Scheduling In Dynamic Environments

By

Peter Burke

Submitted To The University Of Strathclyde

For

The Degree Of Doctor Of Philosophy

Department Of Computer Science

University Of Strathclyde

Glasgow 1989

To Mum and Dad, for their long standing support and sacrifice. Without it, this work would not have been possible.

Acknowledgements

I wish to express sincere thanks to my supervisor, Professor J.T. Buchanan, for his speed of response and invaluable guidance throughout the course of my work. I am also greatly indebted to Patrick Prosser, a fellow group member, for his constructive criticism and comments at vital stages in my project.

My thanks are also due to all members and research students of the Department of Computer Science. Collectively they provided the working environment required to complete a task such as this. In particular thanks must go to Derek Masson, John Costello, David Pritty and Andrew Watson. I would also like to thank those of my industrial collaborators who contributed in some way to the project as a whole.

Finally, I would like to thank my family and friends for their constant support and encouragement. The most important member of this final group is undoubtedly Pauline Berry, for whose understanding and silent suffering I am extremely grateful.

Abstract

Much of the work in the area of automated scheduling systems is based on the assumption that the intended execution environment is static and deterministic. The work presented in this thesis is motivated by recognition of the fact that most real world scheduling environments are dynamic and stochastic. It views the scheduling task as one of satisfaction rather than optimisation, and maintenance over creation.

This thesis reviews existing work in the area and identifies an opportunity to combine recent advances in scheduling technology with the power of distributed processing. Within a suitable problem-solving architecture it is argued that this combination can help to address the fundamental problems of executional uncertainty, conflicting objectives and combinatorial complexity. A scheduling system, DAS, which employs such a problem-solving architecture, is presented. It is distributed, asynchronous and hierarchical, and requires careful management of problem-solving effort. DAS adopts an opportunistic approach to problem-solving and the management of problem-solving effort. The mechanisms which manage problem-solving effort within DAS are also presented. In conclusion it is argued that the architecture and mechanisms presented lend themselves very well to the view taken of the scheduling task.

Table Of Contents

CHAPTER 1

Introduction

Whether the motive be survival or simply an improved return on capital investment, the vast majority of industry is interested in improving its manufacturing performance. Poor methods of production scheduling have for a long time been recognised as a major deterrent to achieving this goal. The potential rewards offered by this area of research have attracted much interest from a number of disciplines. Despite the fact that significant progress has been made in scheduling theory, industry has benefited little. This is largely a consequence of the fact that much of the early scheduling work assumes a *static* execution environment. The issues addressed by this thesis arise out of a recognition that the real world execution environment is highly *dynamic*.

1.1. The Scheduling Problem

Many attempts have been made to define clearly and concisely what is meant by the term scheduling. Most people would claim to have an understanding of what is meant, but detailed accounts vary a great deal. Within the Artificial Intelligence (AI) community it would appear that much of the confusion arises out of the unclear relationship between scheduling and planning. This confusion shows itself in everyday English where the terms *planning* and *scheduling* are used interchangeably and in fact, the New Collins Thesaurus identifies the word *plan* as a synonym for *schedule*. It is not surprising therefore, that it is often difficult to know exactly where to draw the line between planning and scheduling. Unfortunately, this is not a simple demarcation problem and some authors feel that the tasks are so interdependent that one cannot be performed in isolation of the other. M.S. Fox states that the tasks are inseparable [Fox M.S. '83] and he describes *scheduling* as a two stage process. The first stage, to generate a process routing, is defined as the product of a *planning* process while the second stage, the allocation of times and resources is

defined as the product of a *scheduling* process. This definition serves to enhance the view that the distinction between planning and scheduling is indeed a vague one.

To add further confusion, the degree of coupling necessary between the two tasks is dependent on the domain. In the job-shop, the domain with which M.S. Fox is primarily concerned, a very tight coupling is required. In such an environment it is sensible to iterate through the planning and scheduling processes. However, in heavy manufacturing industries which have a much higher momentum, it is not desirable to have such tight coupling. The planning horizon in such industries must often be much longer than would be sensible to perform a detailed allocation of resources.

There is no attempt within this thesis to provide absolute definitions of these terms; however it is appropriate to identify working definitions. The definitions given in [Fox B.R. et al '85b] have been adopted because they allow for maximum opportunism within the scheduling process. Justifications for the desirability of opportunism within scheduling are deferred until later. The definitions are shown below:

Planning:

> *given an initial world, a goal world, and a set of operators (a task and a set of facility capabilities), select a set of operators which will achieve the goal, and generate a minimal set of ordering constraints on operator application (a plan).*

Scheduling:

> *given a set of operators and minimal ordering constraints (a plan), and detailed knowledge of the execution environment (a set of facility availabilities), enforce further ordering constraints on operator application to achieve robust and time-efficient execution of the task (a schedule).*

This thesis is not concerned with planning, only with scheduling as defined above. This definition makes no attempt to identify desirable qualities of a schedule other than that it be robust and time-efficient. It does not concern itself with resource-efficiency or any other potential measure of

quality. As will be discussed shortly, it is not a trivial task to identify a measure of schedule quality.

1.2. Difficulties Inherent In Scheduling

Production scheduling, regardless of the domain, is a difficult task from both theoretical and practical standpoints. Theoretically, the combinatorics of all but the most trivial of scheduling tasks prohibit searching for an *optimal* solution. Even if it were possible, via some super computer, to circumvent the combinatorial problems of scheduling, significant practical obstacles remain. Any serious attempt to schedule in a real world environment must address both the theoretical and practical issues.

From a theoretical point of view, the problem is one of combinatorial complexity with most scheduling problems having an exceedingly large number of potential solutions. For example, ignoring the possibility of alternative routings and gaps in a schedule, the number of possible schedules which result from sequencing 10 orders through 5 processes is $(10!)^5$. The number of possible solutions to a problem grows exponentially with the number of orders, alternative production plans, alternative resources and the many other parameters which form a scheduling problem. In fact, except for some highly specific cases, finding an optimal solution to a scheduling problem has been shown to be NP-complete [Ullman '76]. A very effective demonstration of the combinatorial complexity of apparently trivial scheduling tasks is given by French in his introductory text [French '82] on the subject.

The practical obstacles to effective production scheduling in the real world are just as daunting as the problem of combinatorial complexity. Scheduling in real world domains introduces at least two additional difficulties, that of executional uncertainty and the more fundamental problem of measuring the quality of a schedule. There are many sources of executional uncertainty in any industrial scheduling application, with machine failures, delayed jobs and scrapped jobs all being inevitable occurrences. In order to be able to deal with such

unforeseen events, a scheduling system must be integrated with some form of shop-floor reporting system. However even with the necessary infrastructure, there is no guarantee that operatives on the shop-floor will report events correctly and timeously, or that they will execute the instructions passed to them correctly and timeously.

Identifying the scheduling objectives of an organisation, and therefore a measure of schedule quality, is not a simple task. This is due partly to the fact that there are numerous candidate objectives and partly to the fact that the various candidates are generally conflicting in nature. Mellor [Mellor '66] identifies 27 distinct scheduling objectives most of which conflict with each other. The task of defining the criteria by which a schedule can be considered optimal is therefore very difficult. In most environments a compromise of the various identifiable objectives is desired. Unfortunately, this compromise is highly dynamic in nature as it is influenced by many factors such as the state of the shop-floor, the order book and the current organisational objectives.

1.3. Economics of Manufacturing

The primary objective of manufacturing organisations is to make a profit in financial terms. Net Profit, Return on Investment and Cash Flow are commonly used as financial measures to indicate the state of health of a company. Although very useful in company performance analysis, such measures are generally very difficult to monitor on a daily basis. This renders them of little value as aids in the day to day running of the shop-floor. Many attempts have been made to convert these clearly defined financial measures into useful operational measures which can be used to guide control of the shop-floor. As discussed above, many operational measures can be identified but the problem of resolving conflicts between the various measures presents considerable difficulties. Despite the fact that it is not clear how best to use such measures, it is useful to identify some of the more commonly used measures and to discuss the rationale behind them, in order to gain an understanding of the economics of the shop-floor.

Inventory levels, which includes raw materials, work in progress (WIP) and finished products are commonly used as indicators of shop-floor performance. Inventory is viewed as a useful metric because it is a measurable quantity and has an impact on financial performance. When considered in isolation the relationship is simple, since financial measures which penalise stock levels will improve as inventory levels decrease. This view of the relationship between inventory levels and financial performance can be supported by several arguments, the most prominent being that the higher inventory levels are maintained the more capital is tied up as inventory. Maintaining low inventory levels has the beneficial effect of permitting reduced space requirements in both storage and shop-floor which, in turn, has financial consequences in terms of heating, maintenance and transportation costs.

Manufacturing throughput is another commonly used operational measure. It is interesting to note that two apparently similar measures, throughput and output, exist within the operations management vocabulary. Throughput is defined as the production of items which will immediately generate revenue through sales, making the link between throughput and financial performance clear. The distinction between throughput and output, that finished items are produced for a customer and not for inventory, shows a commonality in the rationale behind both inventory levels and throughput as operational measures.

Unfortunately, the two measures are not quite as compatible as this suggests and there is a fine balance to be struck between minimising inventory levels and maximising throughput. As inventory levels drop, throughput performance becomes vulnerable to disruptions on the shop-floor. A shop-floor disruption which results in a drop in throughput may undo the financial benefits gained by operating with low inventory levels. Conversely, to achieve high throughput levels in the face of executional uncertainty may require unacceptably high inventory levels. The point at which to balance inventory and throughput is difficult to identify, let alone achieve on the shop-floor as it is influenced by uncontrollable variables such as interest rates, taxation levels, exchange rates, raw materials and energy costs.

1.4. Traditional Approaches To Manufacturing Management

Over the years, production managers have adopted a variety of strategies to the highly complex task of controlling a manufacturing environment. Unfortunately, none of the approaches tried so far have provided a satisfactory solution to the problem. This section reviews traditional approaches to manufacturing management in order to identify the need for a scheduling component. In the course of this review it will also be made apparent at which point in the manufacturing management structure a scheduling component is required.

Inventory Control

The earliest mechanisms used to manage manufacturing relied on the monitoring of inventory levels of finished products. Whenever the inventory level of a particular product fell below a certain level, a new batch was ordered. The interested reader is referred to [Whitin '57] which gives a good introduction to the underlying principles of inventory control techniques. All inventory level mechanisms have three basic parameters which must be given *optimal* values if they are to be successful in producing the required products efficiently and on time. The first parameter to be set is the frequency with which inventory is monitored. Depending on the level of automation on the shop-floor, this can be a time consuming and relatively expensive exercise. On the other hand, if it is not performed frequently enough to ensure reasonably accurate data for inventory levels, it becomes a pointless exercise.

The second parameter which must be set is the level at which it is deemed necessary to order a new batch of a particular product. The *optimal* setting for this parameter is one which ensures that the new batch will be produced just before demand for the product in question exceeds the available inventory. In order to give this parameter an *optimal* value, it is necessary to have accurate figures for both the lead time and demand for each product being manufactured.

The third parameter, the size of the batch to reorder, is the one which has received most attention from production managers. The *optimal* value for batch size is a function of the cost of storing and the cost of manufacturing a product. As batch sizes increase, setup costs and other

manufacturing costs which are fixed will decrease per item produced. On the other hand, large batches must eventually be completed and become part of the finished goods inventory, therefore boosting inventory holding costs. Perhaps the most commonly used technique to calculate a value for batch size is the Harris-Wilson Lot-Size Formula shown below:

$$Q=\sqrt{\frac{2RC_3}{C_1}}$$

where

Q = optimum reorder quantity

R = average demand (items/year)

C_3 = cost of setup and reordering

C_1 = inventory holding cost (cost/item/year)

Just as in the case of setting the re-order level, accurate data are required for product demand, setup costs and inventory holding costs when calculating re-order batch size.

In practice, inventory control techniques rely on several simplifying assumptions to make them workable. These assumptions include the notion that each product can be considered in isolation and that both product demand and lead times are constant. Unfortunately, these assumptions do not hold in a typical manufacturing environment. For many applications, inventory control approaches do not perform well, suffering from problems of high inventory holding costs and a lack of responsiveness to changing demand. Further, as a result of being slow to respond to change, there is always a risk of inventory becoming obsolete.' These problems are a consequence of the fact that inventory control approaches only respond to changes as and when they occur; they make no attempt to anticipate change. These weaknesses aside, it should be noted that inventory control approaches do not address the issue of detailed scheduling at all.

MRP

By the 1970s, fierce competition from Japan combined with soaring interest rates forced managers in the West to adopt strategies aimed at reducing inventory levels while at the same time

increasing manufacturing responsiveness. To this end, managers turned to predictive methods of managing production, and Materials Requirement Planning (MRP) [Orlicky '75] in particular. The rationale behind MRP systems is that they enable managers to look to the future, thus allowing them to increase inventory levels only when a perceived need justifies it. Activities in an MRP system fall into three basic stages, as shown below:

(1) Produce a Master Production Schedule (MPS)
(2) Perform Materials Requirements Planning (MRP)
(3) Perform detailed scheduling.

Producing an MPS, which is usually performed as a manual task, gives a forecast of the required quantities of products for some period in the future. It is derived after considering expected customer demands, available capacity and the levels of inventory currently available for each product. It is obviously very difficult to achieve a high degree of accuracy when producing an MPS.

The output of the MPS stage and the required bills of materials, information concerning the materials and components necessary to make a particular product, is fed into the MRP stage. The MRP system then delivers a list of required components and for each component specified, states the quantity required and at what time manufacture should commence. This stage of the process also suffers from inaccuracies, the most serious being in the figures used for lead times. Lead times are assumed to be constant, regardless of the situation on the shop-floor. Over time, the bill of materials can also become a source of inaccuracy when products are modified and the bill of materials is not updated to reflect the modification.

The final stage in an MRP system is detailed scheduling. This is the task of allocating resources to jobs over time. This is a very complex problem, made more difficult by errors fed into the system at the higher levels. It is not unusual for the scheduling task set by the higher levels to be a problem with no solution. Once again, the task of detailed scheduling has been omitted from the set of automated processes.

As experience of MRP systems grew, it became evident that overall performance could be improved if the system was expanded to cover other aspects of company activities such as business and production planning. At the same time, increases in computing power made detailed capacity analysis feasible. This expanded system, called Manufacturing Resource Planning or MRP II [Wight '81], with an ability to perform detailed capacity analysis, should have made the scheduling tasks passed to the lowest level of the system more solvable. Unfortunately, like other stages in the MRP system, this additional stage served only to highlight the need for good quality data if any potential benefit is to be realised. As in MRP, the detailed scheduling problem is not dealt with in MRP II.

OPT

By the 1980s it was apparent that MRP systems would not provide all the answers. The early part of the decade saw the introduction of an alternative methodology for managers, Optimised Production Technology (OPT) [Harrison '85]. OPT provides the functionality of traditional MRP systems in a manner which attempts to overcome some of the fundamental problems associated with MRP systems. Like MRP II, OPT recognises the need to take account of available capacity before performing detailed scheduling. However, unlike MRP II it does recognise that it is not sufficient to prioritise the release of orders by their predicted lead times and subsequently consider capacity constraints. The lead time of an order is dependent on the capacity requirements of other orders in the schedule and therefore must be considered in conjunction with capacity constraints. OPT views capacity constraints in terms of bottleneck resources, based on the premise that if bottleneck resources achieve 100 % utilisation throughput will be maximised.

OPT implements these concepts in a software package which first schedules bottleneck resources in an "*optimal*" manner, possibly resulting in alterations to predicted lead times, and later deals with non-bottleneck resources. It is the OPT literature which claims that bottleneck resources are scheduled in an optimal manner. However, as has been discussed already, it is very difficult to define what is actually meant by schedule optimality therefore casting doubt on this

claim. It is highly likely that what is meant is that OPT achieves some local form of optimality with respect to bottleneck resource utilisation. Although it provides an improvement to traditional MRP systems, OPT retains the notion that scheduling is essentially an off-line task. Bottleneck resources are not static entities as assumed by OPT, since in the face of real world uncertainty they become highly dynamic. A schedule which achieves maximum throughput for a predicted set of bottlenecks is likely to become highly suboptimal when the set of bottlenecks change.

JIT

Despite large scale investment in technologies to support manufacturing management, Western industry remains significantly less efficient than its Japanese counterpart. It is not surprising then, that the West has taken an interest in the manufacturing methods used by the Japanese. The methodology, or some would say philosophy, creating most interest in the West at the moment is Just-In-Time (JIT) manufacture. The JIT philosophy [O' Grady '88] is based on the four objectives shown below:

(1) Attack fundamental problems.
(2) Eliminate waste.
(3) Strive for simplicity.
(4) Devise systems to identify problems.

The first objective is concerned with removing problems such as bottleneck resources, unreliable machines, high scrap rates and poor quality suppliers. It is argued that having removed these fundamental problems both WIP levels and product lead times can be significantly reduced. For the second objective, any process which does not add value to the product, such as transportation, inspection, setup and storage is defined as waste. To achieve the third objective of simplicity, two areas are targeted, the first being material flow. A JIT philosophy requires a move away from traditional process layout, where resources are located near similar resources, to a product flow line where the resources to produce a family of products are located together. The second area targeted concerns the control of flow of materials through the shop-floor. The method favoured by proponents of JIT is the pull/Kanban system. This requires work to be pulled through the factory

by demand rather than the traditional approach which pushes work through. Work centres, organised in flow lines, only produce work when there is a demand for it upstream. The pull/Kanban system is one of two systems commonly used to satisfy the fourth objective, the other being the use of statistical quality control to identify problems.

Presented in this manner, JIT appears to remove the need for complex production scheduling. Indeed, this has been shown to be the case in several successful applications such as the Toyota Corolla production line in Japan and at Harley Davidson in the USA. However, for many applications JIT may not be a workable manufacturing methodology. It is not always possible to reduce scrap rates significantly; there may be technological problems. Cost or space limitations may be inhibiting factors when the prescribed solution is to increase capacity at bottleneck resources. Poor quality suppliers may be a fact of life if the supplier in question has no competition. Where many resources are shared between product lines, it is unlikely to be feasible to convert from a process layout to product flow lines. Within such a framework a simple pull/Kanban system to production scheduling is unlikely to be effective and therefore does not remove the need for detailed production scheduling.

Conclusion

With a single exception, the approaches to manufacturing management discussed in this section present a common view of the scheduling problem. JIT, the odd one out, takes the view that it is possible to create an environment in which scheduling becomes a trivial task while all the others view scheduling as a static and deterministic optimisation problem to be performed off-line. Within this framework, the link between the contributions made by the Operational Research (OR) community to the scheduling problem and the traditional view held by industrial management becomes clear. Traditional applications of techniques such as *Dynamic Programming*, *Branch and Bound Search* and *Integer Programming* share this traditional view of scheduling.

This thesis, and a growing number in the Artificial Intelligence community, perceive scheduling to be a dynamic and stochastic satisfaction problem. The traditional view fosters the

false notions that a static model of the problem is adequate and that the resulting schedule will be executed precisely as specified. These are not the only problems facing traditional approaches. The techniques developed within the OR community suffer from an inability to represent the information necessary to solve scheduling problems and are computationally very expensive. The consequence of having an inadequate problem representation is that when a solution is produced, it solves the modelled problem and not the real one. In an attempt to circumvent the problems of computational expense, the model of the problem being addressed is often simplified, thus aggravating the discrepancy between the generated and desired solution.

1.5. Contributions of This Work

Perhaps the largest contribution which Artificial Intelligence makes to any problem, is an ability to represent significantly more of the available knowledge relating to the task. Representations of both the problem and scheduling heuristics can be much richer. Through the 1980s there has been a growing realisation that richer knowledge representations alone are not sufficient to solve the difficulties present in real world scheduling problems. The real world scheduling environment is dynamic and stochastic, not static and deterministic. For an automated scheduling system to be successful in real world domains it must address the issues of executional uncertainty, conflicting scheduling objectives and combinatorial complexity. Consequently, it is necessary to review the very nature of the scheduling task. This thesis holds the view that the scheduling problem in general and manufacturing in particular is an on-line, dynamic and stochastic satisfaction problem.

The contributions of the work presented in this thesis are twofold. Firstly, a problem-solving architecture suitable for addressing the scheduling problem when viewed in this way is presented. The architecture is hierarchical, asynchronous and distributed in nature. This has obvious benefits in manufacturing domains which are themselves hierarchical, asynchronous and distributed. The task of managing problem-solving effort appropriately is a major concern for any problem solver.

This is particularly true for a distributed asynchronous problem-solver. The second contribution of this thesis concerns the issues of managing problem-solving effort within such a problem-solver effectively. This thesis proposes an opportunistic approach to the difficult task of managing problem-solving effort. The approach presented here was shaped largely by the requirements of a scheduling system which must operate in real world scheduling domains.

1.6. Context of Work

ALVEY project, *IKBS Production Control in Heavy Manufacturing Industry* provided financial support for the work described in this thesis. The project was a collaborative effort involving the University of Strathclyde, British Alcan Plate Ltd. and YARD Ltd.. British Alcan Plate Ltd. provided the project with a demonstrator site on which to develop and test the work being carried out at Strathclyde. The site selected is an aluminium plate manufacturing plant located at Kitts Green in Birmingham. This site was chosen for two reasons. Firstly, it already had the infrastructure necessary to support an IKBS scheduler. This includes integrated systems to perform tasks such as sales order handling, process planning and shop-floor reporting. Secondly, it has very real scheduling problems.

1.7. Thesis Plan

Chapter 2 presents a review of work considered necessary for a full appreciation of the remainder of this thesis. It views scheduling as a specialisation of problem-solving and starts by reviewing techniques available for focusing problem-solving effort. As many of the techniques reviewed were introduced before scheduling had established itself as a research area in AI, the development of these techniques is traced largely through the planning literature. Having reviewed the techniques available, the application and extension of these techniques in scheduling systems is considered in some detail. It is argued that recent advances in scheduling technology should be combined with advances in distributed computing technology to address the difficulties of the

dynamic, stochastic and distributed nature of the real world.

Having identified the potential benefits of such a combination, chapter 3 introduces DAS, a scheduler with a problem-solving architecture which facilitates this. A detailed account of how DAS views the scheduling problem is presented before a description of the system itself. Both the motivation for the architecture and the architecture itself are discussed.

In order to gain any benefit from the problem-solving architecture introduced in chapter 3, it is necessary to support it with mechanisms capable of managing problem-solving effort effectively. The mechanisms which focus and co-ordinate problem-solving effort in DAS are presented in chapter 4. Problem-solving effort is managed by a constraint maintenance system, conflict resolution techniques, an operation priority mechanism and agent communication via message-passing.

Chapter 5 presents a case analysis of DAS. It demonstrates that DAS has both an appropriate problem-solving architecture and the mechanisms necessary to manage problem-solving effort in real world scheduling environments.

Finally, chapter 6 summarises chapters 1 to 5, considers areas of future work and presents the conclusions of this thesis. The future work section discusses both extensions and empirical analysis, and is followed by a few concluding remarks. Here it is concluded that the architecture and mechanisms presented lend themselves very well to the opportunistic approach required to address the problem of scheduling in a dynamic environment.

CHAPTER 2

A Review of Problem-Solving Techniques and Scheduling Systems

2.1. Introduction

Scheduling is an example of a problem-solving activity, and as such has a great deal in common with other specialisations of problem-solving. It is appropriate therefore to review some general problem-solving techniques, their application to scheduling, and individual scheduling systems. This review deals first with some established techniques, followed by a detailed analysis of specific scheduling systems.

The terms *Problem*, *Algorithm* and *Heuristic*, formalised in papers describing the Logic Theory (LT) machine [Newell et al '57], are now commonly used throughout the problem-solving literature. Generally speaking, an *algorithm* can be defined as a generator of solutions which is guaranteed to find a solution to a problem if one exists. A *heuristic*, on the other hand, is defined as a process that may solve a given problem if a solution exists, but offers no guarantee of doing so. In both heuristic and algorithmic cases, the computational cost of generating and evaluating potential solutions is significant. For problems of combinatorial complexity such as scheduling, this computational cost is a major source of concern. It is not the computational cost associated with each individual potential solution that is of significance, but rather the combined cost of the many potential solutions.

Efficient algorithms exist for only a very few problem-solving activities. Finding the maxima for simple differentiable functions or special case scheduling tasks involving one or two machines (eg. Moores's algorithm [Moore '68], Lawlers's Algorithm [Lawler '73] and Johnson's algorithm [Johnson '54]) provide examples of problem-solving activities with known algorithmic solutions. Complete enumeration, or the British Museum algorithm as it is sometimes known, offers an algorithmic solution to all problem-solving activities. Unfortunately, the computational

cost of the British Museum Algorithm is prohibitive for all but the most trivial of problems. Consequently, problem-solving research, including scheduling, has concentrated mainly on methods of improving the performance of heuristic search.

Casting a problem as a state-space search involves traversing a search space made up of nodes in an attempt to find a solution. Each node is different to every other node in the search space and corresponds to one partial solution to the problem. When viewing scheduling as an example of state-space search, each node of the search space corresponds to an instantiation of a possible schedule. Many nodes, sometimes all, will correspond to illegal schedule solutions. It is the aim of heuristic search mechanisms to focus problem-solving effort on areas of the search space which contain legal solutions to the problem at hand, thereby improving search performance. Within state-space search, it is possible to identify two fundamental tactics available to such mechanisms. They can attempt to focus effort either by *reducing the number of nodes* to be visited, or by *determining the order* in which they are to be visited. Both can have a major impact on the efficiency of a search mechanism, and ideally should be used together to achieve maximum benefit.

This review regards scheduling as a heuristic search process and discusses techniques and systems with respect to their ability to effectively focus problem-solving effort by means of the two tactics mentioned above. In section 2.2, significant developments in the techniques available for focusing problem-solving effort which existed prior to the establishment of scheduling as a field in AI are discussed. The development of these techniques is traced largely through the planning literature for two reasons. Firstly because it is an established field and secondly because of its close relationship to scheduling. Existing scheduling systems are reviewed in section 2.3 giving an insight into how the techniques discussed in section 2.2 have been applied to, and extended within, the scheduling domain.

2.2. Techniques

2.2.1. Evaluation Functions

The use of evaluation functions to augment blind search techniques such as depth-first and breadth-first, represents one of the earliest attempts to utilise heuristic information to enhance search performance. Evaluation functions provide a means of rating the individual nodes of a search space. A rated search space is one for which an evaluation function exists. An evaluation function attempts to give a measure of how similar the state represented by a node is to a goal node. A goal node is a node which represents a state considered to be a solution to the search. Within algorithms A* [Nilsson '71] and B* [Berliner '79] evaluation functions are used to improve search order. Evaluation functions are also employed within the branch-and-bound [Land et al '60] search technique. While heuristic methods provide no guarantee of success, it is hoped that by using evaluation functions a goal node will be found earlier than if a blind search technique was used.

