search (construction of the solution), and post-search analysis (evaluation of the solution). In the job selection phase a priority rule is applied to select the next job from the given set of available jobs. This job is passed to the second phase. Here earliest start and latest finish times for each task of the job are calculated without taking the resource requirements into account. In phases three and four the assignment of re-

sources and the calculation of the final start and finish times of all tasks of the job under consideration is carried out. The search is organized by some *beam search* method. Each solution is evaluated within a rule-based post-search analysis. ISIS-3 tries to schedule each job using more information from the shop floor, especially about bottleneck-resources. With this information the job-centered scheduling approach as it is realized in ISIS-2 was complemented by a resource-centered scheduler.

As the architecture of ISIS is inflexible as far as modifications of given schedules are concerned, a new scheduling system called OPIS-1 was developed. It uses a *blackboard approach* for the communication of the two knowledge sources analysis and decision. These use the blackboard as shared memory to post messages, partial results and any further information needed for the problem solution. The blackboard is the exclusive medium of communication. Within OPIS-1 the "analyzer" constructs a rough schedule using some *balancing heuristic* and then determines the bottlenecks. Decision is then taken by the resource and the job scheduler already implemented in ISIS-3. Search is centrally controlled. OPIS-1 is also capable to deal with reactive scheduling problems, because all events can be communicated through the blackboard. In OPIS-2 this event management is supported by two additional knowledge sources which are a "right shifter" and a "demand swapper". The first one is responsible for pushing jobs forward in the schedule, and the second for exchanging jobs. Within the OPIS systems it seems that the most difficult operation is to decide which knowledge source has to be activated.

The third system of the family we want to introduce briefly is CORTES. Whereas the ISIS systems are primarily job-based and OPIS switches between job-based and resource-based considerations, CORTES takes a task-oriented point of view, which provides more flexibility at the cost of greater search effort. Within a five step heuristic procedure a task is assigned to some resource over some time interval.

Knowledge-based systems using an expert system approach should concentrate on finding good models for the problem domain and the description of elementary steps to be taken during the solution process. The solution process itself may be implemented by a different approach. One example for model development considering knowledge about the domain and elementary steps to be taken can be found in [SS90]. Here a reactive scheduling problem is solved along the same line as OPIS works using the following problem categorization: (1) machine breakdown, (2) rush jobs, (3) new batch of jobs, (4) material shortage, (5) labor absenteeism, (6) job completion at a machine, and (7) change in shift. Knowledge is modularized into independent knowledge sources, each of them designed to solve a specific problem. If a new event occurs it is passed to some meta-analyzer and then to the appropriate knowledge source to give a solution to the analyzed scheduling problem. For instance, the shortage of some specific raw material may result in the requirement of rearranging the jobs assigned to a par-

ticular machine. This could be achieved by using the human scheduler′s heuristic or by an appropriate algorithm to determine some action to be taken.

As a representative for many other knowledge-based scheduling systems - see [Ata91] for a survey - we want to describe *SONIA* which integrates both predictive and reactive scheduling on the basis of hard and soft constraints [CPP88]. The scheduling system is designed to detect and react to inconsistencies (conflicts) between a predictive schedule and the actual events on the shop floor. SONIA consists of two analyzing components, a capacity analyzer and an analyzer of conflicts, and further more a predictive and a reactive component, each containing a set of heuristics, and a component for managing schedule descriptions.

For representing a schedule in SONIA the resources needed for processing jobs are described at various levels of detail. Individual resources like machines are elements of resource groups called work areas. Resource reservation constraints are associated with resources. To give an example for such a constraint, (*res*; t_1, t_2, *n*; *list-of-motives*) means that *n* resources from resource group *res* are not available during the time interval (t_1, t_2) for the reasons given in the *list-of-motives*.