The most obvious use of an evaluation function is as a guide when selecting which node to expand next. However, they can also be used to select the most promising operator to apply at any given point in the search [Fox M.S. '83], thus guiding the method of expansion. Later search algorithms, such as Beam Search [Winston '77a], use evaluation functions to both order and prune the search space. Within a beam search, pruning is achieved by expanding only the *n* nodes which are rated most highly at each ply in the search. A general theory of the use of evaluation functions to guide search is given in [Hart et al '68].

In [Nilsson '71], a general problem-solving text, a distinction is made between *state-space* search methods and *problem-reduction* search methods. Incremental state-space search methods deal only with problem *states* and the *operators* which generate one state from another. On the other hand, problem-reduction methods attempt to reason backwards from the problem to be solved using *problem-reduction operators*, to establish subproblems. Problem-reduction methods are discussed here because they provide further examples of the use of evaluation functions. The

problem-reduction process is applied iteratively until the original problem is resolved into a set of primitive problems. In order to maintain a common metric with which to compare the various techniques, this review considers problem-reduction as a technique aimed at enhancing the performance of state-space search and views state generation within state-space methods as a trivial form of problem reduction. This is not wholly inconsistent with Nilsson's view. He suggests that problem-reduction methods can be viewed as a means of enumerating the separate searches for subpaths between proposed stepping stones in the state-space, and for monitoring the progress towards assembling subpaths into complete solutions.

Problem-reduction methods are used to search AND/OR graphs rather than state-space graphs. Algorithms such as mini-max [Winston '77b] and Alpha-Beta [Knuth et al '75] use evaluation functions during the search for a solution tree, a structure which identifies a set of possible "stepping stones" for a state-space search. As in state-space methods, evaluation functions attempt to select the node whose expansion is most likely to succeed in leading to a solution.

While evaluation functions can perform a useful role during problem-solving, they are of limited use in complex problem domains. As the degree of domain complexity increases, so too do the problems of encoding the appropriate heuristic knowledge into an evaluation function.

2.2.2. Means-end Analysis

Means-end analysis is a technique commonly used by human problem solvers when performing tasks such as route planning or theorem proving in elementary symbolic logic. It is a problem-reduction type search method based upon detecting the difference between the current state and the desired goal state. Having detected the difference, this knowledge is then used to generate subgoals which, if achieved, will reduce the difference between the current state and the goal state. This subgoal generation mechanism is applied repeatedly until the generated subgoal matches the desired goal state.

Although other researchers, for example [Duncker '45], had previously reported the use of means-end analysis by subjects solving problems, it was Newell, Shaw and Simon who first specified the technique rigorously and implemented it on a computer. It was in their work in developing the General Problem Solver (GPS) [Newell et al '59] that they implemented means-end analysis as a problem-solving approach. A simple example of the use of means-end analysis to prove a theorem in elementary symbolic logic is given below. For the example, one axiom and two rules of inference are required.

Axiom: (p OR p) implies p

Substitution: any expression may be substituted for any variable in any theorem, provided that the substitution is made throughout the theorem.

Replacement: a connective can be replaced by its definition, and vice versa, in any of its occurrences. (eg. p implies q is defined as Not-p OR q).

Example:

(p implies Not-p) implies Not-p	(Theorem to be proved)
1. (A OR A) implies A	(axiom)
2. (Not-A OR Not-A) implies Not-A	(Subs. of Not-A for A)
3. (A implies Not-A) implies Not-A	(Repl. of OR with implies)
4. (p implies Not-p) implies Not-p	(Subs. of p for A; QED)

At each step of the proof, an attempt is made to reduce the difference, measured in some ad-hoc way, between the current expression and the desired goal expression.

Means-end analysis is a heuristic search mechanism which employs both search space pruning and search ordering to good effect. The search is guided from node to node in an order which reduces the difference between the current node and the goal node. As mentioned in the preceding section, problem-reduction search methods such as means-end analysis make use of AND/OR graphs rather than state-space graphs, thereby assisting in the task of pruning of the search space. When searching an AND/OR graph, some nodes can be identified as being solved or unsolvable. A solved node is a terminal node which corresponds to a primitive problem, a nonterminal node with a number of OR successors at least one of which is solved, or a nonterminal node with a number of AND successors all of which are solved. An unsolvable node is a

nonterminal node with no successors, with a number of OR successors all of which are insolvable, or with a number of AND successors at least one of which is unsolvable. Pruning is achieved by noting that it is never necessary to visit a node which is a descendant of a solved node, or a descendant of a node which has already been established to be unsolvable.

This technique relies on two implicit assumptions which may be justifiable when performing tasks such as elementary symbolic logic theorem proving, but not when scheduling in a manufacturing environment. The first assumption is that the problem solver is given a well defined goal state. A statement such as "(p implies Not-p) implies Not-p" is a well defined goal state, whereas "produce a schedule which satisfies as many constraints as possible" is not. The second assumption is that it is possible to measure the difference between the current state and goal state in a meaningful way. Within a complex scheduling task, it is necessary to use grossly over-simplified metrics when measuring the difference, resulting in misguided searches at anything other than a fairly high level. Perhaps it is an inability to measure effectively the difference between the current schedule state and a goal state which makes scheduling a difficult task for humans. Without this ability, the "default problem-solving approach" of humans, means-end analysis, becomes an unguided search.

2.2.3. Least Commitment

The term, least commitment, is perhaps one of the most overused terms within AI. In its broadest sense, a least commitment approach is one which allows decisions to be deferred until some time in the future in the hope that more information, relevant to the decision, will become available. The goals of such a technique are typically twofold, firstly to avoid committing to poor decisions prematurely and secondly to factor the problem in such a way as to make it more manageable. Within scheduling applications, the two most common forms of a least commitment approach are hierarchical search and non-linear planning.

Hierarchical search provides a method for dealing with very large search spaces. When searching in a hierarchical manner, the search proceeds from the highest, least detailed level to the

lowest, most detailed level. The first explicit use of hierarchical search within planning was in ABSTRIPS [Sacerdoti '74]. Like STRIPS [Fikes et al '71], ABSTRIPS represents operators as rules with pre-conditions and post-conditions. Pre-condition variables are separated into levels of importance. The hierarchical effect is achieved by having the pre-conditions contain only variables considered important at the current level of planning.

Hierarchical search takes place in a stratified, rated search space in which nodes are partitioned into levels. A particular level in a stratified search space completely dominates a lower level only if the sum of ratings of nodes along any search path in the lower level is less than any rating for any node in the higher level. Therefore, a top down search of a stratified, rated search space will have the same results as a complete search of the collapsed space if and only if the space is completely dominated from the top level down. Unfortunately, it is rarely possible to identify these conditions prior to search. It should also be noted that a stratified, rated search space which is completely dominated for one evaluation function may not be for others.

Search ordering within a hierarchical search is explicitly from the highest level down to the lowest level. This says nothing about search order within levels or the quality of this top down ordering for the problem being addressed. Only in the situation in which each level completely dominates its lower levels can this search ordering be considered ideal. However, in most situations, despite a lack of complete dominance, an appropriate hierarchical decomposition will give a reasonable search ordering, and more importantly will greatly prune the search space. Pruning is achieved by excluding all nodes in the search space which are not consistent with existing, higher level decisions.

Another prominent implementation of a least commitment approach comes in the form of non-linear planners. Non-linear planning systems such as NONLIN [Tate '75] and NOAH [Sacerdoti '75] introduced this technique. A non-linear planner is one which does not sequence tasks until forced to do so. It reduces the amount of backtracking required to produce a legal plan because no unnecessary sequencing decisions are taken. Non-linear planning systems also offer

benefits to the scheduling task which often follows plan generation. The output of a non-linear planner may well be a non-linear plan. A non-linear plan is one in which at least two of the activities in the plan can be performed either before or after each other. A strictly linear plan is not capable of representing this opportunity, thus depriving the scheduling system of a degree of discretion when solving its task. Non-linear plans do not actually help order search during scheduling. However, viewing planning and scheduling as part of a larger task, it can be argued that non-linear planning does perform a search ordering function by deferring certain decisions until during the scheduling phase where they can be more effectively dealt with.

The least-commitment approach as a whole, and hierarchical search and non-linear planning in particular, is a useful technique within scheduling. In general it helps to factor the problem in some way, while also leading to a reduction in the number of backtracks required during search.

2.2.4. Constraint Analysis

There are many examples of the use of constraint analysis to improve the performance of heuristic search. By examining the constraints imposed on a problem, it is often possible to exclude a great number of nodes from the search space and occasionally to identify a preferred search order. The majority of the early work on constraint analysis concentrates very much on pruning the search space rather than ordering the search.

REF-ARF [Fikes '70] represents one of the earlier works in constraint analysis. It is a heuristic problem-solving program which accepts problems stated in REF, a nondeterministic programming language. ARF, the problem-solving component, applies constraint satisfaction and heuristic search techniques to solve problems presented to it in this way. Rather than performing a search over the entire search space, REF-ARF utilises the constraints of the problem to exclude certain nodes. Although not removing the combinatorial nature of the problem, constraint analysis is being used to reduce the search space to be traversed. CONSTRAINTS [Steele '80], a more recent work, provides another example of constraint analysis. It provides a language for representing hierarchical constraint networks. Within such networks, constraint propagation and

dependency analysis can be used to identify inconsistent subsets of constraints. Constraint analysis of this nature allows both search space pruning and ordering.

Moving closer to the scheduling domain, MOLGEN [Stefik '81] combines planning with constraint analysis. MOLGEN decomposes its problem into subproblems, therefore introducing a requirement for communication between subproblems. Communication is achieved by constraint-posting, a technique of propagating constraints from one subproblem to another subproblem. In this way constraints from related subproblems are utilised to reduce the search space of each other, while at the same time ensuring mutual compatibility.

As will become clear in section 2.3, constraint analysis can play a significant role in search ordering as well as search pruning. For example, one reasonable search ordering heuristic would be to deal with the most constraining constraints first. This corresponds directly to the fail first principle of Haralick [Haralick et al '80].

2.2.5. Opportunistic Reasoning

The fail first principle referenced above is an example of an opportunistic approach to problem-solving. The earliest example of a system which performs opportunistic reasoning is Hearsay II [Erman et al '80]. It does so by allowing problem-solving to be both data and goal directed, as well as permitting island-driving search as opposed to left to right search. This combination of features is identified as opportunistic reasoning in [Hayes-Roth et al '79].

Hearsay II also introduced the blackboard architecture, an enabling vehicle for opportunistic reasoning. Hearsay II has policy modules which focus the attention of the system in an opportunistic manner. They dynamically determine what parts of the search space require attention and sequence knowledge source executions accordingly.

Opportunistic reasoning can achieve both search ordering and effective search space pruning. By focusing effort in areas requiring attention, a search order is dynamically generated in accordance with the current problem-solving state. The act of instantiating a scheduling decision

can be used to eliminate areas of the search space which are not compatible with that decision. Therefore it seems reasonable to focus the attention of the system in such a way that decisions in highly contended areas of the schedule are dealt with first. By doing this, decisions in highly contended areas of the schedule are allowed to constrain less highly contended areas, thus pruning the search space. It is hoped that nodes pruned in this way are representative of a number of similarly rated nodes, and that not all of these nodes have been pruned.

Opportunistic reasoning is a very useful approach to problem-solving. However, it is not always a simple task to detect the most appropriate opportunity to focus on. Nor is it easy to identify the appropriate response to the selected opportunity. This will be discussed in greater detail later.

2.3. Scheduling Systems

This section reviews the major AI-based scheduling systems in order to determine how well they manage problem-solving effort when searching for a solution. Due to the dynamic nature of most scheduling environments, it is necessary to consider their performance in two situations. The first situation, during schedule synthesis, has received most attention within existing scheduling systems. The second, much less investigated situation occurs when static world assumptions (SWA) fail. The later is now recognised as being as least as important as the former in both planning and scheduling communities.

A recent article from the planning literature [Sanborn et al '88], introducing a model of reaction for planning in dynamic environments, gives a useful description of SWAs and how they fail in a dynamic domain. The SWAs listed include assumptions that the domain remains static during the period in which a schedule is being generated, that no other agents act on the domain and that the physical world can be completely modelled by the scheduler. A dynamic domain fails to satisfy these assumptions in a number of ways. Since it does change over time, it is unlikely to be completely specified at any point in time and it does change due to the actions of other agents.

Conspicuous by its absence from this list is the assumption that things will go according to plan. Within a manufacturing environment, executional uncertainty is a major problem which must be addressed by any scheduling system designed to be of use at the operational level. Responding to executional uncertainty requires an ability to focus attention dynamically on the appropriate area(s) of a schedule. It is against this backdrop that the following reviews should be considered.

Scheduling as a research area within AI began in earnest with the development of ISIS [Fox M.S. '83], the first scheduling system to address the issues arising out of the complexities of typical real world scheduling domains. It is from this point that this review begins its trace of the development of scheduling in AI.

2.3.1. ISIS [Fox M.S. '83]

ISIS is of significance to the AI scheduling community for several reasons, not least of which being that it moved scheduling research into real world domains. The domain chosen for ISIS was a Westinghouse Electric Corporation Turbine Component Plant (WTCP), the task to generate workable schedules in near real-time for the WTCP job-shop in which there are typically 100 to 200 orders in process at any time, with each order requiring 10 or more operations. The task is complicated by the fact that many of the available resources are shared resources. ISIS views the scheduling problem as a constraint-directed search and as a result, the bulk of its contribution is concentrated on constraint classification and representation. It combines a rich constraint representation with many of the techniques discussed in section 2.2 in order to both prune the search space and guide the order in which it is explored. It is the techniques used to bound and order the search of ISIS which is of relevance to this thesis, rather than the constraint representations developed.

Overview

ISIS performs a hierarchical search in which constraints are utilised to focus problem-solving effort. The complete scheduling task is decomposed over a four tier hierarchy in which constraints

are passed between adjacent levels in a manner similar to the constraint-posting of MOLGEN [Stefik '81]. However, contrary to the practice in MOLGEN, constraints passed to lower levels in the hierarchy are relaxable and serve only to guide the search not restrict it.

As well as passing constraints between levels to direct search, intra-level constraint analysis is performed to improve search efficiency. This takes the form of a three stage process comprising pre-search analysis, search and post-search analysis. The role of pre-search analysis is to bound the search space, while the post-search analysis serves to determine if the results of the search stage are acceptable. The first and third stages are not implemented at all four levels of the hierarchy and, where they are, it is generally in the form of a rule base. The development of both the pre and post search stages to make more intelligent use of the knowledge available is cited as an area for future work.

The four tier hierarchy shown below, evolved out of experience gained with prototype versions of ISIS.

Level 1: Lot Selection
Level 2: Capacity Analysis
Level 3: Resource Analysis
Level 4: Reservation Selection

Fig. 2.1

Initially, ISIS viewed scheduling as a two stage process consisting only of levels 1 and 3. Capacity Analysis, level 2, was considered a necessary addition in order to achieve acceptable solutions in the presence of bottleneck resources. Without this additional level, the Resource Analysis level tended to produce schedules which created bottlenecks near the end of the schedule. This is an example of the horizon effect identified in [Berliner '73]. The final level, Reservation Selection, was added to improve the quality of schedules with respect to the work in process time of lots. Instead of the Resource Analysis level making reservations for resources, it generates time bounds on when the resources should be made available to perform an operation. The Reservation Selection level takes cognisance of these time bounds when making reservations for the resources, while trying to satisfy the objective of reducing the work in process time of the lot being

scheduled.

A detailed description of all four levels is given, followed by a discussion of how effective the mechanisms are at focusing-problem solving effort.

Level 1: Lot Selection

This level of ISIS is responsible for selecting which lots to schedule, and in what order, from the set of known lots. Both the priority class and due date of a lot are used to calculate its priority and hence its position in the sequence indicating the order in which to process selected lots. The priority classification of a lot, for example *forced outage*, *critical replacement*, *shop orders* and *stock orders*, corresponds in some way to the urgency with which it must be produced. As the due date of a lot approaches, the urgency with which it must be produced increases and this is reflected in its priority rating. Lots are prioritised first by priority class and then within each class by nearness to due date. There does not appear to be any pre or post search analysis at this level.

Level 2: Capacity-Based Analysis

As stated earlier, level 2 was introduced to counter a horizon effect [Berliner '73]. Fox states that "Because of the resource analysis level's inability to generate enough alternative states, it was not able to "see" machine bottlenecks near the end of a schedule". Level 2 counters the horizon effect by performing a simple analysis of available capacity and from this, generating temporal constraints bounding the start times of the operations which constitute the lot being scheduled. The temporal bounds passed by level 2 are intended to direct the search at the Resource Analysis level in a manner which inhibits the creation of bottlenecks.

Within this level, the notions of pre-search analysis, search and post-search analysis are very tightly coupled. Pre-search analysis is required to identify all possible starting points for the actual search. The search is a two phase process, the first searching forwards and the second backwards through the operations of a lot. Therefore, pre-search analysis consists of identifying all possible first and last operations.

The search itself is performed in a simple breadth-first manner. The first phase, to identify the earliest start time of each operation in the lot, proceeds forwards through the operations of the lot plan. It starts by assuming the release date given to the lot by level 1 to be the start time of the first operation and proceeds to calculate earliest start times for each operation, taking cognisance of existing reservations. The second phase, to generate latest finish times for each operation, is performed in a similar manner, only this time starting at the end of the lot plan and using the finish time allocated to the lot by level 1 as the starting point for the search.

Post-search analysis consists of passing the generated time bounds for earliest start and latest finish times for each operation to level 3.

Level 3: Resource Analysis

The role of this level is to generate time bound constraints for the machines and other resources required to produce the lot currently being scheduled. It is a detailed search in a three dimensional search space. The three dimensions to be searched concern the sequence in which to perform the operations of the lot, the machine to be used for each operation and the queue position at each machine. There are other possible dimensions which can be searched, such as the shift patterns operated on a machine, but these are considered only in exceptional circumstances.

Pre-search analysis, implemented by means of a rule base, is used to determine the direction of scheduling (forwards or backwards), the need for any additional constraints and the search operators which will be used to generate the search space. As indicated above, only three operators, namely *choose-operation*, *choose-machine* and *choose-que-position* are normally specified. However, if pre-search analysis discovers that a previous attempt at scheduling has failed and a subsequent attempt with a relaxed due date has also failed, a fourth operator, *machine shift*, is added to increase the search space. The pre-search analysis rules are not able to determine what the search space operators should be. For example, there are two types of *choose-que-position* operator available, the *eager-reserver* which reserves the earliest time allowed by the finishing time of the previous operation and the *wait-and-see-reserver* which tentatively reserves a

large block of time and leaves the actual choice of reservation to level 4. The *eager-reserver* will result in increased work in process time for lots in the presence of bottlenecks, whereas the *wait-and-see-reserver* leaves scope for bottlenecks to be alleviated at level 4. The choice of which type of *choose-que-position* operator to use is made manually.

The bulk of search in ISIS takes place within level 3. It is performed using a beam search in a collapsed rated space. The motivation for dealing with a collapsed rather than stratified search space is that none of the potential strata are dominant. Each state in the search space is rated according to the number of relevant constraints it satisfies. After each application of an operator, the resulting states are rated with only the best "n" (typically n =9) states being retained for the next iteration of operator application.

ISIS represents a large number of problem constraints, only some of which are considered relevant at any given point in the search space. The relevant constraints, and hence those which should be fed to the evaluation function, are determined by several features of the constraint representation. Constraints can have a limited period of applicability, specified by the *duration* slot of the constraint in question. Constraint *residency*, the attachment of constraints to objects (eg. resources, orders), also serves to indicate the relevance of a constraint to a given decision. A final "catch-all" facility is provided by the *context* slot of a constraint which contains a lisp function. The function takes the search state and a constraint as input parameters and returns a non nil value if the constraint should be used.

Post-search analysis is again implemented as a rule base. Within the Resource Analysis level, post-search analysis has the task of determining whether a satisfactory solution has been found and, if not, determine whether the system should continue searching or alternatively pursue a different search strategy. An alternative search strategy can be achieved by posting the failed lot for re-scheduling. This time, pre-search analysis would possibly add an extra operator to explore new machine centre shifts.

Level 4: Reservation Selection

This level is responsible for selecting a time reservation for all resources required to produce the lot being scheduled. There is no notion of pre or post search analysis at this level. Time reservations are selected to satisfy the time bounds passed from level 3. Where there is discretion, an attempt is made to reduce the work in process time of the lot.

Comment

ISIS employs various mechanisms to focus problem-solving effort during search, most significantly hierarchical decomposition and search space rating. Both serve to prune and order search during schedule synthesis. Schedule maintenance is not a major feature of ISIS.

As stated previously, there are four levels within the ISIS hierarchy which are used to prune and guide search. Levels 2, 3 and 4 are fairly successful in achieving this goal, an essential feature being the ability to relax constraints imposed by higher levels. However, the constraint ordering imposed by level 1, ie: that all constraints associated with a given lot are considered in isolation of lots which remain to be scheduled, is almost certainly a feature aimed at problem simplification rather than as an effective mechanism for focusing problem-solving effort. Having lot selection as the top level in the hierarchy is not justifiable other than in terms of problem simplification as it does not exhibit complete dominance over levels 2, 3 and 4.

ISIS creates schedules in an incremental manner. Once a decision has been made it is not altered to accommodate subsequent scheduling decisions. Consequently, the order in which decisions are taken implicitly defines their importance. Both the practice of scheduling by lot and in process plan order serve to prioritise decisions in a pre-specified manner. It would be better to allow the current set of problem constraints to guide the decision ordering. This would preclude the need for the time bounds generated by the Capacity-Based Analysis level which are only required because scheduling decisions within a process plan are made in an artificially imposed order ignoring the current set of problem constraints.

Rating the search space by means of relevant constraints and their importance is an essential idea in ISIS as the bulk of search, performed in the Resource Analysis level, uses a beam search technique. The beam search requires a means of evaluating the various points in a search, a facility provided by a traditional evaluation function which accepts satisfied constraints as its parameters. We are not interested in the merits of the beam search technique, but rather the effectiveness of the evaluation function.

The effectiveness of the evaluation function is largely dependent on the constraints it is fed as input parameters. The mechanisms of constraint *duration* and *residency* provide effective static constraint filters. That is, they provide an effective means of eliminating constraints considered to be irrelevant to the evaluation of particular search states. The *context* facility, geared towards identifying constraints whose applicability to a particular decision varies dynamically with the larger problem-solving state, is likely to be less effective. In order to perform its task, the function held within the *context* slot must be given an accurate account of the problem-solving state which takes into account detailed constraint interactions. It is unlikely that all the knowledge required to identify the problem-solving state accurately can be captured in a simple procedural representation, or that it could be passed in a parameterised form to the function held in the *context* slot. It is far more likely that the analysis is performed at an aggregate level which allows simple aggregate heuristics to be used to determine constraint relevance. This will lead either to too many constraints being included in a cautious approach, or relevant constraints being excluded in an optimistic approach.

When static world assumptions fail, as a result of unforeseen opportunities or conflicts, the schedules produced by ISIS move progressively further and further out of step with the real world until the schedule finally fails completely. When this occurs the difference between the real world and schedule is considered sufficient to merit the rescheduling of affected lots. During rescheduling, ISIS attempts to maintain shop stability by treating the now corrupt scheduling decisions as preference constraints when rescheduling.

When the order being rescheduled is of the same priority classification as an order which is already scheduled there is an implicit assumption that orders already scheduled are more important than the one currently being scheduled. This failing to prioritise decisions dynamically is compounded by rescheduling in process plan order. No attempt is made to take cognisance of the cause of schedule failure. If the source of failure was investigated it may lead to a change in the focus of problem-solving effort.

2.3.2. SOJA [LePape '85]

SOJA, a scheduling system run daily for a sheet-iron workshop (Althsom-Atlantique), views the scheduling task as a two stage process. The first stage is to select the orders to be performed the next day and the second, to schedule them to resources. However, unlike in ISIS [Fox M.S. '83], there is no requirement to consider all the constraints relevant to an order in isolation of other orders or in process-plan order. Rather, constraints are dealt with in an order which attempts to satisfy those with the least number of admissible values first.

Overview

SOJA takes as its input the state of the workshop in the evening and a list of the operations waiting to be performed. It produces as output a schedule which covers each resource in the workshop for the following day. It does this by first selecting the operations to be scheduled and then scheduling them. The interest in selecting a particular operation depends strongly on the state of the workshop, the other operations to be selected from and those operations already selected. To capture these dependencies, SOJA builds a *selection graph* which it then uses to select the operations to be scheduled. Constraints are represented as disjunctions of facts. Each member of the disjunction represents a choice which may be selected to satisfy the constraint. The scheduling task is viewed as a constraint-directed search in which each scheduling decision consists of choosing one fact from a disjunction of facts. Heuristic scheduling rules and constraint propagation techniques are utilised to enhance the search performance.

Order Selection

As stated above, operations are selected for scheduling by constructing a selection graph *G(X,U)* and using it to select operations in a way which takes cognisance of dependencies between operations. In the selection graph, *X* is a set of nodes which contains a root *R* and particular operations that have been pre-selected, where pre-selection involves selecting only those operations which are ready to be selected. The set of value tagged arcs is denoted by *U* in which arcs are of the form *(op1 op2 val tag)* meaning that if *op1* is selected, then the selection of *op2* is of interest with respect to criterion *tag* with *val* quantifying the degree of interest.

The selection graph is constructed using weighted heuristic production rules of the form "IF conditions THEN pre-selections". Conditions refer to the state of the workshop and the available operations. Pre-selections consist of adding new nodes and arcs to the graph. An example is shown below:

```
Rule 1 (Weight 17)
IF op must be performed before the end of the week
THEN add op and an arc (R op 17 due date) to the graph
```

Figure 2.2

The graph is built up by iteratively applying the rule having the highest weight at each iteration. The graph, initially containing only node *R*, is allowed to grow until a constraint imposed by the capacity of the workshop is violated.