Each job is characterized by a ready time, a due date, precedence constraints, and by a set of tasks, each having resource requirements. To describe the progress of work the notions of an actual status and a schedule status are introduced. The *actual status* is of either kind "completed", "in-process", "not started", and the *schedule status* can be "scheduled", "selected" or (deliberately) "ignored". There may also be temporal constraints for tasks. For example, such a constraint can be described by the expression ($time \leq t_1 t_2$, k) where t_1 and t_2 are points in time which respectively correspond to the start and the finish time of processing a task, and k represents the number of time units; if there have to be at least t time units between processing of tasks T_j and T_{j+1}, the corresponding expression would be ($time \leq (end\ T_j)(start\ T_{j+1})$, t). To represent actual time values, the origin of time and the current time have to be known.

SONIA uses constraint propagation which enables the detection of inconsistencies or conflicts between predictive decisions and events happening on the shop floor. Let us assume that as a result of the predictive schedule it is known that task T_j could precede task T_{j+1} while the actual situation in the workshop is such that T_j is in schedule status "ignored" and T_{j+1} is in actual status "in process". From this we get an inconsistency between these temporal constraints describing the predictive schedule and the ones which come from the actual situation. The detection of conflicts through constraint propagation is carried out using propagation axioms which indicate how constraints and logic expressions can be combined and new constraints or conflicts can be derived. The axioms are utilized by an interpreter.

SONIA distinguishes between the three major kinds of conflicts: delays, capacity conflicts and breakdowns. The class of delays contains all conflicts which

result from unexpected delays. There are four subclasses to be considered, "Task Delay" if the expected finish time of a task cannot be respected, "Due-Date Delay" if the due date of a manufacturing job cannot be met, "Interruption Delay" if some task cannot be performed in a work shift determined by the predictive schedule, and "Global Tardiness Conflict" if it is not possible to process all of the selected tasks by the end of the current shift. The class of capacity conflicts refers to all conflicts that come from reservation constraints. There are three subclasses to be considered. If reservations for tasks have to be cancelled because of breakdowns we speak of "Breakdown Capacity Conflicts". In case a resource is assigned to a task during a work shift where this resource is not available, an "Out-Of-Shift Conflict" occurs. A capacity conflict is an "Overload" if the number of tasks assigned to a resource during a given interval of time is greater than the available capacity. The third class consists of breakdowns which contains all subclasses from delays and capacity conflicts caused only by machine breakdowns. In the following we give a short overview of the main *components* of the SONIA system and its *control architecture*.

(*i*) *Predictive Components* The predictive components are responsible for generating an off-line schedule and consist of a selection and an ordering component. First a set of tasks is selected and resources are assigned to them. The selection depends on other already selected tasks, shop status, open work shifts and jobs to be completed. Whenever a task is selected its schedule status is "selected" and the resulting constraints are created by the schedule management system. The ordering component then uses an iterative constraint satisfaction process utilizing heuristic rules. If conflicts arise during schedule generation, backtracking is carried out, i.e. actions coming from certain rules are withdrawn. If no feasible schedule can be found for all the selected tasks a choice is made for the tasks that have to be rejected. Their schedule status is set to "ignored" and the corresponding constraints are deleted.

(*ii*) *Reactive Components*. For reactive scheduling three approaches to resolve conflicts between the predictive schedule and the current situation on the shop floor are possible: Predictive components can generate a complete new schedule, the current schedule is modified globally forward from the current date, or local changes are made. The first approach is the case of predictive scheduling which already been described above. The easiest reaction to modify the current schedule is to reject tasks, setting their scheduling status to "ignored" and deleting all related constraints. Of course, the rejected task should be that one causing the conflicts. If several rejections are possible the problem gets far more difficult and applicable strategies have still to be developed. Re-scheduling forward from the current date is the third possibility of reaction considered here. In this case very often due dates or ends of work shifts have to be modified. An easy reaction would simply by a right shift of all tasks without modifying their ordering and the resource assignments. In a more sophisticated approach some heuristics are applied to change the order of tasks.

(*iii*) *Analysis Components*. The purpose of the analyzers is to determine which of the available predictive and reactive components should be applied for schedule generation and how they should be used. Currently, there are two analysis components implemented, a capacity analyzer and a conflict analyzer. The capacity analyzer has to detect bottleneck and under-loaded resources. These detections lead to the application of scheduling heuristics, e.g. of the kind that the most critical resources have to be scheduled first; in the same sense, under-loaded resources lead to the selection of additional tasks which can exploit the resources. The conflict analyzer chooses those available reactive components which are most efficient in terms of conflict resolution.

(*iv*) *Control Architecture*. Problem solving and evaluating knowledge have to be integrated and adjusted to the problem solving context. A blackboard architecture is used for these purposes. Each component can be considered as an independent knowledge source which offers its services as soon as predetermined conditions are satisfied. The blackboard architecture makes it possible to have a flexible system when new strategies and new components have to be added and integrated. The domain blackboard contains capacity of the resources determined by the capacity analyzer, conflicts which are updated by the schedule management, and results given by predictive and reactive components. The control blackboard contains the scheduling problem, the sub-problems to be solved, strategies like heuristic rules or meta-rules, an agenda where all the pending actions are listed, policies to choose the next pending action and a survey of actions which are currently processed.

SONIA is a knowledge-based scheduling system which relies on constraint satisfaction where the constraints come from the problem description and are then further propagated. It has a very flexible architecture, generates predictive and reactive schedules and integrates both solution approaches. A deficiency is that nothing can be said from an ex-ante point of view about the quality of the solutions generated by the conflict resolution techniques. Unfortunately also a judgement from an ex-post point of view is not possible because there is no empirical data available up to now which gives reference to some quality measure of the schedule. Also nothing is known about computing times. As far as we know, this lack of evaluation holds for many knowledge-based scheduling systems developed until today.

18.3.3 Integrated Problem Solving

In this last section we first want to give an example to demonstrate the approach of integrating algorithms and knowledge within an interactive approach for OFP and ONC relying on the ACE loop. For clarity purposes, the example is very simple. Let us assume, we have to operate a flexible manufacturing cell that consists of identically tooled machines all processing with the same speed. These

kinds of cells are also called pools of machines. From the production planning system we know the set of jobs that have to be processed during the next period of time e.g. in the next shift. As we have identical machines we will now speak of tasks instead of jobs which have to be processed. The business need is that all tasks have to be finished at the end of the next eight hour shift. With this the problem is roughly stated.

Using further expert knowledge from scheduling theory for the analysis of the problem we get some insights using the following knowledge sources (see Chapter 5 for details):

(1) The schedule length is influenced mainly by the sequence the tasks enter the system, by the decision to which machine an entering task is assigned next, and by the position an assigned task is then given in the corresponding machine queue.

(2) As all machines are identically tooled each task can be processed by all machines and with this also preemption of tasks between machines might be possible.

(3) The business need of processing all tasks within the next eight hour shift can be translated in some objective which says that we want to minimize schedule length or makespan.

(4) It is well known that for identical machines, independent tasks and the objective of minimizing makespan, schedules with preemptions of tasks exist which are never worse than schedules where task preemption is not allowed.

From the above knowledge sources (1)-(4) we conclude within the problem analysis phase to choose McNaughton's rule [McN59] to construct a first basic schedule. From an evaluation of the generated schedule it turns out that all tasks could be processed within the next shift. Another observation is that there is still enough idle time to process additional tasks in the same shift. To evaluate the dynamics of the above manufacturing environment we simulate the schedule taking also transportation times of the preempted tasks to the different machines into account. From the results of simulation runs we now get a better understanding of the problem. It turns out that considering transportation of tasks the schedule constructed by McNaughton's rule is not feasible, i.e. in conflict according to the restriction to finish all tasks within the coming shift. The transport times which were neglected during static schedule generation have a major impact on the schedule length.

From this we must analyze the problem again and with the results from the evaluation process we derive the fact that the implemented schedule should not have machine change-overs of any task, to avoid transport times between machines.

Based on this new constraint and further knowledge from scheduling theory we decide now to use the longest processing time heuristic to schedule all tasks. It is shown in Chapter 5 that LPT gives good performance guarantees concerning schedule length and problem settings with identical machines. Transport times

between machines do not have to be considered any more as each task is only assigned to one machine. Let us assume the evaluation of the LPT-schedule is satisfactory.