Operations are then selected from the graph until there are no "under-loaded" machines in the workshop. The algorithm used to perform this task is shown below:

```
WHILE (there are under-loaded machines)
      (Select an arc R -> Op)
      IF (Op can be selected without overloading its machine)
      THEN (Select it)
      (Record the tag of the arc)
      (Replace each arc Op -> Op' by an arc R -> Op')
      (Take Op out of the selection graph)
```

Figure 2.3

Constraint-Directed Scheduling

Constraint-Directed scheduling in SOJA is the process of sequencing operations through machine centres, utilising the constraints of the problem to guide the search to a solution which is both legal and "good". To do this, SOJA partitions constraints into two classes, hard constraints and preference constraints. Hard constraints are necessary to define the set of legal solutions, whereas preference constraints are used to define a subset of "good" solutions.

Hard constraints such as operation precedence and resource requirements are represented as a disjunction of facts and a scheduling decision consists of choosing a fact from the legal possibilities defined within the constraint. When a decision is made, SOJA propagates the effects of the decision to other constraints. Each hard constraint can be in any one of four states during the scheduling process. If it is *satisfied* then one of the facts of the constraint is implied by previous decisions. If it is *impossible* then each fact belonging to the disjunction is banned. If it is *imperative* then only one fact belonging to the disjunction is not banned. Finally, if it is *disjunctive* then two or more facts can be chosen to satisfy the constraint.

Preference constraints, concerning objectives such as due date requirements and shop stability, are represented as heuristic scheduling rules. These rules have been extracted from problem domain experts and the field of operational research. This is in contrast to ISIS in which preference constraints are represented as relaxable constraints.

Schedules are constructed incrementally, satisfying one constraint at a time until all constraints are satisfied, at which point a solution is inferred. The scheduling algorithm used embodies the principle that problem-solving effort should be focused on areas of high contention in the schedule. This is achieved by algorithmically encoding the following three rules:

(1) When a constraint becomes *impossible*, a selective backtrack procedure is called which selects a decision to be retracted.

(2) When a constraint becomes *imperative*, the only remaining fact is immediately asserted and propagated.

(3) When there are no *impossible* or *imperative* constraints, a fact is chosen, guided by the heuristic scheduling rules, to satisfy a *disjunctive* constraint.

Comment

SOJA offers certain improvements over ISIS in terms of mechanisms for focusing problem-solving effort. This is most obvious during the selection phase in which SOJA considers both inter-lot and intra-lot dependencies. In ISIS, after a lot is selected for scheduling it is scheduled in detail before subsequent lots are selected, therefore precluding consideration of lots still to be selected. The approach used in SOJA will lead to a more effective search order because it groups constraints relevant to the selection problem together allowing consideration of all lots which may be selected.

Within the actual scheduling process itself, SOJA employs another search ordering technique. The scheduling algorithm is geared to address the most constraining constraints first. A constraint (a disjunction of facts) is ranked according to the number of facts from its disjunction which are not banned. This is again more responsive to the characteristics of the given scheduling problem than in ISIS, where search order through a plan is pre-determined.

As well as ordering the search space to focus problem-solving effort, SOJA incorporates mechanisms which prune the search space. The two stage hierarchy comprising *order selection* followed by *constraint-directed scheduling* serves to prune a considerable area of the search space. Although this is a useful technique, in more complex problems such as that considered in ISIS, the fact that order selection does not exhibit complete dominance over the scheduling level can lead to problems. The use of the *selection graph* does however help to minimise the occurrence of such problems. The search space is further pruned by a constraint propagation technique. This serves to remove nodes from the search-space which are inconsistent with the scheduling decisions made

so far.

Although the features noted above represent tangible improvements as far as focusing problem-solving effort is concerned by ordering the search in a way which is responsive to the problem at hand, it should be noted that SOJA deals with a considerably simpler problem than ISIS. It would appear from the selection algorithm that operations are pre-allocated to specific machines, thus precluding the need for such decisions and a significant degree of combinatorial complexity. Furthermore, as scheduling in SOJA is synonymous with sequencing, this seems to imply that an earliest dispatch strategy is always used. However, the most serious weakness of SOJA is one shared by ISIS. Neither SOJA or ISIS provide a reactive capability.

From the literature available on SOJA, two questions about how it functions remain unanswered. Firstly, what generates the heuristic production rules used to construct the selection graph? Secondly, what evaluation functions are used to select an arc from the selection graph?

2.3.3. OPIS 0 [Smith et al '86b]

OPIS 0 is of interest primarily because it was the first system to solve its problem using multiple scheduling perspectives. The scheduling perspective of a system is determined by the manner in which it decomposes the problem. For example, ISIS is said to use an order-based perspective because it views the complete schedule as a collection of order schedules. An alternative possibility is a resource-based perspective, in which the complete schedule is viewed as a collection of resource schedules. Experience gained during the development of ISIS led to the recognition that multiple scheduling perspectives were needed. Some members of the ISIS team went on to confirm their intuitions by developing OPIS 0.

Each scheduling perspective advocates a specific local and incomplete view of the overall scheduling problem in terms of more tractable subproblems. A consequence of this is that a given scheduling perspective is more or less suited to resolving certain classes of constraint conflict. An order-based perspective is well suited to dealing with order-centred constraint conflicts which might

involve the operation precedence constraints of an order and its due date. On the other hand, a resource-based perspective is well suited to dealing with resource-centered constraint conflicts which arise from the need to share resources and involve constraints associated with several orders. The objective of the designers of OPIS 0 was to provide experimental justification to support the claim that it is beneficial to apply both scheduling perspectives as and when appropriate.

Overview

In order to gain any benefit from using multiple scheduling perspectives, it is necessary to partition effort between perspectives such that the most important constraint conflicts can be addressed by the appropriate perspective. A reasonable heuristic, adopted by OPIS 0, suggests that the most important resource-centred conflicts are likely to occur at the bottleneck resources. Therefore a resource scheduler should be applied to bottleneck resources after which an order scheduler can be applied to complete the schedules of each individual order. Because OPIS 0 was developed to provide experimental justification for the value of multiple-perspective scheduling, it was considered acceptable to place some rather severe limitations on system flexibility. The most obvious limitations are that only one pre-specified bottleneck resource is catered for and that effort between the two scheduling perspectives is partitioned statically. As suggested by the heuristic, a resource scheduler is first applied to the single pre-specified bottleneck resource, after which an order scheduler is invoked to work outward from this established portion of the shop schedule to complete the schedules of each individual order.

Resource Scheduler

The resource scheduler of OPIS 0 takes as its input, a set of machines (a work area) and a set of operations to be scheduled on those machines. Each operation belongs to a different order and typically must be scheduled on one of a specific group of substitutable machines in the work area.

The scheduling method used is basically one of iterative selection which terminates when there are no more operations to be scheduled. On each iteration, a set of scheduling decisions is

made, serving to constrain the way in which the remaining operations can be scheduled. Within each iteration there are four distinct phases, the first three of which involve the selection of an element from a set. The selected element in each phase is best with respect to some criteria expressed by heuristic rules. The final phase, resource reservation, is not viewed as a selection process. This will be discussed later. The four phases of each iteration are shown below:

(1) Resource Assignment
(2) Resource Selection
(3) Resource Dispatching
(4) Resource Reservation

In the Resource Assignment phase, the best machine assignment for all remaining operations is chosen from the set of alternative machine assignments. Two criteria, earliest start time and the number of machine setups, are used to determine the best machine assignment. The Resource Selection phase reviews all machines that have one or more assignments from the previous phase. It does this in order to remove machines which are considered unsuitable for scheduling on this iteration. Machines are considered unsuitable if it is feared that if they were scheduled a premature scheduling decision leading to a poor schedule may result. The factors considered when selecting machines to exclude from a particular iteration are machine availability, characteristics of the resource assignment and the length of queue of operations already assigned to a machine. The role of Resource Dispatching is to select one operation for each of the machines which have had operations assigned to them and have not been excluded in the previous phase. The dominant criteria used for selection in this phase is tardiness. The final phase, Resource Reservation, reserves a time interval for each dispatched operation. This results in new temporal constraints for operations which have still to be scheduled.

Order Scheduler

The order scheduler (OS) of OPIS 0 operates in a manner similar to ISIS [Fox M.S. '83]. It has three identifiable phases which can be recognised as the bottom two levels of ISIS, Resource Analysis and Reservation selection. In OPIS 0, the OS is invoked to schedule a specific portion of

the production plan of an order, either the operations preceding or following the bottleneck resource. It selects resources and makes temporal reservations on these resources for the appropriate operations. The three identifiable phases, Search Initialisation, Beam Search and Temporal Reservations are discussed below.

The Search Initialisation phase is responsible for selecting both search direction, forward or backward, and the time anchor, either earliest start time or latest end time. This corresponds to the pre-search phase of Resource Analysis in ISIS. The previously scheduled "bottleneck operation" of the order is utilised to determine both search direction and time anchor. If the operations to be scheduled precede the bottleneck, the OS schedules backwards from the start time of the "bottleneck operation". Conversely, if the operations to be scheduled follow the bottleneck, the OS schedules forwards from the end time of the "bottleneck operation".

The search space traversed by the OS during the beam search phase, corresponding to the search and post-search phase of Resource Analysis in ISIS, is composed of states which represent alternative sets of schedule decisions. The degree of satisfaction of the relevant preference constraints (eg. work-in-process constraints and sequencing preferences) is evaluated to provide the basis for state comparison. The output of the search is a specific routing for the given operations along with an associated interval of time for the chosen resources.

The final phase, corresponding to Reservation selection in ISIS, involves making actual resource reservations within the time bounds provided by the beam search. The time bounds provided are typically larger than the actual time needed to perform the operation in order to allow for local optimisation. Operating within the imposed time bounds, final allocations are made that attempt to minimise the work-in-process time of the order.

Comment

The limitations imposed on OPIS 0 were considerable. The provision for only one pre-specified bottleneck resource, the domination of the resource-based perspective and static partitioning of scheduling effort combine to greatly limit system flexibility. Despite these

limitations, OPIS 0 performed better than both ISIS and COVERT in laboratory tests [Smith et al '86a], thus providing experimental weight to the claims for multi-perspective scheduling.

In OPIS 0, the search order is arranged in a way which considers first high contention decisions (around the bottleneck resource), followed by initially less constrained decisions. This has obvious parallels with Haralick's fail first principle [Haralick et al '80]. By imposing the decisions of the resource scheduler on the order scheduler, the search space can be pruned in an effective manner. In more complex domains, sequencing and hence prioritising decision making in such an aggregate way is almost certain to lead to problems.

The final phase of the resource scheduler, resource reservation, is not regarded as a selection process. This is because it is assumed that the earliest possible reservation is always made. The justification for this is that if the resource scheduler is being applied, then the resource in question is in high contention and therefore there is no need to consider introducing slack time between operations. In view of the complete dominance of the resource-based perspective, this is a potentially dangerous heuristic to use.

2.3.4. OPIS [Smith '87]

As stated in the previous section, OPIS 0 accepted some rather severe limitations on system flexibility. OPIS, a direct descendant of OPIS 0, began life as an extension to OPIS 0, motivated by a desire to take the multi-perspective scheduling strategy into more realistic domains. Realistic scheduling domains include things like time-varying bottlenecks and primary/secondary bottleneck configurations, significantly different from the idealised view assumed in OPIS 0. To facilitate this transition, it was recognised that OPIS must be able to decompose the scheduling problem dynamically in accordance with the current set of problem constraints. To do this OPIS requires the ability to detect areas of the schedule which are currently perceived to be in high contention. In OPIS 0 this was not necessary as the area of high contention had been pre-determined as the pre-specified bottleneck resource.

Giving OPIS the discretion to decompose the problem dynamically presents new difficulties. Because of the static nature of OPIS 0, it was possible to perform a crude form of forward scheduling to ensure that solutions generated for subproblems would be compatible. By removing the OPIS 0 limitations, it is no longer possible to simply allow earlier decisions to dominate subsequent decisions. Within OPIS, the integration of solution components requires the ability to renegotiate specific decisions in the light of subsequent decisions.

The two features of OPIS not present in OPIS 0, namely the ability to recognise important areas of the schedule and to renegotiate earlier scheduling decisions in the light of subsequent events, enabled it to achieve its goal and move into more realistic domains. However, these facilities are also precisely what is required to allow reactive maintenance of schedules. OPIS grew into a system which viewed both predictive and reactive scheduling as an opportunistic process. It merges top down dynamic problem decomposition with bottom up reaction to problems encountered during the synthesis of solution components, or unanticipated events which occur in the production environment. It is this unified view of both predictive and reactive scheduling as an opportunistic process which makes OPIS of interest.

Overview

The OPIS architecture is essentially that of a standard blackboard system [Erman et al '80]. It is motivated by a desire to focus scheduling effort dynamically according to current problem constraints and presumes the existence of a collection of knowledge sources (KS) that can be selectively employed to generate, revise and analyse specific components of the overall schedule. Unlike standard blackboard systems, in which KS are self activating, OPIS has a search manager KS responsible for planning and co-ordinating the scheduling actions to be taken in response to a given scheduling problem. Scheduling proceeds via the formulation and initiation of scheduling tasks which specify a particular scheduling KS and a portion of the current factory schedule to be worked on. The search manager KS maintains a queue of pending subtasks, which constitutes its current plan for solving the scheduling problem at hand. Changes in the factory status and the

execution of scheduling tasks are integrated into the current schedule by a schedule maintenance subsystem. The search manager KS is informed of any changes through the posting of control events which require to be interpreted and converted into modifications to the queue of pending subtasks if appropriate. Thus the manager implements a reactive approach to control, continually revising its "scheduling plan" as the constraint set changes.

Representing and Maintaining the Schedule

In order to permit the opportunistic approach to scheduling advocated by OPIS, it is necessary to maintain a representation of the current schedule and hence the current problem constraints. The schedule is represented in terms of an underlying factory model which includes resources and the operations to be performed. This representation is implemented using frame-based knowledge representation techniques and provides a hierarchical description of the constraints of the production environment. Hard constraints, such as operation precedence, duration and required resources are catered for, as are preference constraints which serve to characterise factory objectives such as minimising WIP or tardiness. This representation allows reasoning over the set of possible schedules rather than prematurely committing to a single schedule, the opportunistic approach proposed in [Fox B.R. et al '85a].

Given this factory model, the schedule is made explicit by instantiating an appropriate production plan for each order to be produced. The current state of the schedule is provided by incrementally maintaining a representation of the temporal constraints on each instantiated operation and the availability of each required resource. These representations are maintained by a set of propagation processes that combine newly imposed constraints with both model defined constraints and those resulting from previous decisions to derive any additional constraints.

As well as making the current set of problem constraints available to scheduling KSs during schedule synthesis, representing the current state of the schedule also provides the basis for detecting conflicts in the schedule. OPIS recognises three basic constraint conflict types. *Time conflicts* in which operation precedence constraints have been violated, *Capacity conflicts* in which

resource availability constraints have been violated and *Time Vs Capacity conflicts* which corresponds to the situation in which a given set of scheduling decisions cannot coexist due to the combined effect of temporal and resource availability constraints. Conflicts are introduced into the schedule by changes that impose additional constraints via the propagation process.

Strategic Alternatives

The strategic alternatives available to OPIS include two general scheduling methods, two schedule revision methods and a single analysis method. The two general scheduling methods, an order scheduler and a resource scheduler, are not discussed here as they have already been covered in the section on OPIS 0. The two schedule revision methods are implemented by a schedule shifter KS and a demand swapper KS.

The schedule shifter implements a reactive method which simply moves the scheduled execution times of designated operations forward in time by a specified amount. Any inconsistencies resulting from these operation shifts are immediately resolved by additional shifting.

Demand swapping is a specialised reactive method applicable in situations where an operation has become unexpectedly and significantly delayed. It implements an exchange of the remaining portion of the schedule of the affected order with the corresponding portion of the schedule of another order of the same type. Heuristic criteria that estimate the relative flexibility of current temporal constraints are used to identify potential swapping candidates and, if at least one exists, select a particular candidate.

The analysis method available in OPIS is implemented as a Capacity Analyser KS. It provides information relating to the current factory load, identifying likely areas of high resource contention. It always operates at an aggregate level in the factory model, constructing a schedule which satisfies the current temporal constraints, using a line balancing heuristic in situations where a choice between resources is necessary. The demand for capacity reflected by this schedule is then compared with the actual capacity available to identify likely bottleneck areas.

Co-ordinating Scheduling Activity

Co-ordination of scheduling activity is based on response to control events which are posted as a result of both externally initiated scheduling updates and internally initiated scheduling actions. Event descriptions provide the search manager KS with an abstract view of the current state of the schedule. The search manager KS draws on two types of control heuristics when responding to posted events on a given control cycle. It first applies a collection of *event aggregation* heuristics followed by *event processing* heuristics.

Event aggregation heuristics are employed to formulate the most appropriate set of problems to address. In many cases, there are relationships between events that suggest they should be considered simultaneously rather than individually. Event aggregation heuristics define the circumstances under which two or more events should be reformulated as an aggregate event.

Event processing heuristics are applied to determine how the system should proceed. It is these rules that encode knowledge of the applicability and capabilities of the various strategic alternatives available to the system. Once all events have been processed, the queue of pending subtasks is updated and the highest priority pending subtask is initiated. Subtask prioritisation is a function of the triggering event type and the characteristics of the event itself.

Comment

OPIS certainly achieved its goal of taking the multiple scheduling perspective methodology into more realistic domains. Initially, the objective was to attain opportunistic problem decomposition and through this deal with more complex domains. However, as OPIS developed another form of opportunism emerged, one which is almost certainly required in real world domains. By adopting the opportunistic approach advocated in [Fox B.R. et al '85a], OPIS was able to provide a reactive capability, necessary in the face of unexpected events. Reacting to unforeseen events is not always a fire fighting situation. The unforeseen event may generate an opportunity as in the case of a machine being repaired sooner than expected.

OPIS provided significant advances in methods for focusing problem-solving effort, most notably through opportunistic problem decomposition and the detection of high contention areas of the schedule. To do this it distributes problem-solving knowledge amongst the various KSs, but it does not address the issue of distributed processing.

Within OPIS, the tasks of predictive and reactive scheduling remain distinct. Event aggregation and event processing heuristics are employed to determine which course of action to follow at any given point in the search. This thesis argues that the applicability of the heuristics used is likely to be domain dependent. Regardless of the loss of generality incurred by the use of domain dependent heuristics, it is further argued that identical mechanisms should be employed for both predictive and reactive scheduling. Separating the tasks requires two things. Firstly, an ability to determine the type of reaction required in a particular conflict situation, and secondly a reactive mechanism to resolve the conflict. It is considered unlikely that the full complexity of all possible conflicts can be captured within a set of heuristics. The prospect of having a reactive mechanism ideally suited to each conflict situation is considered equally unlikely.

2.3.5. SONIA [Collinot et al '88]

SONIA provides another example of the trend towards scheduling systems which take a unified view of predictive and reactive scheduling. It builds on experience gained with SOJA, one of the predictive scheduling systems discussed earlier. The designers of SONIA cite the fact that both predictive and reactive components appear to refer to common pieces of knowledge as the prime motivation for integration.

Overview

Like OPIS, SONIA utilises a blackboard architecture to manage the focus of problem-solving effort. It has a predictive component, two reactive components and two analysis components. As in OPIS, the reactive capability makes it necessary to maintain the current set of problem constraints. The predictive component of SONIA is provided by SOJA and will therefore not be

discussed below.

Management of Schedule Descriptions

Within SONIA, a schedule is represented as a set of resources, manufacturing orders and operations to which constraints are attached. Resources are described at various levels of aggregation. The production plan of an order consists of a hierarchy of operations within which it is possible to specify that operations can be performed in parallel or that they are a set of exclusive alternatives. Each operation has an associated actual status and a schedule status. Relational temporal constraints of the form "event-1 must precede event-2 by period X" are generated in accordance with the status information. When the actual and schedule status become different it signifies the need to repair the schedule. This representation serves to describe implicitly the set of schedules which are consistent with the current set of problem constraints.

As in OPIS, maintaining the current set of problem constraints serves two purposes. It enables problem decomposition without neglecting interactions between subproblems and it enables detection of inconsistencies. Constraint propagation techniques are used to maintain the current set of problem constraints. The constraint propagation system system employed in SONIA is interesting in that it is flexible in the degree of propagation that is performed. This is discussed further in chapter 4.

Strategic Alternatives

SONIA provides three alternatives to resolving inconsistencies arising from ill-considered decisions or shop-floor deviations. In extreme cases it may be considered appropriate to invoke the predictive component to generate a whole new schedule. This can be viewed as a form of reaction. The reactive components which provide the remaining two options can take decisions to reject operations, select new operations to be scheduled, slightly extend work shifts, relax due date constraints and permute operations.

A less drastic form of reaction than producing a completely new schedule involves rescheduling forward from the current date, modifying the existing schedule rather than creating one from scratch. Two versions of such a global rescheduling algorithm exist. The first corresponds to a simple right shifting strategy in which the operations to be performed remain in the same sequence and the schedule is moved forward from the current date. The second attempts to permute operations in order to remove the conflicts.

Although the global rescheduler can be used in any circumstances, local methods are more appropriate when conflicts concern a particular order or work area. At the moment, the only form of local rescheduling is provided by an algorithm which deals with part of the production plan of an order. A resource scheduler is cited as the most likely form of reactive method to be introduced next.

To facilitate appropriate use of the reactive methods available to SONIA, analysis components capable of identifying the problem-solving context are needed. SONIA has two such components, a *capacity analyser* and an *analyser of conflicts*. The capacity analyser of SONIA was "inspired" by the shop-level capacity analyser of OPIS (already discussed) and shall therefore not be described here.

The analyser of conflicts is used to examine a set of conflicts and generate proposals to solve some or all of the detected conflicts. It may decide that it is appropriate to consider certain related conflicts simultaneously, performing much the same role as the event aggregation heuristics of OPIS. The proposed solutions are currently confined to specifying which reactive component to use and in what manner to use.

Co-ordinating Scheduling Activity

As stated earlier, SONIA employs a blackboard architecture to co-ordinate problem-solving activity. It actually uses two blackboards, a domain blackboard and a control blackboard. The domain blackboard is partitioned into three areas, *results*, *capacity* and *conflicts*, which relate to the evolution of the schedule. Results of the activities of the predictive and reactive components are

held in the results partition, the results of the capacity analyser (ie: bottlenecks, under-load) in the capacity partition and any conflicts which are detected are recorded in the conflicts partition of the domain blackboard.

The control blackboard is used to record information relating to the focusing of problem-solving effort. Several partitions exist including an area for *sub-problems*, an area for *strategies* made up of heuristics used by the various components of SONIA, an area for the *agenda* of pending actions, an area to hold the *policies* used to select amongst pending actions and finally, an area *chosen actions*, a history of actions already performed. As in OPIS, there is a control KS responsible for selecting the next KS activation. This KS refers to information in the *policies* partition of the control blackboard when making its decision. There is also another control KS responsible for updating the agenda partition of the control blackboard.

Comment

SONIA is very close in both architecture and functionality to OPIS. This is probably a result of the fact that Claude LePape worked on the development of both systems. While sharing the strengths of the opportunistic approach of OPIS by reasoning with a set of schedules, SONIA does not provide a resource scheduling component to permit an alternative scheduling perspective and hence allow opportunistic problem decomposition.

As in OPIS, the basic approach to reaction is to analyse, via a set of heuristics, the current set of conflicts in order to determine the most appropriate form of schedule reaction. In both OPIS and SONIA, there is an attempt to patch the existing schedule using either a right shifter or a demand swapper algorithm, rather than to reschedule the affected parts of the schedule. The criticisms of this approach made in the comment section on OPIS apply equally to SONIA.

2.3.6. S2 [Elleby et al '88]

S2 is a knowledge-based scheduling system designed to perform operational level scheduling in VLSI wafer fabrication domains. It views scheduling as an incremental constraint satisfaction

problem, and like OPIS and SONIA, takes a unified view of predictive and reactive scheduling. As well as being reactive, S2 is also *adaptive*, ie: it keeps the human scheduler in the schedule generation loop to allow criticism of the proposed schedule to be incorporated.

Overview

The domain in which S2 must operate is typical of many manufacturing domains in that it does not conform to the static world view of scheduling. Two particular difficulties have had a significant effect on the design of S2. Firstly, due to the conflicting and dynamic nature of scheduling objectives, it is difficult to define the criteria by which a schedule can be considered optimal. Secondly, the domain in which S2 must operate is itself highly dynamic with unexpected events such as machine breakdowns occurring at frequent intervals.

In response to the first problem, conflicting and changing objectives, S2 was designed to be adaptive. In an adaptive scheduling system, the automated scheduler initially suggests a feasible schedule which a human scheduler then judges. These judgements are communicated to the automated scheduler which is expected to propose a more suitable schedule incorporating the human criticisms. As well as producing improved schedules, this approach can have a beneficial side-affect in that the human scheduler can learn from the suggested schedules about the relationships between various scheduling objectives.

An automated scheduling system can only take account of information it has been given initially. Unfortunately, acquiring all the relevant information for a scheduling system is not a simple task. Typically, information concerning hard constraints which define the set of legal schedules is more readily available than information concerning soft constraints. Soft constraints, such as scheduling objectives, define the subset of legal schedules which are considered desirable. By being adaptive, S2 allows the human scheduler to intervene during schedule synthesis and add additional soft constraints to the scheduling problem. This alleviates some of the burden from the very difficult knowledge acquisition phase of the development of a knowledge-based scheduling system.

To cater for the dynamic nature of the wafer fabrication environment, S2 adopts the least commitment approach proposed in [Fox B.R. et al '85c]. The proposed approach advocates reasoning with the set of schedules which satisfy all existing commitments, rather than prematurely committing to a single schedule. This allows for both opportunism, the exploitation of a resource becoming available, and contingency, the reduction of the consequences of a resource becoming unavailable.

Architecture

S2 has three major components, a *constraint maintenance system* , a *reactive schedule generator* and a *request interpreter*. The set of constraints defining the class of feasible schedules is stored in the constraint maintenance system. It is through this representation of the class of feasible schedules that the least commitment approach described above is implemented. The constraint maintenance system is able to reason with constraints representing the duration requirements of the fabrication process and precedence relationships between resources. This is discussed further in [Elleby '87].

The reactive schedule generator ensures that the set of constraints recorded in the constraint maintenance system are satisfied during schedule synthesis. It can be considered reactive in that it reacts to new constraints by modifying an existing solution, rather than by scheduling from scratch. In the prototype version, the schedule generator consists of a rule base of various dispatch procedures. Later versions of S2 employ a schedule generator which performs constraint satisfaction problems incrementally with the aid of a tailored form of an assumption based truth maintenance system [de Kleer '86].

The request interpreter provides a means of communicating additional constraints to the constraint maintenance system. Additional constraints can originate from two sources. They can either be the result of criticisms made by the human scheduler or events occurring on the shop floor. S2 operates in conjunction with a work-in-progress tracking system which provides the input to the request interpreter corresponding to events occurring on the shop-floor.