Now, we use earliest start and latest finish times for each task as constraints for ONC. These time intervals can be determined using the generated OFP-schedule. Moreover we translate the LPT rule into a more operational scheduling rule which says: release all the tasks in a non-increasing order of processing times to the flexible manufacturing cell and always assign a task to the queue of a machine which has least actual total work to process. The machine itself selects tasks from its own queue according to a first-come-first-served (FCFS) strategy.

As long as the flexible manufacturing cell has no disturbances ONC can stick to the given translation of the LPT-strategy. Now, assume a machine breaks down and that the tasks waiting in the queue have to be assigned to queues of the remaining machines. Let us further assume that under the new constraints not all the tasks can be finished in the current shift. From this a new objective occurs for reactive scheduling which says that as many tasks as possible should be finished. Now, FCFS would not be the appropriate scheduling strategy any longer; a suitable ad-hoc decision for local repair of the schedule has to be made. Finding this decision on the ONC-level means again to apply some problem analysis also in the sense of diagnosis and therapy, i.e. also ad-hoc decisions follow some analysis-construction sequence. If there is enough time available also some simulation runs could be applied, but in general this is not possible. To show a way how the problem can be resolved similar rules as these from Table 18.3.4 could be used.

For the changed situation, the shortest processing time (SPT) rule would now be applied. The SPT rule is proposed due to the expectation that this rule helps to finish as many tasks as possible within the current shift. In case of further disturbances that cause major deviations from the current system status, OFP has to be reactivated for a global repair of the schedule.

At the end of this section we want to discuss shortly the relationship of our approach to solve production scheduling problems and the requirements of integrated problem solving within computer integrated manufacturing. The IPS has to be connected to existing information systems of an enterprise. It has interfaces to the production planning systems on a tactical level of decision making and the real-time oriented CAM-systems. It represents this part of the production scheduling system which carries out the feedback loop between planning and execution. The vertical decision flow is supplemented by a horizontal information flow from CAE and CAQ. The position of the IPS within CIM is shown in Figure 18.3.13.

We gave a short introduction to an IPS which uses an interactive scheduling approach based on the ACE loop. Analysis and evaluation are carried out mainly by the decision maker, construction is mainly supported by the system. To that end a number of models and methods for analysis and construction have been devised, from which an appropriate selection should be possible. The modular

and open architecture of the system offers the possibility of a step by step implementation which can be continuously adapted to changing requirements.

```
/* Goal rule

Rule 0100

IF      Machine.Sequence = known
THEN    Machine.Schedule = completed
END


/* Determine the scheduling strategy

/* SPT-rule to reduce system overload

Rule 1000

IF      Machine.Status = overloaded
AND     Queue.Orders = not_late
AND     System.Status = overloaded
THEN    Machine.Sequence =
        proc(SPT_Processing, Machine.Duration)

/* FCFS-default strategy

Rule 1500 SELFREF

IF      Machine.Sequence = notknown
THEN    Machine.Sequence =
        proc(FCFS_Processing, Machine.Arrival)
END
```

```
/* Determine the status of the machine

Rule 2000

IF      Machine.Backlog > 40
THEN    Machine.Status = overloaded
END


/* Determine the status of the queue

Rule 3000

IF      Queue.Minbuffer > 20
THEN    Queue.Jobs = not_late
END


/* Determine the status of the system

Rule 4000

IF      System.Jobs > 30
AND     Machine.Number_overloaded > 4
THEN    System.Status = overloaded
END
```

Table 18.3.4 *Example problem for reactive scheduling.*

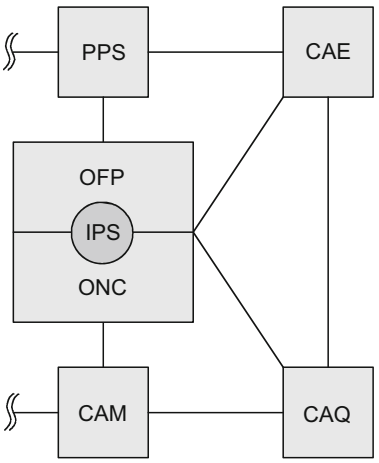


Figure 18.3.13 *IPS within CIM.*

A further application of the system lies in a distributed production scheduling environment. The considered manufacturing system has to be modeled and appropriately decomposed into subsystems. For the manufacturing system and each of its subsystems corresponding IPS apply, which are implemented on different computers connected by an appropriate communication network. The IPS on the top level of the production system serves as a coordinator of the subsystem IPS. Each IPS on the subsystem level works independently fulfilling the requirements from the master level and communicating also with the other IPS on this level. Only if major decisions which requires central coordination the master IPS is also involved.

References

Ata91 H. Atabakhsh, A survey for constraint based scheduling systems using an artificial intelligence approach, *Artif. Intell. Eng.* 6, 1991, 58-73.

BPH82 J. H. Blackstone, D. T. Phillips, G. L. Hogg, A state-of-the-art survey of dispatching rules for manufacturing job shop operations, *Int. J. Prod. Res.* 20, 1982, 27-45.

Bul82 W. Bulgren, *Discrete System Simulation*, Prentice-Hall, Englewood Cliffs, N.J., 1982.

BY86 J. A. Buzacott, D. D. Yao, FMS: a review of analytical models, *Manage. Sci.* 32, 1986, 890-905.

Car86 A. S. Carrie, The role of simulation in FMS, in: A. Kusiak (ed.), *Flexible Manufacturing Systems: Methods and Studies*, Elsevier, 1986, 191-208.

CPP88 A. Collinot, C. Le Pape, G. Pinoteau, SONIA: a knowledge-based scheduling system, *Artif. Intell. Eng.* 3, 1988, 86-94.

CY91 P. Coad, E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

DP88 R. Dechter, J. Pearl, Network-based heuristics for constraint-satisfaction problems, *Artif. Intell.* 34, 1988, 1-38.

DTLZ93 J. Drake, W. T. Tsai, H. J. Lee, Object-oriented analysis: criteria and case study, *Int. J. Softw. Eng. Knowl. Eng.* 3, 1993, 319-350.

EGS97 K. Ecker, J. N. D. Gupta, G. Schmidt, A framework for decision support systems for scheduling problems, *Eur. J. Oper. Res.* 101, 1997, 452-462.

ES93 K. Ecker, G. Schmidt, Conflict resolution algorithms for scheduling problems, in: K. Ecker, R. Hirschberg (eds.), *Workshop on Parallel Processing*, Clausthal University of Technology, 1993, 81-90.

Fox87 M. S. Fox, *Constraint Directed Search: A Case Study of Job-Shop Scheduling*, Morgan Kaufmann, 1987.

Fox90 M. S. Fox, Constraint-guided scheduling - a short history of research at CMU, *Comput. Ind.* 14, 1990, 79-88

Fre78 E. C. Freuder, Synthesizing constraint expressions, *Commun. ACM* 11, 1978, 958-966.

FS84 M. S. Fox, S. F. Smith, ISIS - a knowledge-based system for factory scheduling, *Expert Syst.* 1, 1984, 25-49.

FS90 M. S. Fox, K. Sycara, Overview of CORTES: a constraint based approach to production planning, scheduling and control, *Proceedings of the 4th International Conference on Expert Systems in Production and Operations Management*, 1990, 1-15.

GJ79 M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, 1979.

Har73 J. Harrington, *Computer Integrated Manufacturing*, Industrial Press, 1973.

KA87 J. J. Kanet, H. H. Adelsberger, Expert systems in production scheduling, *Eur. J. Oper. Res.* 29, 1987, 51-59.

Kan86 Kandel, A., *Fuzzy Mathematical Techniques with Applications*, Addison-Wesley, Boston, Mass., 1986.

Kus86 A. Kusiak, Application of operational research models and techniques in flexible manufacturing systems, *Eur. J. Oper. Res.* 24, 1986, 336-345.

KSW86 M. V. Kalkunte, S. C. Sarin, W. E. Wilhelm, Flexible Manufacturing Systems: A review of modelling approaches for design, justification and operation, in: A. Kusiak (ed.), *Flexible Manufacturing Systems: Methods and Studies*, Elsevier, 1986, 3-28.