Comment

S2 addresses a major difficulty experienced by most knowledge-based system designers, the knowledge acquisition phase, by being adaptive. This, combined with the highly dynamic nature of its problem domain, dictates that S2 must be a truly reactive system. Conflict analysis heuristics used in conjunction with schedule revision algorithms would quickly become an inadequate form of reaction in such an environment. The designers of S2 view the scheduling problem as one of incremental constraint satisfaction and make use of an assumption-based truth maintenance system (ATMS) to allow a truly reactive approach. S2 does not distinguish between the tasks of schedule generation and maintenance and employs the same mechanisms for both.

A similar use of ATMS technology can be seen in FLYPAST [Mott et al '88], a system that performs the allocation of flight crews and aircraft to pre-scheduled flights for Naval flying programmes. FLYPAST is not reviewed in this thesis because it does not actually perform a scheduling task and its domain is significantly far removed from manufacturing. However, its domain does share a need for a reactive capability and it is interesting to note a common approach through the use of an ATMS.

2.3.7. ENTERPRISE [Malone et al '83]

Until now, this review has been concerned only with systems which take a centralised approach to scheduling. At this point, decentralised scheduling systems are considered, hence the diversion from chronological ordering. The first decentralised system considered is ENTERPRISE, a system which provides a facility for sharing tasks among available processors on a network of personal computers. Although the scheduling problem dealt with by Enterprise is not from a manufacturing domain and is significantly simpler than that found in most manufacturing domains, it is worthy of mention because it is one of the earliest systems to consider the use of decentralised scheduling techniques.

Overview

In most computer networks, nodes are dedicated to a single user. Whenever a node is not being utilised by its user, the norm is for that node and its available processing power to lie idle. The purpose of Enterprise is to harness the maximum processing power available on a computer network by distributing tasks among available nodes. Therefore, the basic function required of Enterprise is to allocate computational tasks to the processor best suited to performing it at the time it is required. This scheduling problem is considerably simpler than most manufacturing scheduling problems because the tasks to be scheduled are largely independent of one and other and the criteria for selecting a particular processor is well defined. The criteria used is a function of the estimated task completion time, a value which is readily available and provides a uniform measure with which to compare all potential processors, and task priorities. Enterprise is not concerned with the particular global scheduling objective which dictates the allocation of priorities to the tasks to be scheduled, but rather with the mechanism which sequences tasks according to the allocated priorities. By varying the method of allocating priority to tasks, Enterprise can be made to achieve different global objectives.

Motivation For Decentralisation

A decentralised scheduling technique was adopted primarily because it was recognised that in highly parallel systems much of the information used in scheduling is inherently distributed and rapidly changing, for example, instantaneous system load. In such an environment it is sensible to "take the decisions to the information" rather than to transmit the information to a centralised decision maker. Another benefit of decentralised scheduling is the enhanced reliability of a system which degrades gradually rather than suffering total failure in the event of one or more scheduling nodes failing.

Decentralised scheduling in Enterprise is based on the metaphor of a market in which processors (clients) send out *requests for bids* on tasks to be performed, and other processors (contractors) respond with bids giving estimated completion times. The estimated completion

times which are returned reflect machine speed and the files currently loaded on the contractor processor making the bid. This protocol has much in common with the contract net metaphor [Davis et al '83] discussed in the next section, the most important difference being that the protocol used in Enterprise restricts the criteria for mutual selection by clients and contractors to two primary dimensions, task priorities and estimated completion times.

Scheduling Strategies

Simulations of Enterprise were performed to compare three distributed scheduling strategies, *lazy* assignment, *eager* assignment and *random* assignment. In the lazy assignment strategy, idle contractors respond with bids immediately, while busy contractors acknowledge the request for bids and add the task to their prioritised queues. Whenever a contractor becomes idle, it submits a bid for the next task on its queue. Clients defer assigning a task to a specific contractor until the contractor is actually ready to start. It is this last feature, the deferring of decision-making, which earns the strategy its name.

In the eager assignment strategy, tasks are assigned to contractors as soon as possible and then reassigned as necessary. All contractors bid on all tasks even if they are currently busy. Contractors may subsequently notify clients of changes to their bid if tasks are added to their queues or if a task takes longer than expected.

In the random assignment strategy, clients select the first contractor to respond to their request for bids and contractors pick the first task they receive after an idle period. Contractors do not bid at all when they are executing a task, and answer all requests for bids when idle. Whenever a contractor receives a task after commencing execution of an earlier task, the new task is rejected and the client who submitted it must attempt to schedule it elsewhere.

Based on the similarities of the scheduling task and job-shop scheduling, the designers of Enterprise expected the *eager* assignment method to perform best. Surprisingly, it performed very poorly in the simulation tests and two factors were suggested. The first, termed the *stable world illusion*, occurs because tasks are assigned to machines on the assumption that no further tasks will

arrive. If tasks of a higher priority do arrive later, they will displace the first task to later and later start times with the effect that the displaced task might well have completed earlier on another processor. The second factor, *unexpected availability*, occurs when a task takes less time than expected. This can result in fast processors lying idle while high priority jobs wait in queues on slower processors. This occurs because clients are not notified of the fact that a machine has become available early, and therefore cannot reschedule to take advantage of the opportunity. The authors of [Malone et al '83] conclude that the *lazy* assignment method, despite being simpler to implement and less expensive in terms of message traffic, is a superior strategy.

Comment

Although distributed computing and heavy manufacturing environments appear to have little in common, they do share some common motivations for adopting a decentralised approach to their respective scheduling problems. The concept of "bringing the decisions to the information" rather than constantly transmitting information to a centralised decision maker has obvious relevance to manufacturing environments which are often spread over large areas. This is particularly true in heavy manufacturing industries in which there are typically several distinct manufacturing units on any given site. The benefits of enhanced reliability derived from a distributed system will also be welcome in a manufacturing environment.

The factors which resulted in the lazy assignment strategy performing better than the eager assignment strategy have parallels in manufacturing domains. The *stable world illusion* can fail due to the arrival of a rush order or indeed the addition of any order into the system. A job finishing earlier than expected or alternatively, a machine being repaired sooner than expected both give rise to the *unexpected availability* factor. This appears to suggest that the lazy assignment approach is suitable for manufacturing domains, though the domains may be sufficiently different to invalidate this conclusion. In a manufacturing environment the tasks to be scheduled tend not to be independent, since they are often related through precedence relations and share common resources. When sharing resources, it is often desirable to batch or sequence tasks in a way which

reduces machine setup costs or minimises scrap wastage. In such an environment, the buffer provided by a lazy assignment strategy is likely to be ineffective. What is really required is a strategy which considers the global solution while providing the ability to react *opportunistically* to events which alter the assumptions upon which it is based.

2.3.8. YAMS [Parunak et al '86]

The scheduling systems discussed which utilise blackboard style control architectures, OPIS and SONIA, are examples of distributed problem-solvers. In such systems, problem-solving knowledge is distributed throughout specialised knowledge sources which may be invoked, one at a time, to contribute to the current solution hypothesis. Blackboard control architectures address the issue of distributed problem-solving, not distributed processing.

It is possible to distinguish between systems which distribute only problem-solving knowledge and those which also distribute processing by noticing that in the former case, knowledge sources contribute to the current hypothesis in a synchronised manner, whereas, in the latter, several knowledge sources can contribute simultaneously in an asynchronous manner. The ENTERPRISE system reviewed in the previous section is an example of an asynchronous scheduling system. YAMS, a scheduling system for controlling a flexible manufacturing facility, provides probably the best known example of an asynchronous scheduling system. It is described below.

Overview

YAMS (Yet Another Manufacturing System) is a cooperative scheduling system used to control a flexible manufacturing facility. The negotiation protocol used during schedule synthesis is based on the contract net metaphor [Davis et al '83]. Initially, a global schedule is produced using a technology such as MRP or OPT. This schedule assigns fairly broad time windows for the completion of operations for all work orders at a high level of abstraction. This time window is refined through the negotiation of team schedulers and their subordinates who act on behalf of

individual work-cells. A team scheduler broadcasts a message to its subordinates signaling that an operation is to be scheduled. Those subordinates that are able and willing to perform the operation compete for it by returning bids, giving an indication of when they can perform it. The team scheduler selects from these bids to schedule the operation.

YAMS models the factory environment as a hierarchy of work-cells, corresponding closely to the traditional view of a manufacturing organisation. Each node in the hierarchy corresponds to a node in the contract which can negotiate with its parent, children and sibling nodes. Each node has a library of process plans describing the processes which it knows how to perform. These process plans refer in turn to other processes, some of which are not known to the node hosting the main process. Nodes use negotiation to find other nodes to perform processes which they cannot deal with themselves.

Motivation For Decentralisation

The motivation to use distributed AI techniques in general, and the contract net in particular, is as a result of several characteristics of the typical manufacturing domain. A distributed system is able to control not just a single localised set of resources, but several aggregate resources which may be remote from one another. It is not unusual in a manufacturing domain for each set of aggregate resources to require to be controlled faster than is possible if communication to a central scheduling processor is required. The stochastic nature of the manufacturing environment also suggests the suitability of a distributed system. Static models of the capabilities of individual resources are not sufficient, making it necessary to monitor the status of resources if reaction to change is to be feasible. While it is necessary to monitor resource status at a local level, it is desirable to hide as much of the local state of each resource as possible to reduce pathological coupling between components of the system. This is feasible in an appropriately distributed system.

The Contract Net

The contract net [Davis et al '83] models transfer of control in a distributed system with the metaphor of negotiation among autonomous agents. The net consists of a set of nodes that negotiate with one another through a set of messages. Three classes of node can be identified. The *manager* node identifies a task to be performed and assigns it to other nodes for execution. A *bidder* node offers to perform tasks, while the *contractor* node of a negotiation is the successful bidder, the one whose bid is accepted by the manager.

Nodes communicate by means of different classes of messages. Manager nodes issue *Task Announcement* and *Award* messages during the process of establishing a contract. The manager node may also send a *Termination* message to a contractor to interrupt its performance of a contract prematurely. Bidders send *bids*, while contractors send *Acknowledgements* to accept or reject awards and *Reports* indicating the status of a contract. Idle nodes may broadcast their availability with a *Node Availability Announcement.*

The Contract Net In Manufacturing

Certain aspects of the domain in which YAMS operates generate a need for modifications to the pure contract net metaphor. These modifications are required in the interests of problem-solving efficiency and the quality of schedules produced. In both instances, the need for modification can be viewed in terms of system volatility, a measure of how likely the system is to change during operation. For example, if a typical operation takes a few minutes to perform and the system remains stable for days at a time, it has a low volatility. On the other hand, if the durations of operations and period of stability are reversed, then the system is said to have a high volatility. Manufacturing domains have features which are highly volatile while at the same time other features of the domain have a very low volatility. Features such as load on a particular machine, product mix and machine tooling are highly volatile while others like the arrangement of machinery on the shop-floor, machine capabilities and basic resource costs have a low volatility.

In a pure contract net implementation, task announcements are made by broadcasting to all nodes in the net. Because the shop-floor configuration has low volatility, this approach wastes communication bandwidth. To overcome this inefficiency, YAMS adopts a strategy of *audience restriction*, in which a set of nodes called an *audience* is maintained for each task class. The audience consists of nodes which have responded to announcements for previous tasks of the same class. The audience is redefined after each round of bidding to include only those nodes that participated in that round. Later announcements for similar tasks go only to nodes in the appropriate audience.

The contract net metaphor is an efficient mechanism for managing highly volatile systems. In such a system, only local knowledge is likely to be accurate, and there is little point in gathering global knowledge as it will be obsolete before it can be used. Negotiation offers a reasonable way of using local knowledge to achieve acceptable performance. However, adopting a local scheduling approach, as in the contract net, can have a detrimental affect on the quality of the global schedule. Obviously, in a highly volatile domain, a system which runs at all is an achievement and optimality is not a central issue. The fact that manufacturing domains exhibit features of both high and low volatility give rise to two types of scheduling anomaly [Davis et al '83] which occur when using the contract net metaphor. Firstly, nodes can only see task announcements and bids that have already arrived and not those about to arrive. This is a result of *temporal ignorance*, a limited knowledge of the temporal constraints on operations in the schedule. The second anomaly, the compromise anomaly, concerns the situation in which global objectives are sacrificed to achieve local objectives. This is a result of both *spatial* and *loading ignorance*. Spatial ignorance is the result of limited knowledge of where operations in the schedule may be performed, while loading ignorance is the result of limited knowledge of the load on other resources in the factory.

Within YAMS, an attempt has been made to reduce the levels of temporal, spatial and loading ignorance, and therefore the impact of the scheduling anomalies introduced by the contract

net. This is achieved by distributing copies of a global schedule throughout the net, an approach which suffers from the very stochastic problems which motivated the use of the contract net in the first place. This is countered in YAMS in two ways, by adopting a policy of *turnpike scheduling* and utilising the fact that some subsets of the information in the global schedule are less likely to become wrong than others. Turnpike scheduling defines an approach for reacting to unexpected events. It suggests that in the face of such events, it is best to return to the previous schedule as quickly as possible rather than rescheduling. By doing this, it is hoped that the global schedule will remain valid longer.

Comment

The benefits of a distributed asynchronous architecture with respect to the focusing of problem-solving effort are significant. It allows effort to be focused on well defined local problems easily, thus permitting faster response times than would be possible in a centralised system. Also important is the ability to deal with multiple foci of attention simultaneously, a feature often required in a complex manufacturing domain.

While successful in a flexible manufacturing environment, the contract net is not suitable to all manufacturing domains. When using the contract net metaphor, the quality of the schedule produced is maintained by the existence of a competitive environment. Without competition, a single bidder may dominate the schedule to achieve its own local objectives. The approach adopted in YAMS to deal with this, the compromise anomaly, seems to highlight the fact that this really is not a suitable method of co-ordinating effort in an environment in which there is insufficient competition for tasks.

While the focused addressing mechanism employed in YAMS is aimed at reducing overheads incurred by broadcast addressing, it itself seems to be an unnecessary overhead. Due to the low volatility of the number of machines and their capabilities, it would seem sensible to give manager nodes a priori knowledge of potential bidders and their static capabilities. In this way manager nodes could easily determine where to send requests for bids by identifying the class of task to be

scheduled.

2.3.9. CSS [Ow et al '88]

The CSS (Cooperative Scheduling System) project provides another example of the trend towards distributed scheduling systems. It solves its problem in a manner which mimics a group of human schedulers cooperating to develop a schedule, and thus provides a vehicle to study the process by which such groups operate. The CSS project was motivated by the fact that while scheduling in a complex manufacturing environment is typically performed by a group of decision-makers, little is known about the process by which such groups operate.

Overview

The manufacturing facility studied is a large job-shop with about 1000 machines grouped into work-centres of similar machines. Work-centres operate under the supervision of shop foremen who work with shop expediters and schedulers to develop a daily production schedule. In addition, there is a supervisor who works with the foremen and schedulers to estimate completion dates of potential jobs. The job-shop builds orders to customer specification, and therefore the design and manufacturing process required for each job may vary greatly.

As in YAMS and ENTERPRISE, co-ordination of problem-solving activity is based on a contract negotiation process. However, as has been discussed, the contract negotiation mechanism employed in YAMS and ENTERPRISE is dependent on the existence of sufficient numbers of similar resources to generate competition in order to achieve a reasonable level of satisfaction for the goals of the manager. A lack of competition allows a single bidder to monopolise the schedule in a manner which satisfies its own local objectives. The mechanism employed in CSS is appropriate regardless of whether sufficient numbers of similar resources exist to compete for contracts.

CSS is comprised of two types of KS, a work order manager (WOM) and a resource broker. The WOM is responsible for co-ordinating the scheduling of a work-order, which is assumed to

have a pre-determined route, and estimating the completion time of potential jobs. It does this by requesting bids from the appropriate brokers. For each work-centre there is one resource broker which decides on how to bid after inspecting the current schedule at its work-centre. When a WOM receives bids from brokers, it selects the best set of bids to meet the objectives of the work-order.

Motivation For Decentralisation

Again, as in ENTERPRISE and YAMS, there are several reasons cited as the motivation for considering a distributed rather than a centralised system. The first concerns problem complexity. The complexity of most scheduling tasks demands that the problem be decomposed into more tractable subproblems. The structure of the job-shop scheduling domain suggests a natural decomposition around orders and work-centres. The degree of overlap between subproblems is relatively small, making the task of decoupling them easier.

By designing a distributed system, it is possible to retain the existing organisational structure of the human scheduling team. This has the benefit of providing the opportunity for the system to be used interactively by team members in a decision-support mode. It also tends to make the system more acceptable to its users.

In CSS, computational costs are reduced by adopting distributed processing as well as distributed problem-solving. Decomposing the problem into a set of more tractable subtasks permits the use of low cost hardware. Very often, networking a collection of standard low cost devices is considerably less expensive than a powerful specialised machine capable of solving the original non-decomposed task. Further, the introduction of concurrent processing can lead to a reduction in computation time.

Co-ordinating Scheduling Activity

The medium for communication between a WOM and brokers is modelled as a contract. Establishing a contract is a three stage process initiated by the first stage, a *call* which specifies a

request for work from a client (WOM) to a contractor (broker). A *call* contains information about the operation to be performed including the earliest time at which the operation may start. The second stage of a contract is typically a *bid*, which specifies a candidate time slot for the operation. The final stage, an *award*, represents the finalised contract. An award message, giving the temporal constraints of the operation concerned, is sent to indicate that a bid has been accepted.

The *calls* sent out by a WOM are directed only to brokers that can perform the operations in question, rather than broadcasted as in YAMS and ENTERPRISE. This performs a role similar to that of the focused addressing mechanism found in YAMS. A WOM does not send out *Calls* to the appropriate brokers simultaneously, rather they are staggered according to the precedence constraints imposed by the process plan. That is, a WOM will request bids for the first operation in a process plan and award a contract before requesting bids for the second operation in the process plan. This process is repeated in a left to right manner through the process plan.

As stated previously, the communication mechanism adopted by CSS does not rely on the existence of similar resources to generate competition for operations. It is not so much a need for competition between brokers that is required, but rather that the WOM should be provided with a number of options to select from when awarding a contract. This is provided for in CSS by allowing contractors to submit more than one bid, thus allowing the WOM greater flexibility in building a good work-order schedule.

Work Order Manager

The WOM provides estimated completion dates for prospective orders and is additionally responsible for finalising contracts with brokers, which results in an order being scheduled, when an order is accepted. Orders are accepted into the system by a human shop-floor supervisor. The objectives of the WOM are to minimise both the completion time and work-in-process time of the work order it is currently dealing with. The WOM has access to information about the capabilities of the various brokers and the manufacturing requirements of work orders.

A WOM processes only one work order at a time, sending out calls to brokers in a manner indicated by the precedence constraints of the process plan. The rationale for scheduling in process plan order is to avoid the generation of irrelevant bids. That is, a call is made only when all information regarding the constraints on the bids is known. Although synchronising decisions in process plan order will avoid the generation of irrelevant bids, it may do so at the cost of imposing artificial constraints on the problem. For example, if resource brokers are motivated to schedule operations as late as possible, the task of all but the first broker will be made more difficult by this approach. Only if brokers schedule operations as early as possible will this approach not over-constrain their tasks.

The WOM begins by sending out calls for all possible first operations, selecting the bid which best meets its objectives and tentatively scheduling the selected operation in accordance with the accepted bid. A similar process is initiated for all possible second operations, and so on until all operations have been tentatively scheduled. In order to choose between bids, the WOM constructs an AND/OR search tree which it searches using a two-pass best-first search strategy.

In making *awards*, the WOM tries to provide a degree of slack to each broker to give them some flexibility in scheduling its operation to meet the terms of the award. It does this by allocating longer than is needed to perform the operation. After examining the tentative schedule it has developed to estimate a completion date, the WOM can re-distribute queueing time as slack in the *awards*, thus providing brokers with a degree of flexibility.

Resource Broker

The resource broker represents a set of resources which can perform similar operations. The role of the resource broker is to find a resource and time slot which meets the constraints of the *calls* it receives. The WOM is not concerned with, and therefore does not need to know, the specific resource selected to perform an operation.

When a broker receives a *call* from a WOM, bids are generated which satisfy three hard constraints. The first requires that selected resources must be capable of performing the operation.

The second requires that the physical capacity constraints of the resource should not be exceeded and, finally, operation precedence constraints must not be violated. The candidate time slots generated, within the boundaries imposed by these constraints, are then evaluated against the local objectives of the resource broker. A subset of these are then returned to the WOM as bids.

Comment

The problem of multiple conflicting objectives is a major issue for most problem-solvers, and certainly for schedulers. By adopting a cooperative distributed framework, CSS permits the distribution of objectives amongst the various problem-solving agents. These agents can then negotiate a compromise solution among the conflicting objectives present. Such an approach has the benefit of highlighting the local objectives of the decision-makers throughout the system. By making local objectives accessible as parameters, they can be varied in order to investigate their effect on the global solution.

Another positive feature of CSS concerns the integration of order scheduling and the estimation of delivery dates for potential orders. In many applications, the scheduling task defined by an order book and promised delivery dates is impossible to satisfy even before executional uncertainty is introduced into the equation. This occurs because the delivery dates promised by sales staff very often do not take cognisance of the existing shop load or predicted product mix. Integrating the tasks of scheduling and estimating delivery dates greatly increases the likelihood that the scheduling problem posed does in fact have a solution.

Despite the fact that CSS does support distributed processing, it does not support complete asynchronicity. While bidders may operate concurrently, there is at most one WOM operating at any given moment. By considering only one order at a time, a WOM is unable to compromise between the objectives of multiple orders. This, coupled with the fact that only one WOM operates at a time, generates a situation similar to that found in ISIS, that is, all constraints relating to a particular order are considered in isolation of orders which are yet to be scheduled.

Further synchronisation is enforced by ensuring that scheduling decisions are made in process plan order. Bidders are only notified of an order when it is considered the correct time to do so. This technique, aimed at reducing the amount of backtracking performed during search, is only appropriate if operations are being scheduled using an early dispatch strategy. If, for example, operations are scheduled using a JIT strategy, it would be more appropriate to schedule backwards rather than forwards through the process plan.

In CSS, in order to ensure that a broker can meet the terms specified in a bid, it is necessary to ensure that only one bid is outstanding, awaiting a reply from a WOM at any given time. This is achieved by cancelling all unawarded bids whenever a call for a new prospective work-order is issued or an award is made for that work-order. Therefore, even if multiple WOMS are permitted to operate concurrently, individual brokers may only deal with one WOM at a time. In the situation where a broker is in a position to bid for operations from more than one order, this is another limitation on asynchronous behaviour. This is a consequence of the localised view of problem-solving encouraged when using the contract net as a means of co-ordinating distributed activity.

2.4. Conclusions

This chapter has given a detailed account of the major scheduling systems of interest to the AI community. It is appropriate to recap, in the form of a brief history, before presenting concluding remarks. This section gives that history, draws a conclusion from it and states briefly its relevance to the work presented in the remainder of this thesis.

Scheduling as a research area within AI began in earnest with the development of ISIS [Fox M.S. '83], the first scheduling system to address the issues arising out of the complexities of typical real world scheduling domains. ISIS was developed at CMU and brought a rich constraint representation and knowledge intensive search techniques to the scheduling problem. However, ISIS and its immediate successors (eg. SOJA [LePape '85]), view scheduling as a predominantly

predictive task, with little or no attention being paid to reactive requirements. It was again a research group based at CMU, many of whom had been involved with the ISIS project, which made the next significant step towards scheduling in real world domains. They developed OPIS [Smith '87], a scheduling system which embraces the opportunistic problem-solving approach advocated by Fox and Kempf [Fox B.R. et al '85a]. OPIS provides a reactive scheduling capability, viewing both predictive and reactive scheduling as an opportunistic process. Both OPIS and SONIA [Collinot et al '88], a system similar in nature to OPIS, attempt to integrate closely the tasks of predictive and reactive scheduling. Despite recognising that "predictive and reactive components appear to refer to common pieces of knowledge" [Collinot et al '88], both OPIS and SONIA employ separate mechanisms for predictive and reactive scheduling. S2 [Elleby et al '88] was the first scheduling system to view the task as being predominantly one of schedule maintenance rather than creation. Within S2, the same mechanisms are employed to perform predictive and reactive scheduling.

At the same time as the developments described above were taking place, advances in computing technology stimulated work in the area of distributed processing. As discussed earlier, a distributed processing capability offers many potential benefits to the scheduling problem. One of the earliest systems to recognise this fact was ENTERPRISE [Malone et al '83], a system which provides a facility for sharing tasks among available processors on a network of personal computers. The communication protocol used by ENTERPRISE has much in common with the contract net metaphor of [Davis et al '83]. YAMS [Parunak et al '86] is perhaps a better known example of a decentralised scheduling system. It too bases its communication protocol on the contract net metaphor. When using the contract net metaphor, the quality of solution produced is maintained by the existence of a competitive environment. Without competition a single bidder may dominate the solution to achieve its own local objectives. CSS [Ow et al '88] is a distributed scheduling system which also bases its communication protocol on the contract net. However, the variation of contract net employed in CSS is not dependent on competing resources to maintain schedule quality.

There has not been a great deal of research in the area of distributed scheduling, but as has been shown, the existing work is based largely on the contract net metaphor. For a great many scheduling domains, and manufacturing in particular, this is not a good method of co-ordinating problem-solving effort. A more suitable approach to this problem for many applications is offered by communication via *message-passing* [Hewitt '77]. Smith and Hynynen [Smith et al '87] present a scheduling framework based on cooperative problem-solving, distributed according to a hierarchical factory model and co-ordinated via message-passing. DAS shares some of the features of the Smith and Hynynen model, most notably its hierarchical architecture and message-passing as a negotiation protocol. This is discussed further in chapter 3.

The advances made in scheduling technology over the past decade have been significant. Techniques which permit multiple-perspective scheduling, opportunistic problem decomposition, reactive scheduling and adaptive scheduling bring nearer the possibility of an automated system which can deal with both the complexity and stochastic nature of the real world. At the same time, advances in computing technology have made distributed processing feasible. The benefits which distributed processing can offer the scheduling problem are numerous. There is an opportunity to combine the benefits of distributed processing, which can deal with the distributed asynchronous nature of the real world, with the recent advances in automated scheduling techniques. Together, this combination can address the fundamental problems of combinatorial complexity, executional uncertainty and conflicting scheduling objectives.

The remaining chapters of this thesis present DAS, a distributed asynchronous scheduler. DAS relies on message-passing for inter-agent communication and incorporates many of the advances discussed in relation to the centralised scheduling systems reviewed. Consequently, DAS owes much to the developments over the past decade in both centralised and decentralised scheduling techniques. In particular, the remaining chapters demonstrate how DAS caters for scheduling in a dynamic environment, and how it deals with the issue of managing problem-solving effort.

CHAPTER 3

DAS: A Distributed Asynchronous Scheduler

This chapter presents DAS, a distributed asynchronous scheduler which combines recent advances in scheduling techniques with the power of distributed processing. Because DAS has been designed to cope with the harsh reality of manufacturing scheduling environments, it does not conform to the traditional view of scheduling, and, in fact, has its own philosophy concerning the very nature of the task. The DAS philosophy is presented first, as it is fundamental to an understanding of some of the design decisions which are be presented later. Having established how DAS views the scheduling problem, representational issues within it are discussed. A detailed description of how factory resources, items of work and the schedule itself are represented within DAS is given and, where appropriate, the influence of the DAS philosophy on representational issues is highlighted. The final section of this chapter presents the architecture which supports DAS and its view of scheduling. The main features of the architecture are that it is distributed, asynchronous and hierarchical. A detailed argument in favour of these features follows a description of the architecture itself. The section concludes by commenting on the relationship between the architecture and other work in the area.

3.1. DAS Philosophy

Scheduling Environment

Traditional views of the scheduling problem have to a large extent been influenced by the static world assumptions discussed in chapter 2. Unfortunately, neither these assumptions nor the traditional views they encourage hold true in the real world. Unlike a large proportion of existing work in the scheduling area, DAS caters for scheduling problems which occur in real world environments, and consequently views the scheduling environment as a *stochastic* rather than a

deterministic one. Executional uncertainty, which can manifest itself in the form of machine breakdowns, late arrival of work or simply an operative not adhering to the specified schedule, is the primary source of the stochastic nature of the environment. Even if it were possible to exclude executional uncertainty and hence stochasticity, the static world view of scheduling remains inaccurate. Scheduling is a highly *dynamic* task with external factors combining to change the problem and its objectives. In the short term, the introduction of new work via the order book may significantly change the problem, while in the longer term interest rates, raw material costs and other economic factors may alter its objectives.

Objective

Traditionally, the scheduling task has been viewed as an optimisation problem. There are several reasons why DAS prefers to view the task primarily as one of *satisfaction*, with *optimisation* a secondary consideration. As discussed in chapter 1, defining the criterion to be used as a measure of schedule merit is a very difficult, if not impossible, task. Even if it were possible to define a suitable measure of schedule merit, the dynamic and stochastic nature of the scheduling environment combine to ensure that generating a satisfactory solution, let alone an "optimal" one, is a far from trivial exercise. A third problem concerns the fact that many of the problems posed to a scheduling system, human or automated, are infeasible at the outset. Infeasible scheduling problems are set because salesmen, often responsible for setting delivery dates, are generally unaware of the state of the shop floor and the existing order book, and therefore cannot give realistic delivery dates. Even with an accurate view of the shop floor and the order book, it is not an easy task to estimate a delivery date. The difficulties described above lead to the conclusion that it is reasonable to view scheduling as a satisfaction rather than optimisation problem.

Predictive Vs. Reactive

Traditionally, most scheduling systems, IKBS or conventional, attempt to schedule some time into the future to what is considered to be a sensible or desirable time horizon. Assuming that the generated schedule remains workable for the period it covers, the temporal horizon serves to define the frequency with which a new schedule should be created. Unfortunately, in most situations, the period for which a schedule remains an accurate account of what has actually happened is a small fraction of the period it is intended to cover. As the schedule drifts further from reality, it becomes more difficult and less beneficial to adhere to it. One approach to this problem is to completely recreate the schedule whenever it drifts beyond a certain point. The time required to create a fresh schedule every time this happens makes this approach prohibitive in many applications. The most common alternative is to repair or "patch" the existing schedule, manually or otherwise, as and when things go wrong. Automated versions of such an approach rely mainly on a range of ad hoc techniques, the relevance of which is usually determined by some fairly high level heuristics. More typically, human schedulers find themselves engaged in a near constant stream of "fire-fighting" tasks in an attempt to maintain the schedule in a usable form. This is a poor situation for two reasons. Firstly, humans are not particularly good at dealing with this low level type of task, and secondly it generally prevents them from dealing with the higher level scheduling issues to which they are better suited. DAS prefers to take the view that schedule *creation* and schedule *maintenance* are essentially the same task, and should be performed using the same mechanisms. Both appear to require the same knowledge and have the same objectives, namely to produce a schedule which satisfies the current set of problem constraints.

Systems which produce schedules periodically are essentially *predictive*, with some occasionally possessing a limited *reactive* capability. In order to survive in a highly dynamic environment, DAS is predominantly reactive. Like OPIS [Smith '87], DAS merges top down dynamic problem decomposition with bottom up reaction to problems encountered during schedule synthesis or unanticipated events in the production environment. DAS takes a unified view of both

predictive and *reactive* scheduling and can therefore use exactly the same techniques and knowledge for both. The concept that schedules should be *maintained* rather than recreated at regular intervals is completely consistent with the view that scheduling in a highly dynamic environment should be primarily *reactive* and event-driven. However, taking a predominantly *reactive* approach in no way suggests that there is nothing to be gained from taking a *predictive* approach at a higher level where things are generally less dynamic.

The requirement for regular schedule *maintenance* and a *reactive* rather than *predictive* approach to scheduling strongly suggests that it is an *on-line* rather than *off-line* task. In order to react to problems in an appropriate and timeous manner, and hence maintain schedule workability and quality, it is necessary to have an accurate and up-to-date knowledge of shop-floor events. To derive maximum benefit from a scheduling system such as DAS, it must be integrated with some form of shop-floor reporting system and be run *on-line*.

Problem-Solving Approach

Opportunistic problem-solving [Fox B.R. et al '85a], made appropriate by the highly *dynamic* and *stochastic* nature of the task, forms an integral part of the DAS philosophy. DAS subscribes to the view that it is better to maintain a representation of the set of all possible schedules than to commit prematurely to a single schedule. This approach exploits executional uncertainty in a partial solution to the inherent problem of combinatorial complexity while at the same time exploiting the problem of combinatorial complexity in a partial solution to the problem of executional uncertainty. On the one hand, it allows executional uncertainty to reduce the size of the search space to be traversed. On the other hand, the inherently large size of the search space is utilised to provide alternative solutions when executional uncertainty excludes the current solution. Dynamic problem decomposition, introduced by OPIS [Smith '87], represents another form of opportunism which is also embodied within the DAS philosophy. This form of opportunism is again made appropriate by the dynamic and stochastic nature of scheduling problems found in manufacturing domains. As will be discussed further in chapter 4, DAS

introduces further opportunism in its approach to identifying difficult, and therefore important, areas of the schedule. The mechanisms used for co-ordinating problem-solving effort within DAS also operate in a completely opportunistic manner.

Another component of the DAS problem-solving approach concerns the impact of events to which DAS must react. It is reasonable to expect that many events which require reaction will be relatively benign. They are benign in the sense that, although they may cause the schedule to drift from reality, the deviation needs only local reaction in order to return to a feasible schedule. DAS always attempts to localise the impact of any reaction in the interests of system responsiveness, computational efficiency and schedule stability.

3.2. DAS Representation

Scheduling is a knowledge-intensive activity in terms of both domain knowledge and general scheduling expertise. The representation of this knowledge is a major issue for all scheduling systems and is one of the major areas in which AI-based approaches offer benefits over more traditional operational research approaches. DAS employs frame-based knowledge representation techniques supported by KEE [Intellicorp '86], the software development environment within which DAS was constructed. The frame-based representation facility provided by KEE also supports procedural attachment and hence object-oriented programming. Both these features have been used extensively to deal with the representational issues present within DAS. This section gives a detailed account of how DAS models its environment and the schedules it produces within a frame-based structure. It describes the various knowledge-bases in DAS followed by the major representation elements, namely resources, operations and plans. The use of access-oriented programming within DAS is discussed in chapter 4.

3.2.1. Knowledge Bases

One of the requirements imposed on the design of DAS is that it should, as far as possible, be a generic scheduler. To this end, there has been an attempt to separate domain-specific

knowledge from general scheduling knowledge. This has resulted in a design comprising four knowledge bases, as shown in figure 3.1.

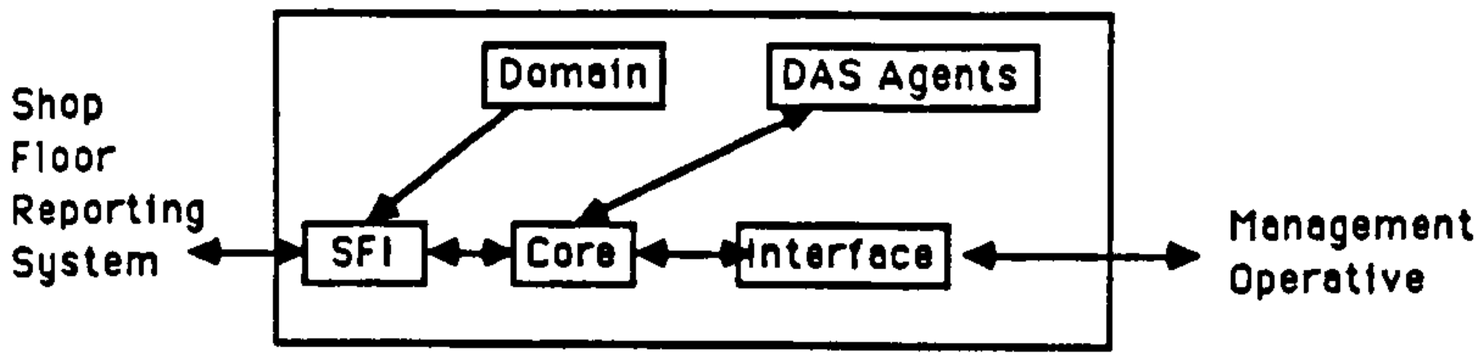

Figure 3.1

As well as showing the four knowledge bases of DAS, figure 3.1 also includes a *shop-floor reporting* (SFR) system and a *management operative* as external interfaces. Both place very different needs on DAS and highlight the dual nature of its functionality. On one hand, the *SFR* system, and ultimately the shop-floor, views DAS as an operational level scheduling system which produces *work-to-lists* for operatives and resources on the shop floor. On the other hand, a management operative may view DAS more as a strategic level tool which provides assistance when making management decisions about labour requirements, preventive maintenance schedules and even general scheduling policies. Additionally, a management operative may also wish to intervene and over-ride some of the decisions made by DAS if it is felt that DAS lacks knowledge in a specific area and is consequently making poor decisions.

The final component of figure 3.1 is a box marked "DAS agents". DAS decomposes the scheduling task across a three tier hierarchy. Each node in the hierarchy corresponds to a subproblem and has an associated problem-solving agent. At the lowest level in the hierarchy, the operational level, each problem-solving agent is an *O-agent*. At the intermediate level of the hierarchy, the tactical level, each problem-solving agent is a *T-agent*. The highest level, the strategic level, has a single *S-agent* as its problem-solving agent. Section 3.3.1 presents a full description of each type of problem-solving agent.

SFI

The *SFI* (shop floor image) knowledge base is required to perform the translation from a shop floor view of the world which is highly domain-specific, to a DAS view which is domain-independent. In the case of the demonstrator site, British Alcan at Kitts Green, the shop-floor deals in terms of aluminium plates, collections of plate referred to as lots, and process plans. DAS deals in terms of plans and operations. Constraints are attached to plans and operations in order to create an accurate model of the plates, lots and process plans they represent. Work enters DAS from the order book through the *SFI* knowledge base in the form of plates, lots and process plans and is subsequently converted into DAS operations and plans. Whenever DAS makes a scheduling decision, it does so on its representation of the world. Therefore, there is a need to convert scheduling decisions made on DAS operations back into the entities recognised by the SFR. This process is again performed by means of the *SFI* knowledge-base.

Core

Operations and plans generated via the *SFI* knowledge-base are introduced into the *core* knowledge-base. The *core* knowledge-base views jobs only in terms of operations and plans, and is the only knowledge-base accessed by DAS when making scheduling decisions. Other than plans and operations, the *core* knowledge-base contains a hierarchical structure to which scheduling-specific rather than domain-specific knowledge is attached. More will be said about this structure in section 3.3. The *core* knowledge-base, and the objects held within it, collectively represent the current set of problem constraints.

Interface

The *interface* knowledge-base, as its name suggests, deals with the MMI aspects of DAS. A management operative will almost certainly want to have the ability to display schedules, partial schedules and numerous other aspects of the scheduling environment. All objects created in order to display the desired information are held within the *interface* knowledge-base. Examples of the

various displays available are given in chapter 5. As well as fulfilling an MMI requirement, the *interface* knowledge-base and the functionality it supports also provided a very useful development debugging tool.

Domain

The fourth and final knowledge-base, the *domain* knowledge-base, holds all the domain specific-knowledge required by DAS. This is used by the *SFI* knowledge-base when performing the translation from a domain-specific representation to the domain-independent representation. The *domain* knowledge-base knows nothing about the representations used by DAS and would require to be entirely rebuilt for each individual scheduling site. The *SFI* knowledge-base would require only partial rebuilding, whereas the active elements of DAS, the *core* and *interface* knowledge-bases would remain intact from site to site as long as there is no requirement for additional functionality.

3.2.2. Resources

The resources with which DAS is primarily concerned are shop-floor resources, and machine centres in particular. Shop-floor resources are modelled at two levels of abstraction, ie: at an individual resource level and at an aggregate resource level. Each individual resource on the shop-floor has an associated frame in the *core* and *domain* knowledge-bases. A typical frame from each knowledge-base is shown in figures 3.2 and 3.3 respectively. Within the *core* knowledge-base, the frame associated with a particular resource contains slots which indicate scheduling preferences, facilitate a dual representation for agents and give details of the local scheduling problem. The scheduling preference information concerns choices such as how to select the next operation to schedule and how to choose a value to give the selected operation a start time. For example, on a particular machine it may be best to schedule operations using a JIT strategy, whereas on others it may be better to adopt an earliest dispatch strategy.

Each agent in the system exists as a unique software process operating asynchronously with respect to other agents in the system. *O-agents* are attached to specific resources via its *process* slot. The value in slot *process* of a particular resource corresponds directly to the software process being used to implement its associated *O-agent*. *O-agents* have their own internal view of the world, held within the *internal.representation* slot of a resource. The *message.buffer* slot of a resource is crucial to the maintenance of this internal representation. It is through messages sent to this buffer that an agent updates its internal representation. The *each.tick* slot on a resource allows the granularity of time considered at a resource to be varied in accordance with the typical duration of the operations processed there. For example, it is not appropriate to schedule an operation of duration twenty four hours to an accuracy of one second. The value held in the *each.tick* slot identifies the smallest unit of time considered to be significant at a resource. An *O-agent* can therefore reduce the size of an operation's domain by a factor of *each.tick*, thus reducing the scale of the scheduling problem at an operational resource.

Each specific resource represented within the *core* knowledge-base has both a *scheduled.ops* slot and an *unscheduled.ops* slot. The *scheduled.ops* slot contains a list of operations which have been given start times on that resource. The *unscheduled.ops* slot contains a list of operations which have been allocated to this resource but as yet have not been given start times. Together the *scheduled.ops* and *unscheduled.ops* slot define the scheduling problem at a resource.

UNIT: saw.20.foot
MEMBER OF: operational.resources

SLOT NAME	VALUE
CONSTRAINT:	single.server
EACH.TICK:	180
INTERNAL.REPRESENTATION:	#<*resource* 374055175>
MESSAGE.BUFFER:	unknown
PROCESS:	#<process SAW.20.FOOT 15442365>
QUANTUM:	2000
SCHEDULED.OPS:	L1258.saw, L2541.saw, L2441.saw
SELECT.OP:	most.constrained
SELECT.VALUE:	jit
REST:	5
UNSCHEDULED.OPS:	L8412.saw

Figure 3.2

Figure 3.2 presents a frame from the *core* knowledge-base which is used to represent resources on the shop floor. Only the more relevant slots are shown, including some to be discussed later.

Within the *domain* knowledge-base, the frame representing a specific resource contains non-negotiable technological constraints. These include physical dimension constraints, weight constraints and the range of processes which can be performed on the resource. Methods used to calculate machine setup times required between different types of work, the time required to load an item of work and the time required to process an item of work are also held here.

UNIT: saw.20.foot MEMBER OF: operational.resources	
SLOT NAME	VALUE
BREAKDOWNS:	8%
CONSTRAINT.HEIGHT:	3 foot
CONSTRAINT.LENGTH:	20 foot
PROCESSES:	final.sawing
TIME.TO.LOAD:	saw-20-load-method
TIME.TO.PROCESS:	saw-20-process-method
TIME.TO.SETUP:	saw-20-setup-method

Figure 3.3

The aggregate representation of resources exists only in the *core* knowledge-base. An aggregate resource is a collection of similar resources. For example, if there are three annealing furnaces in the factory, there will be a single frame in the *core* knowledge base representing the aggregation. The information held here is a combination of scheduling preferences, an internal representation of the agent's problem and a partial definition of the local schedule. Each aggregate resource has an associated temporal horizon, defined in slot *look.a.head*, and a preferred method for selecting the next operation to delegate, held in the *select.strategy* slot.

T-agents are attached to aggregate resources via the *process* slot of the appropriate aggregate resource. In exactly the same way as for individual resources, the value held in slot *process* of an aggregate resource corresponds directly to the software process being used to implement its associated *T-agent*. *T-agents* also have their own internal view of the world held within the *internal.representation* slot of an aggregate resource. The *message.buffer* slot of an aggregate resource is again essential to the maintenance of this internal representation.

The scheduling problem at an aggregate resource is partially defined by the list of operations held in the *op.store* slot. For any particular aggregate resource, this slot holds a list of all the

operations which must be processed by one of its subordinate resources but which have not yet been allocated to a specific resource. Figure 3.4 shows the relevant slots present in a frame used to represent aggregate resources in the *core* knowledge-base.

UNIT: saws MEMBER OF: aggregate.resources	
SLOT NAME	VALUE
INTERNAL.REPRESENTATION:	#<*tactical-resource* 25633113
LOOK.AHEAD:	30
MESSAGE.BUFFER:	(saw.wessex complete),(add L1412.saw)
OP.STORE:	L1412.saw, L8331.saw
PROCESS:	#<process saws 15442505>
REST:	10
SELECT.STRATEGY:	latest

Figure 3.4

In addition to operational and aggregate resource representations, DAS also has a *strategic.unit*. The *strategic.unit* is concerned with the scheduling problem as a whole. It contains the slots necessary to provide the *S-agent* with a dual representation of its problem. These include *process*, *internal.representation* and *message.buffer* slots. The scheduling problem at the strategic level deals with plans (discussed in section 3.2.4) rather than operations. This is reflected in the representation of the scheduling problem maintained at the *strategic.unit*. It has a *plan.store* slot containing a list of plans which have not yet been delegated to the tactical level, and a *started.plans* slot containing a list of plans which have either been partially or wholly delegated to the tactical level. The only scheduling preference information currently held at the *strategic.unit* concerns the selection strategy used when deciding which plan to delegate to the tactical level next. It is envisaged that preferences which affect the nature of the generated schedule will also be held here. The strategic unit is shown in figure 3.5.

UNIT: strategic.unit MEMBER OF: entities	
SLOT NAME	VALUE
INTERNAL.REPRESENTATION:	#<*strategic.level* 40045041>
MESSAGE.BUFFER:	unknown
PLAN.STORE:	P1419, P1347
PROCESS:	#<process strategic.unit 15442567>
REST:	20
SELECT.STRATEGY:	first
STARTED.PLANS:	P1258, P2541, P2411, P1412, P8331

Figure 3.5

Supporting or secondary resources for these main resources, with the exception of labour, are not catered for in the current version of DAS. To include such a feature requires an extension to, rather than a re-design of, DAS. Within the demonstrator site, secondary resources, such as loading cranes and transportation pallets, do exist. However, as they do not represent a limiting resource in terms of scheduling, it was considered reasonable to treat them as infinite resources. For all the resources which are to be scheduled, labour is assumed to be a necessary supporting resource.

3.2.3. Operations

An operation represents a particular process which must be performed during the manufacture of a specific work item. For example, the ultrasonic scanning process of lot B514 will be represented by an operation, as will any other process carried out on lot B514 during its manufacture. Operations satisfy three vital representation requirements within DAS. Firstly, they collectively model the order book of work to be scheduled, or more precisely the part of the order book which has been released into the system. Secondly, they implicitly define the current schedule and finally, they provide a means of representing unavailable resources through special operations.

Each operation is represented by a frame which has slots containing technological and temporal constraints. The major temporal constraints imposed on an operation are its duration, and the release date and due date of the lot it represents. Since the duration of an operation may vary according to whichever specific resource it is assigned, this variation in temporal constraints must also be represented.

The technological constraints imposed on an operation by the size, weight and processing requirements of the work item it represents are condensed into a single slot at the time the operation is created. The *SFI* knowledge-base accesses the *domain* knowledge-base when creating an operation to determine which resources are capable of performing the required process on the

work item it represents. The slot *possible.resources* contains a list of resources capable of performing the operation and effectively represents the technological constraints acting on the operation. Each entry in the list is a pair of the form *(Resource Duration)*, the second element being the duration of the operation on that particular resource.

Two further slots on an operation, *start.time and resource*, implicitly define the current schedule. When every operation in the system has been allocated to a resource and given a start time, the schedule is complete. As stated previously, the opportunistic nature of DAS requires that the set of all schedules allowed by the current set of problem constraints be maintained and not just the current schedule. The *possible.resources* and *legal.starts* slots of operations combine to achieve the desired representation. The *legal.starts* slot of an operation contains an ordered list of intervals indicating when it is possible to schedule this operation and remain consistent with all intra-plan scheduling decisions and other sources of temporal constraints. The value held in the *legal.starts* slot is maintained by a constraint maintenance system discussed in chapter 4.

The final slot to be discussed here is the *priority* slot of an operation. The priority of an operation provides a measure of the degree of difficulty experienced when trying to schedule this operation. This should not be confused with a user-defined metric such as customer importance. The value held in this slot is maintained by the appropriate *T-agent*. This is discussed further in chapter 4. Figure 3.6 shows the more important slots present in a frame used to represent an operation.

UNIT: L2541 MEMBER OF: operations	
SLOT NAME	VALUE
AFTER:	L2541.packing
BEFORE:	L2541.stretch
COMPOSITE.OP:	unknown
CONSTITUENT.OPS:	unknown
DURATION:	3960
END.GUESS:	2832520319
EXTERNAL.LEGAL.STARTS:	((2832516360 2832516360))
LEGAL.STARTS:	((L2541.packing start) (0 2832516360)), ((L2541.stretch start) (2832516360 999999999)), ((L2541.anneal start) (2832516360 9999999999)), ((L2541.packing due) (0 2956996798)), ((L2541.anneal guess) (2832514737 9999999999))
NOT.DURING:	unknown
OP.TYPE:	simple
PLAN:	P2541
POSSIBLE.RESOURCES:	(saw.20.foot 3960), (saw.ty.sa.man.1 2700), (saw.wessex 3600)
PRIORITY:	0
PROCESS:	saws
RESOURCE:	saw.20.foot
SELECT.VALUE:	jit
START.TIME:	2832516360
TYPE:	work

Figure 3.6

Unavailable Resources

Within DAS, resources are assumed to be available twenty four hours a day, seven days a week unless otherwise specified. In the current implementation, three situations which invalidate this assumption are catered for. A resource can be unavailable due to lack of labour, preventive maintenance or as a result of machine failure. All three are represented as a special type of operation, thus retaining a consistent view of the scheduling task for DAS. A lack of labour is represented by a *no.shift* operation which has a single value in both its *possible.resources* and *legal.starts* slot. Preventive maintenance tasks are represented by a *maintenance* operation which has a single value in its *possible.resources* slot but may have a range of values in its *legal.starts* slot. A machine failure is represented by a *repair* operation which has a single value in its *possible.resources* slot and a single value indicating the time of machine failure in its *legal.starts* slot. All three of these operations are identical to the operations which represent work items with the exception of the values held in slot *type*. Slot *type* is used to specify the class of operation from a range of *work*, *no.shift*, *maintenance* or *repair*. It is possible to schedule non-work operations in exactly the same manner as *work* operations because the technological and temporal

constraints attached to these operations ensure that they are scheduled appropriately.

Composite Operations

Until now, the operations discussed have all been simple operations representing exactly one lot on the shop floor. In certain situations there is a requirement to group work from different orders for processing. Batching work for a furnace treatment or a particular sawing operation are both examples of such situations. Grouping work for processing effectively creates implicit temporal relationships between operations from different lots which were hitherto unrelated. This may be represented within DAS via the concept of a composite operation, a structure which makes explicit the additional implicit temporal relations. In the furnace batching example, each lot, which is a member of the batch, is represented as an operation and the furnace load as a whole is represented as a composite operation.

There are three types of composite operation available in DAS, namely *concurrent*, *sequential* and *permutable*. A *concurrent* composite operation is one in which all the constituent operations must be performed simultaneously, as is the case in furnace batching. A *sequential* composite operation is one which specifies the sequence in which constituent operations must be performed, a useful structure when attempting to minimise machine setups. A *permutable* composite operation is one which places no additional temporal relations between the constituent operations, but does specify that they should be performed together with respect to other operations. This may be useful in order to allow some shop-floor operative discretion, while maintaining overall control of the schedule.

All operations have an *op.type* slot which specifies whether it is a *simple*, *concurrent*, *sequential* or *permutable* operation. They also have a *constituent.ops* slot and a *composite.op* slot. In the case of simple operations, the *constituent.ops* slot is always empty. The *composite.op* slot of an operation contains the name of any composite operation of which this operation is a constituent. It should be noted that composite operations can be cascaded together to create a hierarchy of operations, that is, there is no requirement for constituent operations to be simple

operations.

The main reason for introducing composite operations into the representational repertoire of DAS was to cater for the need to batch work for particular resources. Initially it was thought that batches would be created, and represented by composite operations, at either the strategic or tactical level. However, after some considerable effort, it was recognised that creating batches at any level other than the operational level would impose artificial constraints on the problem and rob DAS of a great deal of its reactivity. As a result, batches are created and maintained by *O-agents* (see section 3.3.1 and [Burke et al '89]). Composite operations are no longer required to make explicit the relations formed during batch creation because the *O-agent* writes its decision to every operation in the batch; thus intra-plan constraint propagation is sufficient.