LGW86 A. J. Van Looveren, L. F. Gelders, N. L. Van Wassenhove, A review of FMS planning models, in: A. Kusiak (ed.), *Modelling and Design of Flexible Manufacturing Systems*, Elsevier, 1986, 3-32.

Mac77 A. K. Mackworth, Consistency in networks of relations, *Artif. Intell.* 8, 1977, 99-118.

McN59 R. McNaughton, Scheduling with deadlines and loss functions, *Manage. Sci.* 12, 1959, 1-12.

Mon74 U. Montanari, Networks of constraints: Fundamental properties and applications to picture processing, *Inf. Sci.* 7, 1974, 95-132.

MS92a K. Mertins, G. Schmidt (Hrsg.), *Fertigungsleitsysteme 92*, IPK Eigenverlag, 1992

MS92b W. Mai, G. Schmidt, Was Leitstandsysteme heute leisten, *CIM Management* 3, 1992, 26-32.

NS91 S. Noronha, V. Sarma, Knowledge-based approaches for scheduling problems, *IEEE Trans. Knowl. Data Eng.* 3, 1991, 160-171.

PI77 S. S. Panwalkar, W. Iskander, A survey of scheduling rules, *Oper. Res.* 25, 1977, 45-61.

RM93 A. Ramudhin, P. Marrier, An object-oriented logistic tool-kit for schedule modeling and representation, *Proceedings of the International Conference on Industrial Engineering and Production Management*, Mons, 1993, 707-714.

Ran86 P. G. Ranky, *Computer Integrated Manufacturing: An Introduction with Case Studies*, Prentice-Hall, Englewood Cliffs, N.J., 1986.

Sch89a G. Schmidt, *CAM: Algorithmen und Decision Support für die Fertigungssteuerung*, Springer, Berlin, 1989.

Sch89b G. Schmidt, Constraint satisfaction problems in project scheduling, in: R. Słowiński, J. Węglarz (eds.), *Advances in Project Scheduling*, Elsevier, 1989, 135-150.

Sch91 A.-W. Scheer, *CIM - Towards the Factory of the Future*, Springer, 1991.

Sch92 G. Schmidt, A decision support system for production scheduling, *Journal of Decision Systems* 1, 1992, 243-260.

Sch94 G. Schmidt, How to apply fuzzy logic to reactive scheduling, in: E. Szelke, R. Kerr (eds.), *Knowledge Based Reactive Scheduling*, North-Holland, 1994, 57-57.

Sch96 G. Schmidt, Modelling Production Scheduling Systems, *Int. J. Prod. Econ.* 46-47, 1996, 106-118.

Sch98 G. Schmidt, Case-based reasoning for production scheduling, *Int. J. Prod. Econ.* 56-57, 1998, 537-546.

SFO86 S. F. Smith, M. S. Fox, P. S. Ow, Constructing and maintaining detailed production plans: investigations into the development of knowledge-based factory scheduling systems, *AI Mag.* 7, 1986, 45-61.

Smi92 S. F. Smith, Knowledge-based production management: approaches, results and prospects, *Prod. Plan. Control* 3, 1992, 350-380.

SPP+90 S. F. Smith, S. O. Peng, J.-Y. Potvin, N. Muscettola, D. C. Matthys, An integrated framework for generating and revising factory schedules, *J. Oper. Res. Soc.* 41, 1990, 539-552

SS90 S. C. Sarin, R. R. Salgame, Development of a knowledge-based system for dynamic scheduling, *Int. J. Prod. Res.* 28, 1990, 1499-1512.

Ste85 K. E. Stecke, Design, planning, scheduling and control problems of flexible manufacturing systems, *Ann. Oper. Res.* 3, 1985, 3-121.

WBJ90 J. R. Wilfs-Brock, R. E. Johnson, Surveying current research in object oriented design, *Commun. ACM* 33, 1990, 104-124.

Wir76 N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

Zad65 L. A. Zadeh, Fuzzy sets, *Information and Control* 8, 1965, 338-353.

Zad73 L. A. Zadeh, The concept of linguistic variables and its application to approximate reasoning, Memorandum ERL-M411, UC Berkeley, 1973.