3.2.4. Plans

Each lot in the system is represented by a process plan consisting of a collection of operations linked together by temporal precedence constraints. The temporal precedence constraints may describe a non-linear process plan. A non-linear plan is one which specifies that at least two of its operations may be permuted. The definition of planning given in chapter 1 was selected because it allows maximum opportunism within the scheduling process. Essentially it embodies the principle that if there is no technological reason for one operation to precede another, the output of the planning stage should be a non-linear plan which defers sequencing of operations until there is a good reason to do so.

Although the temporal relations used within plans are based on Allen's theory of action and time [Allen '84], only a subset of these relations are currently employed. Allen [Allen '84] identifies seven temporal relations, growing to thirteen if inverses are included. Due to the granularity of representation, ie. DAS does not cater for secondary resources, the *before* relation and its inverse *after* are sufficient to represent the majority of temporal relationships within a plan. Non-linearity is represented within DAS via the *not.during* relation. The relation A *not.during* B is defined to mean that A may be executed *before* or *after* B and vice-versa, but that the execution

of A and B must not overlap. Within Kitts Green, the processes of ultransonic scanning and precipitation treatment may be performed in any order. Having the ability to represent non-linear plans allows DAS to defer the sequencing decision until it has a good idea of the respective machine loads at the appropriate time and make a more informed decision.

Relations such as *meets*, *overlaps*, *during*, etc would be a necessary addition to DAS if it were extended to deal with secondary resources. For example, if a crane is to be scheduled to perform a loading operation immediately before a furnace treatment process is initiated, the loading operation may be linked to the furnace operation by the *meets* relation rather than the *before* relation. All other relations identified by Allen (and excluded from DAS) also deal with situations in which operations must overlap in time or run contiguously, typical of the relationship between primary and secondary operations.

Each plan represents a planned lot which has a release date and a due date. Lots also have a *release.guess* slot which provides an estimate of the release date, before the exact release date is known. The release date of a lot is given by the time the lot actually enters the shop-floor. The due date of a lot specifies the date on which it is desirable to complete the lot. These temporal constraints, imposed on lots and the plans used to represent them, are also represented on the operations which constitute a plan. The first operation in a plan is given the release date while the last operation is given the due date of the lot being represented. Dummy first and last operations can be introduced to cater for the situation where there are multiple possible first and last operations. The effective temporal constraint on an operation is held within its *legal.starts* slot.

Figure 3.7 shows the more important slots in a frame used to represent a process plan. The slot *decision.order* shown in the figure allows a decision-making ordering to be imposed on the operations within the plan. It does this by dictating a sequence for the release of operations into the lower levels of the scheduling system. This is discussed in more detail later.

UNIT: P2541 MEMBER OF: plans	
SLOT NAME	VALUE
DECISION.ORDER:	(L2541.saw L2541.packing)
DUE.DATE:	unknown
OPERATION:	L2541.stretch, L2541.anneal, L2541.packing, L2541.saw
RELEASE.DATE:	unknown
RELEASE.GUESS:	unknown

Figure 3.7

3.3. DAS Architecture

In order to realise the goal of a scheduling system which combines recent advances made in the field of scheduling with the power of distributed processing, it is first necessary to develop an architecture capable of supporting it. This section discusses the benefits of adopting a distributed asynchronous approach to scheduling, and gives a detailed account of the DAS architecture. Having presented the architecture, arguments for the design decisions taken are then presented.

3.3.1. A Distributed Asynchronous Architecture

The architecture presented here is an analogue of an idealised management structure in that it is hierarchical, distributed, and depends heavily on communication for success. It is a three tier hierarchy consisting of a strategic, tactical and operational level as shown in figure 3.8.

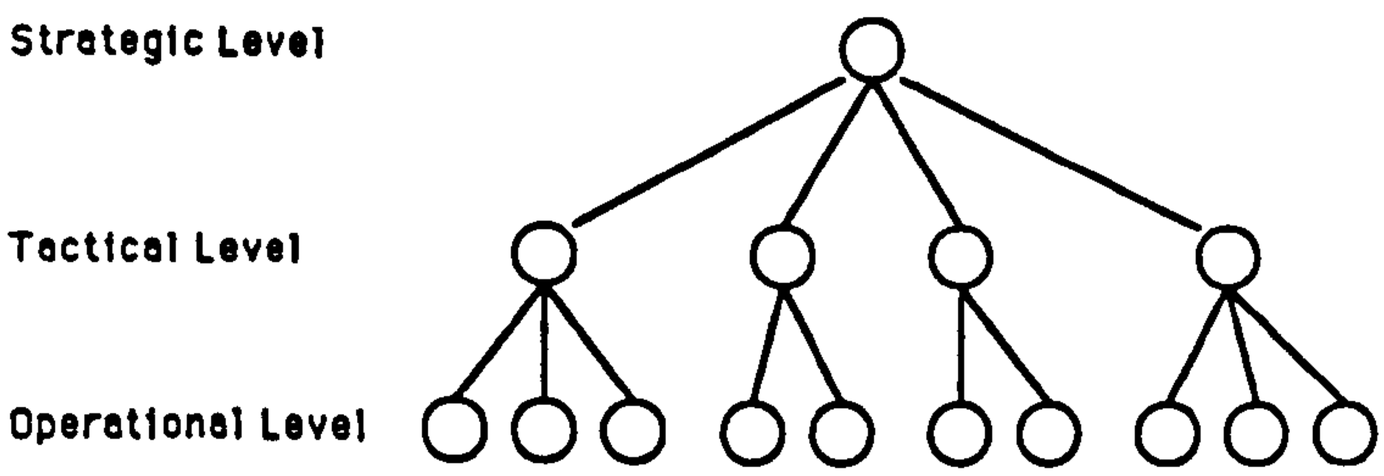

Figure 3.8

Each level in the hierarchy defines one or more scheduling focal points, within which decision-making may proceed asynchronously with respect to all other focal points. Each focal point has an associated scheduling agent responsible for all activity at that point in the hierarchy. The type of scheduling agent associated with a node is dependent upon the level of the node in the

hierarchy. Communication between nodes in the hierarchy is achieved via message-passing and may occur in both directions across manager/subordinate relations defined by the hierarchy, and on a peer-to-peer basis within levels of the hierarchy. Much of this communication is supported by a constraint propagation system.

As the functionality of each level is dependent on the functionality of the levels below it, it will be easier to describe the components of the hierarchy in a bottom-up manner. Therefore, the operational level will be considered first, followed by the tactical level, then the strategic level.

Operational Level

The operational level of DAS corresponds to the shop-floor level within the management structure analogy. Each shop-floor work-centre constitutes a scheduling focal point and has an associated *O-agent* to deal with its scheduling needs. An *O-agent* takes a fairly local, short-term view of its scheduling problem, but is kept informed of external events through regular communication. As is the case with shop-floor operatives, *O-agents* take part in considerably more peer-to-peer communication than manager/subordinate communication, except in times of difficulty. The *O-agent* is not part of the work being presented in this thesis. It is described here in order to allow a fuller understanding of the work which is.

Problem Definition

The role of an *O-agent* is to generate a schedule for its associated work-centre. This involves giving start times to all the operations allocated to its associated work-centre. By the time work has filtered down the hierarchy to the operational level, all the hard and the majority of soft constraints associated with it have been converted into temporal constraints. Therefore, when producing a schedule, the *O-agent* is concerned only with giving operations start times which satisfy the temporal constraints acting on them and the capacity constraints of its resource.

Despite reducing the task of an *O-agent* to one in which it need only consider a well-defined set of temporal and capacity constraints, the task which remains is not a trivial one. The primary

source of difficulty for an *O-agent* arises from the fact that its problem is open rather than closed. It is open in the sense that work may be added to or deleted from its problem, the temporal constraints on operations within its problem may change and executional uncertainty on the shop floor has the potential to render any scheduling decisions invalid. The actual operations it must schedule may vary as new work is added into the system, orders are scrapped or work is load-balanced onto other resources in order to resolve a conflict. The temporal constraints on the operations to be scheduled vary in accordance with the scheduling decisions of other *O-agents* and the reporting of when events actually occur as opposed to when they were expected to occur. As well as causing modifications to the temporal constraints of the problem, executional uncertainty can manifest itself in the form of machine breakdowns, early machine repairs or simply an operative not adhering to the specified schedule, thus invalidating existing scheduling decisions.

Reactivity and Explanation

An *O-agent* views its task as an instance of the constraint satisfaction problem (CSP). However, unlike most traditional forms of the CSP, an *O-agent* views its problem as a *dynamic* or *open* CSP. This is an important distinction in situations where it is necessary to produce solutions in near real-time, as is typically the case in factory scheduling problems. Whenever a fast response to change is required, it makes sense not to throw away the knowledge that was gathered when generating what is now a corrupt solution. In fact, it is extremely likely that there is a great deal of overlap between the previous problem, for which a solution exits, and the slightly modified problem which must now be solved. In order to take advantage of this fact, the *O-agent* employs truth maintenance system technology to allow it to decide which parts of its knowledge learned while solving earlier problems remains valid.

As well as supporting a reactive capability, the use of truth maintenance technology provides another major benefit in the form of an explanation facility. Whenever an *O-agent* is posed a problem which it cannot solve, it has the ability to identify which operations in its problem are the source of difficulty. In a reactive mode constraint conflicts first show themselves in the form of

difficult operational level problems, and an explanation facility for such problems is therefore a very useful feature when resolving conflicts. A more detailed account of the *O-agent* can be found in [Burke et al '89].

O-agent Parameters

The temporal constraints acting on the operations allocated to a resource along with the value held in the *constraint* slot of the resource serve to define the set of legal solutions to the associated *O-agent's* problem. Although DAS views the scheduling task essentially as one of satisfaction, soft constraints can be utilised to control the nature of local solutions. Each *O-agent* has a number of parameters which can be varied to strive for some local objective. The two most important parameters are held in the *select.op* and *select.value* slots of a resource. The value of the *select.op* slot specifies how to select for scheduling the next operation from the *unscheduled.ops* slot. The *select.value* parameter has a more direct impact on the nature of the schedule produced as it specifies how to select a start time from the set of legal values. There are a suite of values for this slot ranging from *earliest dispatch* through to *JIT*.

The precise nature of the temporal constraint acting between operations which must be satisfied by an *O-agent* is specified in the *constraint* slot of a resource. The value held in this slot identifies the function to be used by the *O-agent* when performing forward checking [Haralick et al '80] during search. Forward checking is a method of removing solutions from the search space which are inconsistent with the decisions made so far. Typically the function specified in this slot is encoded to ensure that no two operations are scheduled to be performed simultaneously on the same resource. This is specified by placing a value of *single.server* in the *constraint* slot of the resource. In situations where it is appropriate to perform operations simultaneously, at a large furnace for example, a value of *batch.server* is placed in the *constraint* slot. Other types of capacity constraint which can be coded as functions may also be passed to the *O-agent* via the *constraint* slot.

Tactical Level

The tactical level of DAS is viewed as being similar in nature to the level at which a shift leader or perhaps shop foreman operates in a factory management structure. Both the scheduling problem and resources at the tactical level are of a higher level of abstraction than those at the operational level. Each aggregation of resources defines a scheduling focal point, to which a *T-agent* is attached. A *T-agent* has a set of subordinate *O-agents* and is itself subordinate to the *S-agent*. Unlike *O-agents*, *T-agents* tend to communicate more up and down the hierarchy than across it, in keeping with the analogy drawn between *T-agents* and shift leaders.

Problem Definition

The role of a *T-agent* is essentially to co-ordinate activity within its sphere of influence, defined by the set of aggregate resources to which it is attached. It is responsible for delegating work to its subordinate *O-agents* and resolving conflict situations for them as and when the need arises. Discussion of the role of a *T-agent* within conflict resolution is deferred until chapter 4, where it is dealt with in the context of other mechanisms used to co-ordinate problem-solving effort. In much the same way as an *O-agent* is primarily concerned with satisfying temporal constraints, a *T-agent* is primarily concerned with satisfying technological constraints. Examples of technological constraints are the weight, temperature or perhaps a size limit of a resource.

A *T-agent's* problem is a dynamic constraint satisfaction problem. As discussed earlier, the technological constraints imposed on an operation are condensed into the value held in its *possible.resources* slot at the time it is created. Therefore, ignoring temporal considerations, the contents of the *possible.resources* slot defines the set of legal solutions to the technological constraints acting on the operations within a *T-agent's* problem. As is the case at the operational level, the *T-agent* does not deal with a closed, static problem. The problem of a *T-agent* can be changed by the introduction of work via the strategic level. The *T-agent's* problem also changes when it loses one of its subordinates through machine failure, machine maintenance or labour shortage. Although a *T-agent* is responsible for resolving conflicts which occur within its sphere

of influence, it is not always possible for it to do so. In such situations work may be pulled out of its problem, thus providing another source of change within its problem.

T-agent Parameters

The *T-agent* has a number of parameters, or soft constraints, which can be varied in order to control the nature of the solution generated. Resources have two slots which affect the solutions produced by its associated *T-agent*, namely the *look.a.head* and *select.strategy* slots. The value in the *look.a.head* slot determines the temporal horizon of a *T-agent*, while *select.strategy* determines how to select the next operation for delegation. Different types of resources may require different temporal horizons depending on their typical processing times. Varying circumstances on the shop-floor may also require different temporal horizons. For example, if a resource which normally has a temporal horizon of one week is known to be going down for preventive maintenance sometime in the next fortnight, it makes sense to extend the temporal horizon of its *T-agent* far enough into the future to allow it to take cognisance of the expected down-time. Operations may be selected for delegation on the basis of tardiness, user-priority, perceived system priority or any other heuristic which is considered appropriate at a particular resource. The chosen criteria is identified via the *select.strategy* slot.

Internal Representation

The *T-agent*, like other agents in the hierarchy, maintains its own internal representation of the world. This representation is concerned largely with the operations currently within the *T-agent's* problem and the resources to which it can delegate. In addition it has a *rest* attribute, a *T-assistant* and a *temporal-capacity* list. The *rest* attribute and *T-assistant* are discussed in the following section. The *temporal-capacity* list is used to maintain information about unavailable subordinate resources. It acts as a record of machines which are either unmanned or in need of repair, thus assisting the *T-agent* in delegating work.

The internal representation has three lists of operations. It has an *opstore*, a list of operations awaiting delegation, a *delegated-ops* list, a list of operations which have been delegated, and finally a *conf-ops* list, a list of operations currently involved in a constraint conflict. The *T-agent* selects operations for delegation from the *opstore* and adds operations to the *delegated-ops* list when they are delegated. The *delegated-ops* list is necessary to maintain an internal representation of all the operations in a *T-agent's* problem. The *conf-ops* list is used by a *T-agent* to prevent it from sending redundant pleas for assistance to the *S-agent*. Operations in this list have already been the subject of messages sent to the *S-agent* and are therefore awaiting some form of conflict resolution activity. They are removed in response to some action taken by the *S-agent* aimed at resolving the conflict in which they are involved.

Finally, the internal representation contains two lists of subordinate resources. The first, *resource-list*, is initialised to be a list of all the resources to which the associated *T-agent* may delegate. Whenever work is delegated to a particular resource, the resource is removed from the *resource-list*. This has the effect of inhibiting further delegations to it as only resources in the *resource-list* are considered for delegation. A resource is put back onto the *resource-list* when its associated *O-agent* signals that it has solved its current scheduling problem. The second list is the *res-in-conf* list, a list of resources which are currently in a state of conflict. This is used by the *T-agent* to determine which subproblems may have benefited from a recently completed action aimed at conflict resolution.

Method

Once initialised, a *T-agent* enters a three step loop. It reads and processes messages received since the last time it read its message buffer, attempts to delegate work to its subordinates and finally sleeps for a pre-specified period of time. The *rest* attribute of an agent's internal representation indicates for how long it should sleep. The messages which a *T-agent* might receive, and its response to these messages are discussed in the next section. Delegation, the second step of the loop, is discussed below.

When selecting work from its *opstore* for delegation, a *T-agent* gives precedence to *repair*, *unmanned* and *maintenance* operations. Every time it enters the delegation step of its loop, a *T-agent* delegates all such operations without considering the current state of the recipients. This policy is justified by the fact that the operations in question represent non-negotiable events which must be scheduled. If the recipients find themselves in difficulty as a result of this, they will request assistance from their superior *T-agent*.

It is more common for the *opstore* of a *T-agent* to contain only work operations. In this situation the *T-agent* performs its task in two stages. It first selects an operation to delegate, and then selects a resource on which to perform it. The selection of an operation for delegation must take cognisance of the "*look.a.head constraint*" in effect at a particular aggregate resource. This acts as a filter, removing all operations with temporal constraints which preclude them from being scheduled within the required temporal horizon. There are various methods of choosing an operation from those which successfully pass through the filter. The criteria to be used by a particular *T-agent* is specified in the *select.strategy* slot of its associated aggregate resource.

Having selected the next operation for delegation by whatever means, the *T-agent* must next select a subordinate resource to perform it. Each *T-agent* in the system has its own *T-assistant*. One of the functions performed by a *T-assistant* is to help its *T-agent* make delegation decisions. In response to a request from its *T-agent*, a *T-assistant* will return a list of possible resources for an operation. The returned list will be a subset of the list held in the *possible.resources* slot of the operation. Any resources found in the *possible.resources* slot but not in the list returned by the *T-assistant* have been omitted as a result of a learning process. That is, previous scheduling decisions, conflict situations and current beliefs lead the *T-assistant* to the conclusion that delegating the operation in question to one of the omitted resources will give rise to a conflict. Essentially, the *T-assistant* acts to prevent its *T-agent* from re-grouping work which has earlier been established to be a bad combination. In order to perform its task, a *T-assistant* must be kept informed of relevant events by its master *T-agent*. A detailed account of the *T-assistant* can be

found in [Burke et al '89].

The *T-agent* makes use of other information about its problem to improve the "quality" of its decision. The *T-agent* is aware of both the temporal constraints acting on the operation to be delegated and the periods during which its subordinate resources are unavailable. It uses this information when selecting a resource for the operation in an attempt to minimise backtracking. In this way the *T-agent* improves the quality of its local decision-making, where the measure of quality is search efficiency.

T-agent Messages

To achieve co-ordinated problem-solving activity from a distributed problem-solver requires communication. Message passing between scheduling agents provides DAS with one form of communication. Consequently, a *T-agent* must send and receive a variety of message types. Many of the messages a *T-agent* must deal with occur as a result of conflict situations. Discussion of such messages is deferred to chapter 4. In addition to messages sent by problem-solving agents, a constraint maintenance system (CMS) also generates message traffic. Discussion of the CMS, a mechanism for co-ordinating problem-solving effort, is also deferred to chapter 4. Some of the messages which a *T-agent* may send are listed below.

<add op_i>

A *T-agent* does not actually send this message to its subordinate agents directly. The act of delegating op_i to a particular resource causes the CMS to send it to the appropriate *O-agent*.

<delete op_i>

Like the <add op_i> message, this message is sent by the CMS as a consequence of a *T-agent* removing an operation from a particular resource.

Due to its position in the hierarchy a *T-agent* receives messages from both superior and subordinate agents. Additionally, it receives messages generated via the CMS. Subordinate *O-agents* have a range of messages which they can send depending on their current state. The *S-agent* also has a

number of messages at its disposal. Many of the messages the *S-agent* sends to its subordinate *T-agents* are necessary to allow the *T-agent* to keep its *T-assistant* adequately informed. Some of the message types received by a *T-agent* are listed below.

<complete>

Sent by an *O-agent* to notify its superior that it has solved its problem. The *T-agent* responds by putting the appropriate resource back into its *resource-list* and informing its *T-assistant* that the resource is now in a consistent state.

<new-op op_i>

Sent by the *S-agent* when it introduces a new operation into a *T-agent's* problem. The *T-agent* must add the operation to its *opstore* and inform its *T-assistant* that the new operation exists.

<end op_i>

The *T-agent* receives this message via the CMS in response to an operation being marked as complete. The *T-agent* removes op_i from its internal representation, including the *T-assistant*. If op_i is an *unmanned* or *repair* operation, the *T-agent* must also update its *temporal-capacity* list.

<modify op_i>

This message is generated by the CMS. The *T-agent* must update its internal representation of op_i.

Strategic Level

The strategic level of DAS corresponds to the factory manager within the management structure analogy and therefore there is only one scheduling focal point at this level. The scheduling focal point at the strategic level has an *S-agent* associated with it. The *S-agent* takes a very high level view of the scheduling problem, leaving the detailed scheduling activity to the lower levels. It communicates with its subordinate agents and receives messages informing it of the introduction of new work.

Role of the S-agent

The *S-agent* performs several functions. These include delegation of work to the tactical level, conflict resolution and synchronising particular aspects of problem-solving behaviour. The *S-agent* is responsible for resolving conflicts which cannot be dealt with at tactical level. It is also envisaged that it may perform the task of interpreting management objectives in order to fine tune the operation of the lower levels. Both its role as conflict resolver and problem-solving activity synchroniser fall within the scope of chapter 4, and are therefore not discussed here.

The task of delegating work to the tactical level is a simple one because each item of work can only be dealt with within one of the focal points at the tactical level. This is not a function of the architecture, but rather of the domain for which the exemplar system is being developed. The choice of which plan to select next represents the only discretion available to the *S-agent* when passing work to the tactical level. The criteria used to select a plan, for example the latest or most important, is specified via the *select.strategy* slot on the *strategic.unit*.

Internal Representation

The *S-agent* maintains its own internal representation of the world. It consists of two lists, a *planstore* and a *startedplans* list. The *planstore* contains a list of all the plans in the system which are awaiting delegation to the lower levels. That is, none of their operations have been delegated to the tactical level. As will be discussed further in chapter 4, it is not necessary to delegate all the operations from a process plan to the tactical level at the same time.

The *startedplans* list contains an entry for each plan which has one or more operations at the tactical or operational levels. When the *S-agent* delegates an operation from a plan for the first time it removes the plan from the *planstore* and creates an entry for it in the *startedplans* list. An entry in the *startedplans* list contains information identifying the corresponding plan and the operations within that plan which have been scheduled. This information is required by the *S-agent* when co-ordinating problem-solving effort.

S-agent Messages

Like other agents in the hierarchy, the *S-agent* is capable of sending and receiving messages. Only those which are not directly concerned with resolving conflict situations are listed below. The messages it can send are listed before those that it can receive.

<new-op op_i>

Sent by the *S-agent* to a *T-agent* informing the *T-agent* that op_i has been introduced into its problem.

<new-plan $plan_i$>

The *S-agent* receives this informing it that a new plan has been introduced into its problem.

<end op_i>

Received by the *S-agent* informing it that op_i has been completed. The *S-agent* assumes that op_i had been scheduled and updates the appropriate entry in the *startedplans* list accordingly.

3.3.2. Motivations For The DAS Architecture

The main features of the DAS architecture are that it is hierarchical, distributes problem-solving activity, and permits the various subproblems to be solved asynchronously. These features address a number of important issues including system response to executional uncertainty, the problem of conflicting scheduling objectives and combinatorial complexity. This section argues in favour of a hierarchical, distributed, asynchronous architecture and highlights how the issues identified above are catered for by such an architecture.

Benefits Of A Distributed Architecture

Manufacturing organisations are coming under increasing pressure to be more responsive to both market forces and the dynamics of the shop-floor. In order to become more responsive to

market forces, there is a need to shorten often long and bureaucratic decision-making chains, and at the same time move towards a JIT style of production. Distributing decision-making responsibility, while ensuring horizontal cooperation between problem-solvers, can assist with both these objectives. The distribution of responsibility serves to shorten decision-making chains, while decision-making based on horizontal cooperation is consistent with operating in a JIT manner.

The executional uncertainty present in the vast majority of manufacturing environments ensures that there is also a need to be responsive to events such as machine failures, scrapped orders and many other potential problems. A distributed architecture offers benefits over its centralised counterpart by permitting unanticipated events to be dealt with locally. Localising the impact of an unanticipated event allows a quick local response to the problem as well as improved global performance. The improvement in global performance results from the fact that decisions which are not affected by the event, and their associated problem-solvers are not involved in the reaction process.

Whether the scheduling task is performed by a human or a computer, conflicting scheduling objectives present a very real and difficult problem. The task of resolving conflicting objectives cannot be dealt with simply by allocating static priorities to the various objectives. Scheduling objectives and their relative priorities change with many things, among them market forces and executional uncertainty. DAS allows conflicting objectives to be distributed throughout the various nodes of the architecture, which can then negotiate a compromise solution. Details of how the various nodes are coerced into generating a favourable compromise solution are given in chapter 4. Distributing local objectives throughout the system ensures that they are represented far more explicitly than if they are embedded in a large algorithmic compromise. This makes it much easier to vary local objectives and measure the impact of these variations on the overall solution.

Another difficulty inherent in scheduling, i.e. problem complexity, also benefits from a distributed architecture. Scheduling is a very complex task requiring problem decomposition in the interests of problem tractability. DAS takes advantage of the necessity for problem

decomposition by permitting individual problem-solvers to operate independently on their more tractable local subproblems. While it is beneficial to allow independent problem-solving, it is also important to support communication among the various problem-solvers as and when appropriate. The provision for subproblem communication within DAS is important for two reasons. Firstly, because it is in the nature of the task being decomposed that there will be subproblem interactions, and secondly because unexpected events on the shop floor may give rise to additional subproblem interactions.

Benefits Of An Asynchronous Architecture

As mentioned earlier, there are many problem-solving architectures, including the blackboard architecture, which deal with distributed problem-solving, but not distributed processing. Essentially, the various problem-solving nodes in a blackboard-like system contribute to the global hypothesis in a synchronised manner, whereas in DAS they contribute in an asynchronous manner. The asynchronous nature of the DAS architecture permits concurrent processing.

In terms of pure computational considerations, an asynchronous architecture offers a more inviting solution than a synchronised architecture. With an appropriate problem decomposition and concurrent processing, an asynchronous architecture offers the possibility of reducing the time required to generate a solution. Perhaps of more significance, concurrency provides the opportunity to expand scheduler processing power in accordance with any shop floor expansion. In this way, at least in principle, the time required to generate a schedule can be maintained at a relatively constant value independently of fluctuations in plant size. Obviously, as the plant size and the number of subproblems increase, the number of subproblem interactions also increase. This may result in an increase in the time required to generate a solution.

As well as reducing computational expense in terms of execution time, a distributed asynchronous architecture can reduce the financial cost of the computational facilities required by a scheduling system. The combined cost of several low cost, standard devices plus a communication medium to inter-connect them is often lower than the cost of a centralised device powerful enough

to deal with the scheduling task as a whole. Furthermore, if a centralised solution is to attempt to provide a reactive capability, it too will need communication links to terminals on the shop floor, adding additional expense to the already high cost of a centralised device. Regardless of the size of the initial capital outlay, the majority of organisations are more likely to embrace familiar technology than high cost unfamiliar technology. Also, a high cost device is likely to have correspondingly high training and maintenance costs, whereas there is a distinct possibility that the existing staff of an organisation will have a degree of familiarity with the low cost device selected for an implementation, thus reducing both the time and cost required for training.

Automated scheduling systems are charged with the responsibility of deciding how best to utilise the available resources of an organisation. The resources it deals with, machinery, products and labour, represent a very large capital investment on the part of that organisation. It is not surprising then, that system reliability is an important consideration when developing an automated scheduler. In terms of system reliability, a distributed asynchronous architecture is again superior to a centralised system. Whenever a centralised system suffers a hardware failure, the scheduler it supports suffers a complete failure. In a genuinely distributed system supported by an asynchronous architecture, a hardware failure does not result in total failure, but rather a graceful degradation of service. It may be possible to transfer the tasks from a failed processor to other processors, thereby maintaining a complete scheduling service at a somewhat reduced rate of response. Alternatively, a human scheduler could be introduced into the system to play the role of the failed processor without ever informing the other problem-solvers of the failure.

When dealing with shop floor dynamics, concurrent processing supported by an asynchronous architecture offers another major benefit. In manufacturing environments it is not uncommon for several unexpected events to occur at once. By permitting concurrency, an asynchronous architecture can deal with multiple local problems simultaneously. This is obviously a much better situation than being faced with the task of prioritising the current set of problems.

Benefits Of A Hierarchical Architecture

It would have been entirely possible to implement a DAS-like scheduling system within a single-level, distributed, asynchronous architecture. However, the decision to impose a hierarchical structure upon what was already a distributed and asynchronous architecture has several justifications. The most obvious justification is that the decision-making process within production scheduling is naturally hierarchical. Having a hierarchical structure provides benefits when addressing the issues of problem decomposition, focusing problem-solving attention, applying appropriate scheduling methods and regulating communication.

A hierarchical model allows the scheduling task to be decomposed in several ways. Firstly, it allows the task itself to be viewed at varying levels of abstraction from a very detailed level up to a fairly aggregate view of the task. The temporal horizon considered by the various levels in the hierarchy may also be varied, ranging from a very short-term view at the lowest level to a long-term view at the highest. Executional uncertainty argues against maintaining the detailed schedules found at the lowest level of the hierarchy over a long period of time. As well as complicating the reaction process, the probability of a detailed schedule remaining workable diminishes as the time to execution increases. However, to avoid resource contention, it is necessary to maintain a predictive view over a longer temporal horizon. This can be achieved within the higher levels of the hierarchy in which a less detailed view of the schedule is maintained. Finally, within manufacturing facilities there is usually some natural decomposition of the problem across major resources. Resouces can be represented by a single node at the lowest level, and aggregated as appropriate at the higher levels.

By allowing the problem to be viewed at various levels of abstraction, over varying temporal horizons at particular areas in the factory, a hierarchical architecture allows problem-solving effort to be focused at the most appropriate level along all three of these dimensions. In a reactive mode, DAS attempts to localise its problems by keeping them as low as possible in the hierarchy and only involving other areas of the factory if necessary. When acting predictively, DAS prefers

to deal at as high a level as possible, avoiding detailed scheduling decisions. Having scheduling agents distributed throughout a hierarchical structure satisfies the requirements of both these desires.

As noted above, a hierarchical structure allows the scheduling problem to be viewed at multiple levels of abstraction. The type of scheduling decision to be made, and hence the appropriate scheduling methodology is dependent on the level within the hierarchy at which that decision is being made. Distributing agents throughout the hierarchy allows the most appropriate scheduling method to be applied to any given problem. Within DAS there is one type of agent for each level in the hierarchy. The behaviour of each individual agent can be further tailored to adopt a scheduling methodology appropriate to the particular area of the factory it is dealing with. To achieve a similar effect within a centralised architecture would result in considerable complexity in the control heuristics necessary to invoke the appropriate scheduling methodology.

Due to its distributed nature, the DAS architecture relies heavily on communication amongst its subproblems for success. While communication is necessary, it is also important to regulate it in such a way that prevents irrelevant communications swamping the system. The hierarchical structure of DAS provides a mechanism for regulating communication between levels and thus blocking a significant amount of irrelevant communication.

3.3.3. Evaluation of Architecture

This section has presented and argued in favour of a hierarchical, distributed, asynchronous scheduling architecture. This architecture is now reviewed with respect to recent research in the area of distributed scheduling. The major obstacle to be overcome within a distributed, asynchronous scheduler is the requirement to co-ordinate problem-solving activity of the various components of the system in a globally consistent manner. One aspect of an architecture which has most bearing on its ability to co-ordinate problem-solving activity is its provision for inter-node communication.

As discussed in chapter 2, there has not been a great deal of research in the area of distributed scheduling, with the majority of existing work being based largely on the contract net metaphor. For a great many scheduling domains, and heavy manufacturing in particular, this is not a particularly good metaphor for co-ordinating problem-solving activity. Although it is usual for there to be clearly defined manager/contractor relationships in the domain, as required by the contract net, the communication mechanism offered by the contract net is not appropriate for two reasons. Firstly, it is dependent on the existence of a sufficient number of similar resources to generate competition for work in order to avoid the situation in which a single contractor monopolises the schedule in a manner which satisfies its own local objectives at the expense of global objectives. The second weakness of the contract net communication mechanism in manufacturing domains is more of an efficiency concern than a functionality problem. The strength of the contract net lies in its ability to manage highly volatile systems. While many aspects of the manufacturing domain are highly volatile, features such as factory layout and machine capabilities are fairly static. The pure contract net metaphor makes poor use of this low volatility knowledge by adopting a broadcast strategy to communication. While the introduction of mechanisms such as focused addressing to cater for this weakness do help, they appear to be an unnecessary overhead.

A more suitable approach to communication in a distributed asynchronous scheduler operating within a heavy manufacturing domain is offered by *message-passing* [Hewitt '77]. Activity in message-passing systems occurs in response to messages sent by other agents in the system. Ideally, agents in a message-passing system should know which other agents in the system are potentially affected by local events, but need not know what such an event means to affected agents. In manufacturing domains, the areas of a schedule potentially affected by local events are generally known from fairly static relations within the factory model and the most appropriate response to such an event is often best formulated by the receiving agent. These characteristics make message-passing a suitable means of communication in a distributed asynchronous scheduling system.

Message-passing has been used as a communication mechanism within a distributed scheduling system before DAS. Smith and Hynynen [Smith et al '87] present a scheduling framework based on cooperative problem-solving, distributed according to a hierarchical factory model and co-ordinated via message-passing. The DAS architecture and the modelling framework proposed in [Smith et al '87] have several things in common, the most obvious being that they both perform problem distribution based on a hierarchical structure. Also common to both is the motivation for problem distribution and the method of communication between nodes in the hierarchy. However, as will be discussed shortly, [Smith et al '87] suggests the contract net as a suitable method of communication for the task with which DAS deals.

Despite the common framework, there are significant differences between the model proposed by Smith and Hynynen and the DAS architecture. The most notable difference concerns the node structures found within DAS and the Smith and Hynynen model. Unlike DAS, which has a different node structure at each level in the hierarchy, the Smith and Hynynen model proposes that the same node structure be used throughout the hierarchy. In fact it suggests that each node should consist of an OPIS-like scheduler. This is acknowledged within [Smith et al '87] as not being a suitable node structure at the level of real-time control of actual machines and material flows, ie: the level at which DAS operates. The tasks performed at the various levels of the hierarchy, within the scheduling portion of the larger production management hierarchy addressed by Smith and Hynynen, are sufficiently diverse to justify specialised node structures.

Smith and Hynynen suggest that communication based on the contract net metaphor is suitable for the real-time scheduling task performed by DAS. For the reasons cited above, message-passing is considered to be a superior mechanism even at this task level within heavy manufacturing domains. Perhaps in domains such as flexible manufacturing systems or large job-shops in which machines can perform many functions and there are many machines which can perform each task, the contract net metaphor may be more appropriate.

Representation of the global hypothesis is another area in which DAS and the model proposed by Smith and Hynynen differ. Within DAS, each node has its own internal representation of its portion of the schedule and additionally there is an external representation of the global schedule. Within the Smith and Hynynen model there appears to be only one version of the schedule distributed throughout the hierarchy. The significance of the concept of a node having its own internal view of its world and an external view shared by other nodes will be made apparent in chapter 4.

CHAPTER 4

Managing Problem-Solving Effort In DAS

4.1. Introduction

The scheduling architecture introduced in chapter 3 offers the possibility, at least in principle, of focusing problem-solving effort in a highly effective manner. It permits multiple focal points of scheduling activity, each of which can have varying degrees of abstraction and temporal horizon, to be considered simultaneously. However, if any benefit is to be gained from the flexibility afforded by the DAS architecture, it is necessary to supplement it with mechanisms capable of managing problem-solving effort in a globally consistent manner. In a distributed problem-solver, this requires co-ordination as well as focusing of problem-solving effort. This chapter presents the mechanisms which manage problem-solving effort in DAS.

In order to co-ordinate problem-solving effort effectively, it is necessary to maintain the current global hypothesis and make it accessible to all problem-solving agents. The global hypothesis is a data structure used to represent the current problem-solving state. It is implicitly defined by operations, resources, plans and the constraints attached to them as described in chapter 3. Both the scheduling decisions of agents and events from the scheduling environment considered to be significant are recorded in the global hypothesis. It is not uncommon for constraints (scheduling decisions and significant events) added in this way to have implications for the global hypothesis which are not fully represented within the constraint itself. A constraint maintenance system (CMS) is employed to perform the inferencing necessary to make such implications explicit. As well as maintaining a consistent view of the global hypothesis, the CMS also provides a means of highlighting conflicts between problem-solving nodes in the system. In this way it also acts to focus problem-solving effort.

An agent accesses relevant parts of the global hypothesis in order to obtain an external view of the world. As noted earlier, each agent in DAS also maintains an internal representation of its world. Initially, an agent creates its internal representation by making a direct copy of the appropriate parts of the global hypothesis. After initialisation an agent maintains its internal representation entirely via message passing. The fact that each agent maintains its own internal representation allows a situation to arise where one agent can be out of step with other agents in the system. If used in a controlled manner, this facility can be a useful tool when focusing problem-solving effort.

Executional uncertainty, infeasible scheduling problems and incompatible scheduling decisions are just a few of the factors which combine to ensure that conflicts will arise in the global hypothesis. It is not sufficient merely to detect the existence of a conflict, but to be able to identify its cause and have mechanisms available to resolve it. Scheduling agents at each level in the DAS hierarchy have the ability to both identify the potential cause of certain classes of conflict and the mechanisms necessary to resolve the identified conflicts. The class of conflict which can be dealt with by any particular agent is dependent on its level within the hierarchy. Conflict resolution often requires the co-ordination of problem-solving effort.

Due to the predominantly reactive nature of DAS, co-ordination of problem-solving effort occurs mainly in response to conflict recognition. DAS reacts to all conflicts in a manner which is independent of whether the conflict occurred as a result of executional uncertainty or in the process of schedule synthesis. Despite placing the emphasis on a *reactive* capability, DAS does provide some mechanisms which permit *predictive* co-ordination of effort. That is, it is possible to perform a pre-analysis of the problem in order to enforce an appropriate co-ordination of problem-solving effort. Predictive co-ordination of effort is considered to be appropriate only where it can offer a significant improvement in performance for a relatively low cost. Such mechanisms are only useful in dynamic environments when they are supported by *reactive* mechanisms capable of resolving the situation which occurs on failure of one or more of the underlying assumptions of the

predictive phase.

This chapter first presents information considered necessary to allow a full understanding of the CMS. This includes a discussion of appropriate techniques, their complexity, and a review of related work. The CMS is then presented, followed by a detailed account of the conflict resolution mechanisms available. The chapter concludes with a discussion of the facilities provided for *predictive* coordination of effort.

4.2. Background Information

This section presents information considered appropriate for a full understanding of the CMS described in section 4.3. Temporal constraints, the concept of network consistency and the constraint propagation techniques employed to achieve it are discussed. The complexity of label inferencing, the particular form of constraint propagation employed by the CMS, is considered before a brief history of related work is presented.

4.2.1. Temporal Constraints

The constraints with which the CMS must deal are temporal in nature. Many of the constraints found in scheduling problems are temporal in nature, for example the release date and due date of an order. It is not only work items as a whole which are subject to temporal constraints, since the processes which must be performed to produce a work item (represented by operations in DAS) also have associated temporal constraints in the form of expected duration and required temporal precedences. Temporal constraints are by far the most volatile in a scheduling problem and play a significant role when deciding how to satisfy other non-temporal constraints. The CMS is therefore primarily concerned with maintaining consistency amongst temporal constraints, and consequently amongst scheduling agents.

It is possible to identify two distinct classes of temporal constraint, namely symbolic and numeric. Symbolic representations, such as those presented in [Allen '83], emphasise temporal relations between events. For example, the temporal precedence constraint which specifies that

during the manufacture of aluminium plate, the annealing process must precede the ultrasonic scanning process is a symbolic temporal constraint. Generally speaking, symbolic representations do not handle numeric data such as event start times and durations efficiently, whereas numeric representations do. On the other hand, numeric representations, which put the emphasis on the description of events, are unable to explicitly represent relationships between events. The list *(5 10)*, representing the fact that an operation is restricted to start somewhere between time 5 and time 10, is an example of a numeric temporal constraint. Both classes of temporal constraint are integrated within DAS, a situation which allows the numeric representation to act as a refinement of the relations expressed by the symbolic representation.

4.2.2. Constraints and Consistency

The CMS is mainly concerned with maintaining the temporal constraints acting on the operations of a process plan in a consistent manner. The process plan of an order can be modelled as a constraint network comprised of nodes and arcs, or in terms of *events* and *sets of possible occurrences* (SOPOs) as proposed in [Rit '86]. When representing a process plan as a constraint network, nodes of the network correspond to operations of the plan and arcs correspond to the temporal precedence relations acting between operations. Each node has an associated domain defining the legal values of that node. Rit presents a model for the propagation of temporal constraints in which the basic objects are events which are linked by symbolic temporal relations. Within this model events are characterised by SOPOs, where an occurrence is defined as the interval during which an event happens. Obviously, when representing a process plan within this framework, operations become events, temporal precedence constraints become symbolic temporal relations and SOPOs act as a refinement of the symbolic temporal precedence relations. Both models have been introduced here because it will be convenient to switch between the two when reviewing related work later on.

Using Rit's terminology, the task of maintaining the temporal constraints on the operations within a plan in a consistent manner is referred to as the *constrained occurrences problem.* He

defines it as the task of modifying the SOPOs used to characterise events in such a way as to make them compatible with the symbolic relations acting as constraints between the events. In constraint network terminology, the forward inferencing which achieves the desired level of consistency within a network is referred to as *assimilation*. Whether it is viewed as an example of the constrained occurrences problem or assimilation, the task of forward inferencing is usually performed using constraint propagation techniques. The degree of consistency achieved by constraint propagation within a constraint network is determined by the correspondence between the values allowed by the domain of each node and the constraints acting between nodes. Mackworth [Mackworth '77] identifies three levels of consistency, namely node, arc and path consistency. They are described below.

(i) Node Consistency

Node i is node consistent iff for any value x in D_i, the domain of i, $P_i(x)$, the unary predicate on node i holds. For example, if a variable has a constraint (unary predicate) which requires it to take only values greater than ten, then that variable is node consistent if its domain contains no values less than or equal to ten. A network is node consistent if all the variables in the network are node consistent.

(ii) Arc Consistency

Arc (i,j) is arc consistent iff for any value x in D_i such that $P_i(x)$ holds, there is a value y in D_j such that both $P_j(y)$ and $P_{ij}(x,y)$ hold. $P_{ij}(x,y)$ holds if the binary predicate P_{ij} is satisfied when node i has the value x and node j has the value y. A network is arc consistent if all the arcs in the network are arc consistent. An example of a consistent arc is shown in figure 4.1.

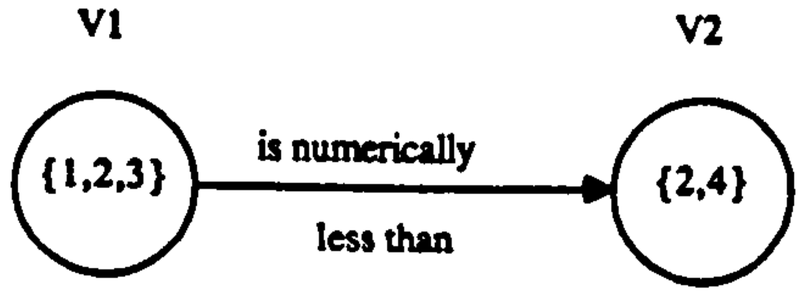

Figure 4.1

However, if the domain of V_1 included a value of 4 or greater, this arc would not be arc

consistent as there would be no value of V_2 which could satisfy the predicate P_{12} if V_1 was assigned this value.

(iii) Path Consistency

A path of length m through the nodes $(i_0,i_1,...,i_m)$ is path consistent iff for any values x in D_{i_0} and y in D_{i_m} such that $P_{i_0}(x)$ and $P_{i_m}(y)$ and $P_{i_0i_m}(x,y)$ hold, there is a sequence of values z_1 in D_{i_1},..., z_{m-1} in $D_{i_{m-1}}$ such that:

(a) $P_{i_1}(z_1)$ and ... $P_{i_{m-1}}(z_{m-1})$

(b) $P_{i_0i_1}(x,z_1)$ and $P_{i_1i_2}(z_1,z_2)$ and ... $P_{i_{m-1}i_m}(z_{m-1},y)$.

Montanari [Montanari '74] has shown that if every path of length two in a network is path consistent then the network is path consistent. Mackworth presents the following example, shown in figure 4.2 of a path inconsistency. The path is of length $m = 2$, and the binary predicate denotes strict lexicographic ordering.

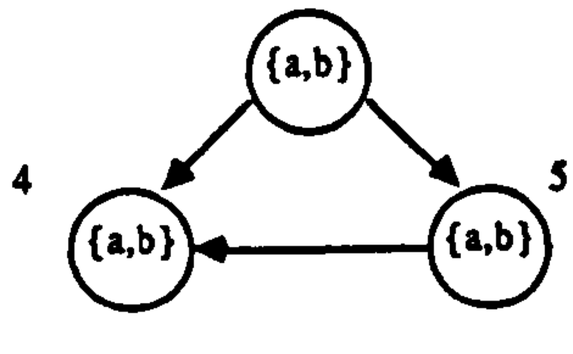

Figure 4.2

He states that a path inconsistency appears on path *3-5-4*. Condition (a) is irrelevant because there are no unary predicates in the network, while condition (b) requires that two binary predicates $P_{3,5}$ and $P_{5,4}$ hold. Additionally, $P_{3,4}$ must hold with the same sequence of values for variables V_3, V_5 and V_4. $P_{3,4}$ can be satisfied by giving V_3 a value of a and V_4 a value of b, and $P_{3,5}$ can be satisfied by again giving V_3 a value of a and V_5 a value of b. However, there is no value in D_5 which can simultaneously satisfy $P_{3,5}$ and $P_{5,4}$, hence the path is path inconsistent. Path consistency is preferable to arc consistency because it detects inconsistencies which would otherwise go undetected. For example, if the portion of the network shown in figure 4.2 was made arc consistent, it would look like the network shown in figure 4.3. In this network all the variables have at least one possible value, thus

the inconsistency has gone undetected. If the network had been made path consistent, the inconsistency would be highlighted by the fact that V_4 would have a null domain.

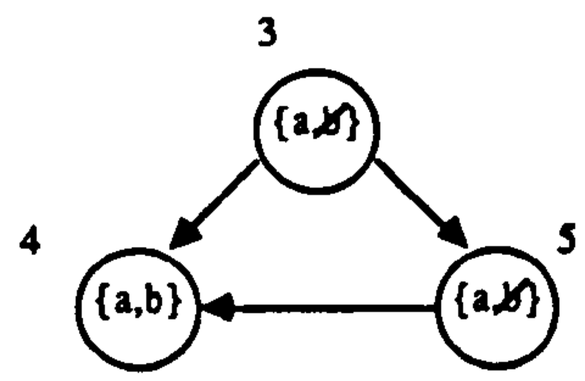

Figure 4.3

4.2.3. Constraint Propagation

Constraint propagation techniques date back to the Waltz filtering algorithm [Waltz '72]. Many subsequent constraint propagation systems, including those used for temporal constraint propagation, are based on this algorithm. Constraint propagation algorithms operate by first deducing the implications of a change in the area of the network which is local to the induced change. If these deductions have implications for other nodes in the network, further deductions are performed and recorded. In this way the consequences of a change gradually spreads throughout the network. A Waltz-like constraint propagation algorithm, AC-2 from [Mackworth '77], is shown in figure 4.4.

```
Procedure AC-2
For each node in the network do
   make it node consistent
   Q ← all arcs incident on node

   While Q not empty do
      While Q not empty do
         get first arc (k,m) from Q
         If (k,m) needs modified
         Then
            modify it
            Q' ← all arcs incident on k
         Q ← Q'
         Q' ← empty
         end
      end
end
end.
```

Figure 4.4

Constraint propagation has many characteristics which make it a popular method of forward inferencing. Firstly, it is based on a simple control structure which makes it easy to code, extend

and analyse. Secondly, it degrades well under time limitations because interrupting the process does not invalidate inferences already established. It is amenable to parallel implementation and well suited to incremental systems in which not all constraints are known at the outset. Finally, the concept of propagating gradually through a network until quiescence, is consistent with locality assumptions often found in AI. For example, within physical reasoning programs (eg. [Davis '83]) it is often assumed that physical effects propagate across connections between components. Another example is found in the scheduling domain where it is assumed that temporal effects propagate to adjacent operations in a process plan.

Having introduced the notion of constraint propagation, it is appropriate to consider some classification of constraint propagation systems. Davis [Davis '87] distinguishes six categories of constraint propagation, using the type of information propagated as a basis for discrimination. The interested reader is referred to [Davis '87] for a detailed account of his six categories which are *constraint inference*, *label inference*, *value inference*, *expression inference*, *relaxation* and *relaxation labeling*. At a grosser level than this it is possible to identify two classes of constraint propagation, namely those which deal with temporal constraints and those which deal with non-temporal constraints. Not surprisingly, temporal constraint propagation can be further classified as being either symbolic or numeric. Symbolic systems combine temporal relations using a temporal logic, such as that given in [Allen '81], to infer additional relationships and to detect contradictions within a network. They are suitable for tasks in which the relative occurrence of events in time is important. Numeric systems allow for the deduction of new temporal equalities and inequalities, and are better suited to tasks which involve placing events at an absolute position on the time line rather than a relative one. Therefore, symbolic systems are appropriate in applications such as planning, whereas numeric systems are better suited to tasks such as scheduling. Using the categorisation identified by Davis, symbolic temporal propagation is described as *constraint inference*, while numeric temporal propagation is referred to as *label inference*. The CMS performs label inferencing.

As the CMS performs label inferencing, it is appropriate to consider this form of constraint propagation in more detail and to identify its main characteristics. [Davis '87] identifies several characteristics including constraint language, query language, query answering language, node type and label type. He goes on to conclude that the label language and constraint language of a label inferencing system are the key characteristics for categorisation. Label languages tend to use either signs or intervals within their representation. Sign labeling is often sufficient in applications where the reasoning is qualitative rather than quantitative and there is an interest in the sign rather than the magnitude of the quantities being reasoned over. ENVISION [de Kleer et al '84] is an example of a system which uses sign labeling to perform qualitative reasoning about the behaviour of physical systems over time. Interval labeling must be used when sign labeling is too coarse a measurement, or when it is difficult to identify a zero point around which signs should change. In interval labeling, intervals may be open, closed or half-open, with or without an ability to represent non-contiguous intervals. They are sufficiently flexible to permit representation of both an approximate value or a degree of uncertainty and are inexpensive to represent. The choice of which interval representation is most suitable is application-dependent. As is the case with label languages, there are many possible constraint languages ranging from simple unary predicates through systems of equations and up to transcendental equations. Within the scheduling domain, unary predicates and binary predicates capable of expressing temporal precedence relations are the most common.

As can be inferred from the characteristics of label inferencing systems listed above, a constraint network used to support assimilation may also be used to support query answering. The majority of systems are designed such that most of the inferencing takes place during assimilation, thereby allowing queries to be answered quickly from the quiescent state of the network. Obviously, information used to answer queries must pass from the constraints to the queries via the node labelings, ie: the domains of the nodes. Unfortunately, node labels provide only a "narrow bandwidth communication channel" which makes it difficult to ensure that information is not lost during query answering. Consequently, label inferencing systems are rarely complete, leaving

most system designers to strive for the more achievable goal of ensuring that the assimilation and query answering processes are each separately complete. An example demonstrating a loss of information during query answering is given below.

Initial Conditions
A is initially labelled as {1,2,3}
B is initially labelled as {2,3,4}
With constraint A = B

After Assimilation
A is labelled as {2,3}
B is labelled as {2,3}

Figure 4.5

If the system is now presented with the query *Is A equal to B ?*, the best answer which can be derived from the labels is maybe. The correct answer is yes because the constraint $A = B$ excludes the possibility of A being equal to 2 while B is 3 and vice-versa. There are a number of ways round this problem, including using nodes which correspond to complex terms, though this increases the complexity of both assimilation and query answering. An alternative approach, adopted by McDermott in an early version of SPAM [McDermott '80], is to use conservative labels such that all selections of values from the labels satisfy the constraints. This approach is also problematic because unlike label inferencing it is not deductively sound, ie: it permits inferences which are not warranted by the constraints.

4.2.4. The Complexity of Label Inferencing

As is the case in most areas of artificial intelligence, combinatorial explosion is a major concern for the designers of label inferencing systems, particularly when the label language is quantitative rather than qualitative. The efficiency of algorithms which perform label inferencing to ensure consistency within a constraint network has therefore received considerable attention. Unfortunately, it has been shown that the task of determining consistency within a temporal constraint network is an instance of the consistent labeling problem [Tsang '87], a task which is known to be NP-hard [Mackworth '77].

Vilain and Kautz [Vilain et al '86] are interested in constraint propagation algorithms for temporal reasoning, and consider the computational aspects of both point-based and interval-based representations. They show that achieving path consistency in a network represented within the interval algebra of Allen [Allen '83] is NP-hard. On the other hand, an algorithm is given which determines path consistency in the point based algebra in $O(n^3)$ time and $O(n^2)$ space, where n is the number of points about which assertions have been made. However, the computational benefits afforded by the time point algebra have a cost in terms of its expressive power. A point-based representation is not capable of representing many of the relationships which are desirable in a scheduling context. Most significantly, it is not able to represent any relation in which there is ambiguity concerning the relative position of whole intervals rather than points on the time line (eg. the *not.during* relation). [Tsang '87] points out that it is the presence of disjunctive temporal relations, which cannot be represented within the point-based representation, and not the chosen representation which makes consistency checking NP-hard.

Given the exponential nature of determining consistency within the interval algebra, Vilain and Kautz suggest several practical solutions to this problem. One option would be to limit the number of intervals dealt with to something of the order of a dozen. With such a small problem size, the asymptotically exponential performance need not be noticeably poor. Alternatively, where it is not possible to limit the problem size sufficiently, it is possible to trade completeness for polynomial performance. For example, Allen [Allen '83] presents a polynomial time constraint propagation algorithm for the interval algebra which is sound but not complete, ie: it will not generate inferences not warranted by the constraints of the network but it may not remove all inconsistencies. It may also be acceptable to use an algorithm which is exponential in asymptotic complexity, but which generaly performs much better that this. [Valdes-Perez '87] presents just such an algorithm which is sound and complete for the full interval algebra. In practice, the improved performance of the Valdes-Perez algorithm is achieved through heuristic pruning and clever backtracking.

Not all applications require the full interval algebra, a fact which it may be possible to exploit when developing a consistency checking algorithm. Rit [Rit '86] states that if the temporal constraints being propagated over can be represented by windows, and disjunctive relations are eliminated by enumerating the different terms, the task of determining consistency can be solved using an arc consistency algorithm. Polynomial algorithms, such as Mackworth's AC-3 algorithm [Mackworth '77] can be applied to determine consistency. A detailed account of the complexity of several polynomial consistency checking algorithms can be found in [Mackworth et al '85].

4.2.5. Related Work

As is true of all the mechanisms discussed in this chapter, the primary motivation for having a CMS is to improve search efficiency. This, allied with its more specific objectives of consistency maintenance, conflict detection and an ability to cater for non-monotonic inferencing, could easily lead to the conclusion that the truth maintenance system technology of Doyle [Doyle '79] and de Kleer [de Kleer '86] is appropriate. However, two characteristics of the inferencing performed by the CMS make it unwise to maintain inferences within a computationally expensive truth maintenance system. Bear in mind that the fundamental question addressed by truth maintenance systems can be summarised loosely as "*Which of the inferences made so far remain valid after certain assumptions have altered ?*", an instantiation of the frame problem [McCarthy et al '69]. This is not a major issue for the CMS because firstly, the inferences generated are relatively inexpensive to generate and secondly the possibility of an inference recurring within a given problem is fairly remote. Therefore, although it may appear to be functionally similar to a TMS at a high level, the CMS does not resemble a TMS internally. The CMS owes considerably more to the constraint propagation literature than to the truth maintenance system literature.

Temporal constraint propagation dates back to the CPM algorithm [Johnson et al '74], the forerunner of PERT. Despite the fact that the majority of work discussed here is from the scheduling literature, it should be noted that temporal constraint propagation is not limited to

scheduling applications. In fact, the term *window*, commonly used to refer to the domain of a variable within interval label inferencing, was introduced by Vere in DEVISER [Vere '83], a planning system which attempts to take cognisance of time when generating plans. DEVISER is similar in nature to NOAH [Sacerdoti '77] and NONLIN [Tate '76] in that it is a non-linear planner, but has additional functionality which permits it to consider the start times and durations of the activities to be planned. Within DEVISER, windows are used to provide an upper and lower bound on the time when an activity may occur. A window is represented as a triple of the following format: *(Earliest-start-time Ideal Latest-start-time)* in which *Ideal* is considered to be the most desirable start time from the range allowed by the other two fields. This representation is not capable of representing non-contiguous intervals and therefore cannot support disjunctive constraints. Constraint propagation, or window compression as Vere refers to it, is invoked during plan generation whenever nodes are linked, expanded or ordered to resolve a conflict. (The reader is assumed to be familiar with terminology used by Tate [Tate '76] regarding plan generation). Any one of these events modifies the existing plan, thus generating a new constrained occurrences problem, which is then solved using window compression.

The constraint propagation module of ISIS, reported in [Smith '83], like DEVISER also assumes contiguous start time intervals for activities. It employs a critical path method to compute the earliest and latest bounds of an activity. As in DAS, activities can be represented at various levels of abstraction, therefore requiring an ability to propagate temporal constraints up and down the abstraction hierarchy. Within ISIS, composite operations exist which represent exclusive alternatives, an ordered sequence of activities or operations that can be performed in parallel. The constraint propagation module of ISIS propagates through all three types of composite operation as well as linear plans.

TMM [Dean '85] provides another example of the use of constraint propagation outside of the scheduling domain. The main concern of TMM is to keep track of how the facts that it knows about become true or false over time. It takes as its input a number of time tokens, assertions,

rules and constraints. Time tokens denote instants of interest, assertions concern facts or events, rules describe relationships among events and facts, and constraints identify the times at which the various time tokens take place. Temporal constraints are represented as contiguous intervals which bound the time difference between two tokens. The constraint propagation mechanism within TMM performs both label and constraint inferencing. Although the ability to perform both label and constraint inferencing may be desirable, the computational expense of such an approach is likely to place practical limitations on the problems which can be addressed. Furthermore, applications such as scheduling do not derive a great deal of benefit from the power of such a mechanism. Scheduling applications tend to deal with static pre-calculated process plans, and therefore do not require constraint inferencing.

Until now, the constraint propagation systems which have been discussed have represented temporal constraints as a contiguous interval. This representation is not flexible enough to permit the inclusion of disjunctive constraints within the constraint network. Disjunctive constraints, such as OpA *not.during* OpB, are common in manufacturing scheduling domains and necessitate a representation of temporal constraints which permits non-contiguous intervals. As discussed earlier, [Rit '86] represents temporal constraints as SOPOs and states that they can be considered as a disjunctive clause in which each part of the clause is a single occurrence. There is no restriction preventing SOPOs from representing non-contiguous intervals, and in fact Rit goes on to present a constraint propagation algorithm capable of propagating disjunctive constraints. The algorithm is a variation of the Waltz filtering algorithm [Waltz '72].

As discussed in chapter 2, OPIS evolved out of experience gained during the ISIS project. As OPIS increased in sophistication to support its opportunistic and reactive characteristics, it became necessary to enhance the existing constraint propagation system. Within OPIS, the computation and maintenance of operation time bounds is accomplished by an object oriented propagation process. Each propagation message contains the temporal constraint itself, the origins of the constraint and the required level of precision. As in ISIS, operations are represented at

various levels of abstraction. The *required level of precision* field of a propagation message indicates how far down the hierarchy it is necessary to propagate this message. The origin of a constraint in the propagation message is included to aid in conflict resolution. Despite these enhancements to the constraint propagation module of ISIS, the more advanced version employed in OPIS continues to cater only for temporal constraints which can be represented as a contiguous interval.

Other work in the area of temporal constraint propagation, which may become relevant to DAS if it were to be modified for other domains, concerns methods for controlling constraint propagation and the propagation of local preferences. Much of the work which investigates techniques for controlling constraint propagation has been carried out by Collinot and LePape. [Collinot et al '87] discusses techniques for dynamically varying the amount of computational effort spent during constraint propagation, while the constraint propagation system described in [LePape et al '87] allows the amount of propagation to be varied through the specification of levels of precision. In the latter case, the variation is defined with respect to a predetermined set of parameters. Collinot and LePape argue that the amount of effort expended during propagation should be proportional to the amount of interaction that propagation is expected to detect or avoid, and that the knowledge required to regulate this mechanism is available.

Until now, the propagation systems discussed in this section all attempt to achieve a degree of consistency between the constraints acting on a network and the values permitted by the domains of the nodes of the network. [Sadeh et al '88] propose a constraint propagation mechanism which attempts to give some indication of local scheduling preferences as well as excluding values which are inconsistent with the constraints acting on the network. That is, preference propagation strives not only for admissibility but also for optimality by reflecting preference interactions locally. It is proposed that this may be achieved by representing temporal constraints as probability distribution functions rather than by intervals. The notion of representing local preferences is not new, for example, the *Ideal* field of a window in DEVISER is used to represent a local preference.

However, unlike the approach proposed by Sadeh and Fox, the window compression algorithm employed in DEVISER takes no account of the *Ideal* field which is statically fixed.

4.3. The Constraint Maintenance System of DAS

This section presents the constraint maintenance system of DAS. The role of the CMS and the requirements imposed upon it by DAS are considered. Following this, a detailed account of the algorithm and constraint representations used by the CMS is given. A complexity analysis of the CMS concludes the section.

The CMS has three main features to recommend it. Perhaps most importantly for its application within DAS is its ability to cater for the retraction of constraints both correctly and efficiently. Secondly, it permits propagation over constraint networks representing non-linear process plans in time polynomial with the number of operations in the plan. Finally, it performs its task of maintaining consistency in a manner which allows it to assist in the task of conflict resolution.

4.3.1. Role of The CMS

The constraint maintenance system (CMS) of DAS performs two major functions, both of which are necessary in order to manage problem-solving effort effectively. Firstly, it must ensure that the various agents in the system share a common view of the world. Secondly, it is responsible for highlighting constraint conflicts which occur either during schedule synthesis or as a result of executional uncertainty.

Within DAS, the subset of the current set of problem constraints considered by an agent constitutes its view of the world. The current set of problem constraints are a function of the initial problem specification, the decisions of scheduling agents and significant events from the scheduling environment. In order for agents to co-operate effectively, and thus not waste problem-solving effort, it is important to ensure that each agent has a view of the world which is consistent with the views held by the other agents in the system. The CMS performs this role,

allowing each agent to adopt a local view of its problem while retaining a reasonable guarantee of global consistency. To achieve this, the CMS makes explicit the implicit effects of constraints added to the system. That is, it infers constraints which occur as a consequence of the addition of new constraints and adds them to the current set of problem constraints. The *reactive* and *opportunistic* characteristics of DAS make it essential that agents are kept informed of the impact of the currently active constraints on their local subproblems.

As well as providing problem-solving agents with a consistent view of the world, maintaining the current set of problem constraints and their consequences presents an opportunity to detect conflict. By making the consequences of all known constraints explicit, the CMS is able to highlight incompatible constraints. Constraints need not necessarily be completely incompatible for the CMS to provide useful information; it also helps in the detection of highly contended areas of the schedule.

There are a number of reasons why conflicts occur in a schedule, of which incompatible scheduling decisions and poor problem specifications are only two. For example, a scheduling problem which includes an order having a due date earlier than its release date obviously contains conflicting constraints. Conflicts may also arise as a result of the non-monotonic inferencing performed by the CMS. Inferencing is non-monotonic if the addition of new knowledge about the world can decrease the set of inferences which can be drawn. For example, the act of reporting the actual completion time of an operation may cause a previously calculated estimate to be invalidated. The inferencing performed by the CMS is necessarily non-monotonic in order to cater for the use of defaults and the highly dynamic nature of the scheduling environment. Both default reasoning and executional uncertainty permit the possibility that additional knowledge about the world can cause existing inferences to be invalidated. The interested reader is referred to [Ramsay '88] for a good introduction to non-monotonic reasoning.

As discussed in chapter 2, mechanisms which attempt to focus problem-solving effort within a heuristic search can do so either by reducing the number of nodes to be visited or by determining

a suitable order in which they are to be visited. The CMS incorporates both techniques to achieve this objective. Maintaining consistency within the global hypothesis serves to prune the search space of a great many illegal solutions, while conflict recognition assists in the ordering of the search.

4.3.2. Unary Constraints

The constraint networks to be propagated over contain both unary and binary constraints. Binary constraints are required to represent temporal precedence relations between operations, while unary constraints, described below, are used to represent various attributes of individual operations. Change is always introduced into the network via a unary constraint. A change in the value of a unary constraint results in a modification to the domain of the affected operation, thus initiating propagation between operations over binary constraints.

The unary constraints associated with an operation which may initiate temporal constraint propagation fall into one of four categories. The four categories, *scheduling decisions*, *preference constraints*, *real world events* and *predictively generated constraints* are listed below in figure 4.6 along with the specific constraints which are members of each category. Each of the unary constraints listed is implemented as a slot within the frame representing an operation.

CATEGORY	INSTANCE
Scheduling Decisions	Start Time, Resource
Preference Constraints	Due Date
Real World Events	Release Date, Actual End Time
Predictive Constraints	Release Guess

Figure 4.6

Of course, other preference constraints, real world events and non-negotiable constraints are catered for by DAS. However as they do not constitute sources of temporal constraint propagation they are not listed in figure 4.6. Examples of such constraints are particular machine preference for an operation, machine failure, scrapped orders and technological constraints. Each of the unary constraints listed is discussed below.

An operational-level scheduling decision, that of giving an operation a start time, introduces a new unary constraint to the problem and may initiate temporal constraint propagation. The introduction of such a unary constraint has the same effect as giving the operation in question a single point domain. Although the domain of the operation is not actually modified, the effects of giving it a start time must be propagated over the binary constraints representing temporal precedence relations.

A tactical-level scheduling decision, that of allocating an operation to a particular resource, may also initiate temporal constraint propagation. Allocating an operation to a particular resource may cause its duration to be modified, thus invalidating any earlier propagation based on the old value. This can occur because operations are initialised to have their worst-case duration prior to being allocated to a particular resource. If an operation is subsequently delegated to a resource on which it has a shorter duration, it is necessary to undo the earlier propagation and re-propagate using the new, more accurate value for the duration of the operation.

Giving an order a due date has the effect of constraining each operation within the process plan of that order to complete before a certain date. The required completion date of each operation is dependent on the duration of other operations in the plan. Similarly, giving an order an expected release date allows an "earliest start" constraint to be calculated for each operation in the process plan of that order. However, the task of accurately predicting when a work item will be released onto the shop floor is not trivial. To accommodate this, the actual release date of an order is accepted as a refinement of the expected release date. As is the case when the duration of an operation is refined, any earlier propagation based on the estimated value is undone, and a new propagation is initiated using the correct release date.

Estimating the release date of an order is not the only source of uncertainty catered for by the constraint propagation system, since the duration of an operation is also subject to variation. The variation being discussed here concerns the actual execution time of an operation on a particular resource rather than a variation which is dependent on the resource used to perform an

operation. To cope with this, DAS accepts a completion message indicating the actual completion time of an operation as a refinement of the expected completion time. On receipt of a completion message, propagation based on an estimated completion time, calculated from the scheduled start time of the operation plus its expected duration, is undone and a new round of propagation based on the reported value is initiated.

4.3.3. Propagation Messages

The CMS is mainly concerned with maintaining the global hypothesis in a consistent manner. The global hypothesis is used by an agent as its external representation of the world. As discussed in chapter 3, an agent maintains its internal representation via message passing. Many of the events which cause the CMS to update the global hypothesis also require an agent to update its internal representation. To cater for this, the CMS generates messages in accordance with the various sources of constraint propagation. It is important to note that communication via message passing in DAS operates on the basis of addressed messaging rather than a broadcast arrangement. That is, messages are only sent to agents who have an interest in the message in question. In all cases it is left to the receiving agent to determine the appropriate action to take on receipt of a message.

The most common message generated by the CMS is the *<modify op_i priority>* message. It is sent to an agent whenever one of the operations in its problem has a modification to the value in its *legal.starts* slot. The CMS sends the message to the agent identified by the value held in the *resource* slot of the affected operation. The value of the priority field of a *<modify op_i priority>* message is determined by the source of the modification. A message sent as a result of a non-negotiable event such as the late completion of an operation, is given a priority of -1. This is interpreted by the receiving agent as a message which cannot be ignored. A more interesting use of the priority field occurs when the message is sent as a result of a negotiable event such as an operational level scheduling decision. The messages generated as a result of an operation being

given a start time take on a priority equal to that of the operation which has just been scheduled. Operation priority is discussed further in section 4.4.1.

An <add op_i> message is sent to an agent to inform it that op_i has been introduced to its problem. Similarly, a <delete op_i > message is sent to an agent to inform it that op_i is no longer part of its problem. The CMS sends these messages in response to a change to the value in the *resource* slot of an operation. It sends a <delete op_i > to the agent identified by the old value in the slot, and an <add op_i > to the agent identified by the new value in the slot.

As will be discussed further in section 4.4.4, one of the conflict resolution mechanisms available to the *S-agent* is the synchronisation of decision-making through process plans. To facilitate this, the CMS sends <*dec-made* op_i> and <*dec-cancelled* op_i> messages to the *S-agent*. A <*dec-made* op_i> message is sent in response to op_i being given a start time. This informs the *S-agent* that a scheduling decision has been made on op_i. A <*dec-cancelled* op_i > message is sent whenever a start time is removed from op_i, indicating the cancellation of an earlier decision.

4.3.4. Requirements of Constraint Propagation

The two main requirements imposed upon the design of the CMS concern its ability to deal with constraint retractions and the non-linearity of process plans. Both requirements have implications for the representation and propagation of temporal constraints. The need to cater for constraint retractions is a consequence of the dynamic and stochastic nature of its intended operating environment coupled with the inherently asynchronous nature of DAS itself. The need to cater for non-linear process plans is simply another manifestation of the opportunistic ethos of DAS.

The earlier discussion of potential sources of constraint propagation should provide sufficient supporting evidence for the statement that constraint retraction is a major issue within DAS. Additionally, by permitting *O-agents* to become temporarily out of step with one and other, the asynchronous nature of DAS allows incompatible scheduling decisions to be made, thus providing

another source of constraint retraction. The type of constraint retraction being discussed here concerns only unary constraints rather than the binary constraints of the constraint network being propagated over. It concerns constraints such as start times, release dates and due dates rather than temporal relations such as *before*, *after* and *not.during*. However, the temporal constraint representation and method of propagation employed by DAS does permit efficient addition and retraction of temporal relations. This was an important feature in an early version of DAS in which composite operations were explicitly created, modified and destroyed. The act of modifying a composite operation changes the constraint network being used to represent process plan(s) involved in the composite operation.

Obviously, the degree to which constraint retraction is expected to occur within DAS makes the efficiency of this activity a major concern. However, it is perhaps more important to ensure that the network is returned to the state it would have been in had the constraint never been imposed, a feature apparently not considered by traditional constraint propagation algorithms. Most existing algorithms, such as those analysed by Mackworth and Freuder [Mackworth et al '85], are founded on the premise that the merit of an algorithm is inversely proportional to the number of redundant variable domain revisions attempted during propagation. However, when incremental constraint retraction is necessary, some apparently redundant revisions are necessary to ensure that constraints and their propagated effects are retracted correctly. This is demonstrated by the example of figure 4.7 ((a) to (d)) in which OpA is *before* OpB which is *before* OpC in the same linear process plan.

Within figure 4.7 a grey box associated with an operation indicates the domain of that operation, ie: the interval during which it is legal to schedule this operation to start. The domains shown are all contiguous intervals because the process plan being used in the example is a linear plan. A white box within a grey box indicates the duration of the associated operation, while its position is of no significance as the operation has not yet been scheduled. Upon allocation of a start time, the white box of an operation is changed to a black box and is positioned to reflect the

allocated start time.

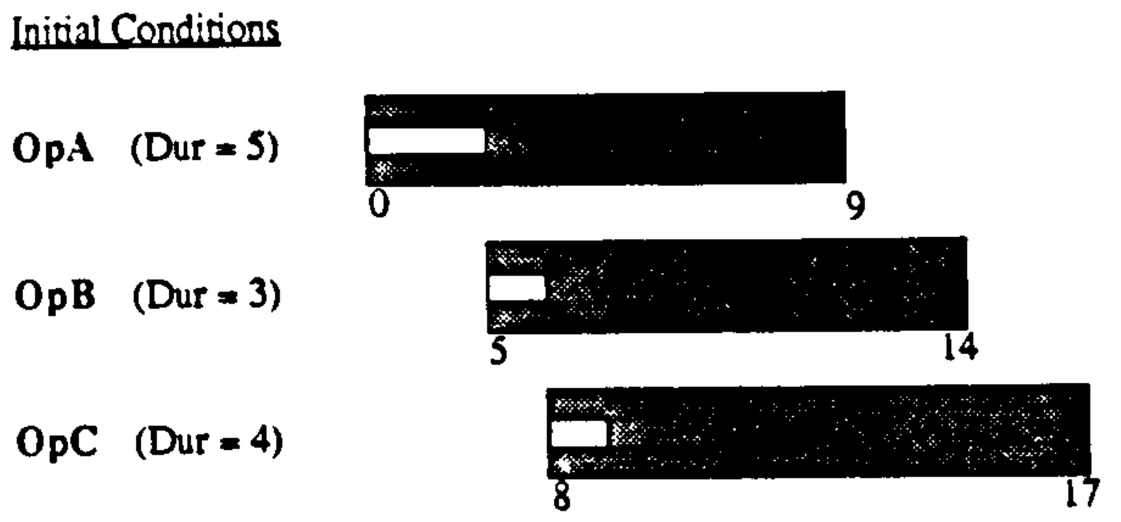

Figure 4.7 (a)

Figure 4.7 (a) shows the initial temporal constraints acting on the operations of the process plan. They are calculated from the release date, due date and temporal precedence relations of the process plan. Figure 4.7 (b) shows the temporal constraints acting on the network after OpB has been allocated a start time of 10. The significant change to note here is that the earliest time at which it is legal to schedule OpC to start is no longer 8, but 13.

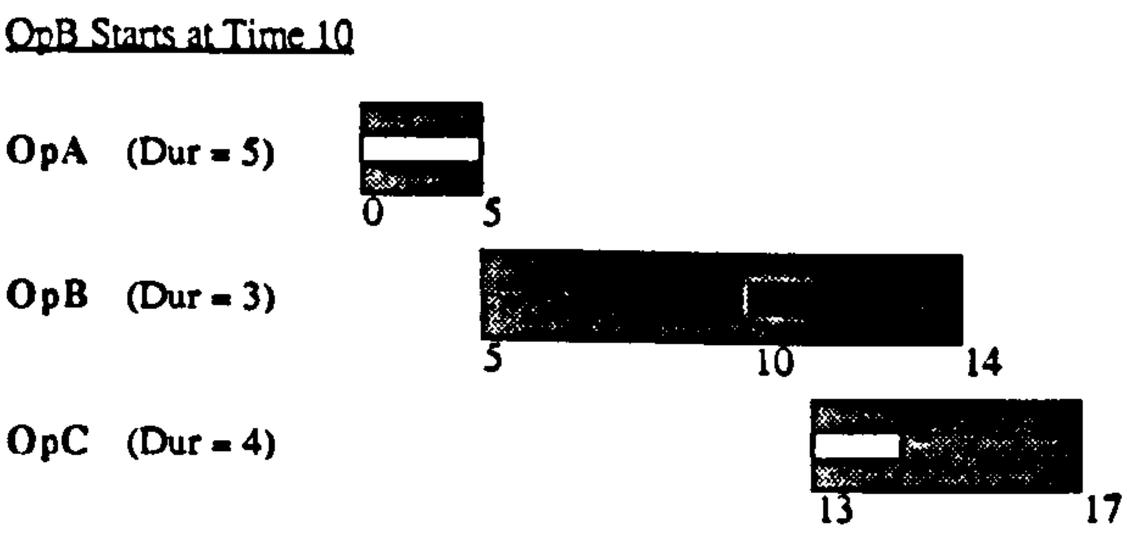

Figure 4.7 (b)

Figure 4.7 (c) shows the temporal constraints acting on the network after a second scheduling decision has been made, that of starting OpA at time 2. Although this pushes the earliest temporal constraint on OpB from 5 to 7, it has no affect on the earliest temporal constraint acting on OpC. This is because the decision to start OpB at time 10 is more constraining on OpC than the decision to start OpA at time 2. It is therefore redundant, at least for the time being, to propagate the consequences of the scheduling decision on OpA to OpC.

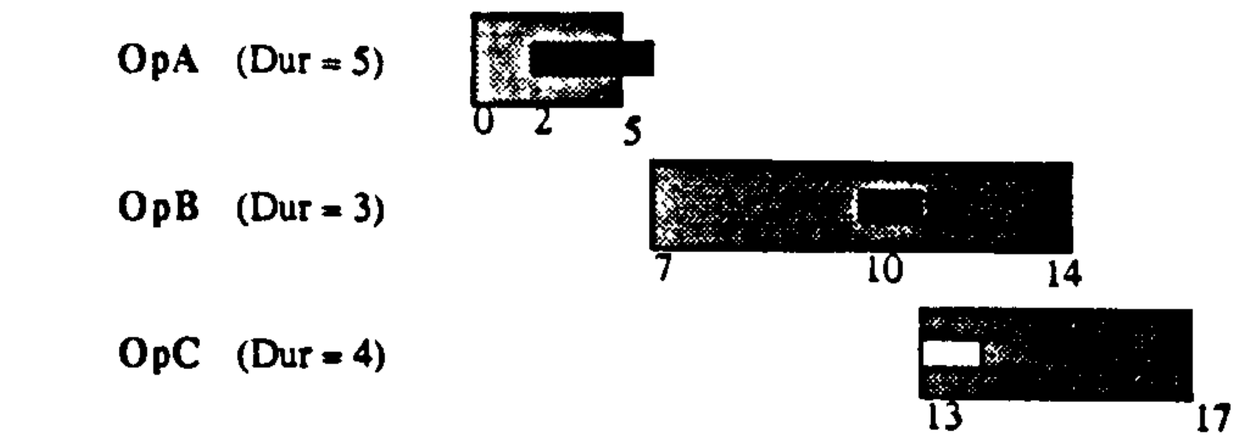

Figure 4.7 (c)

However, if the decision to start OpB at time 10 is subsequently withdrawn, the temporal constraint acting on OpC is dependent on the scheduling decision on OpA. There are several possible approaches to dealing with the retraction of the scheduling decision on OpB, the simplest being not to propagate the affects of the retraction at all. This leaves OpC over constrained by requiring it to start after time 13. Alternatively, the earliest start constraint of OpC could be removed entirely leaving OpC under constrained. Neither of these options are particularly attractive as they both leave the constraint network in a state in which the domains of its nodes are inconsistent with the constraints acting on the network.

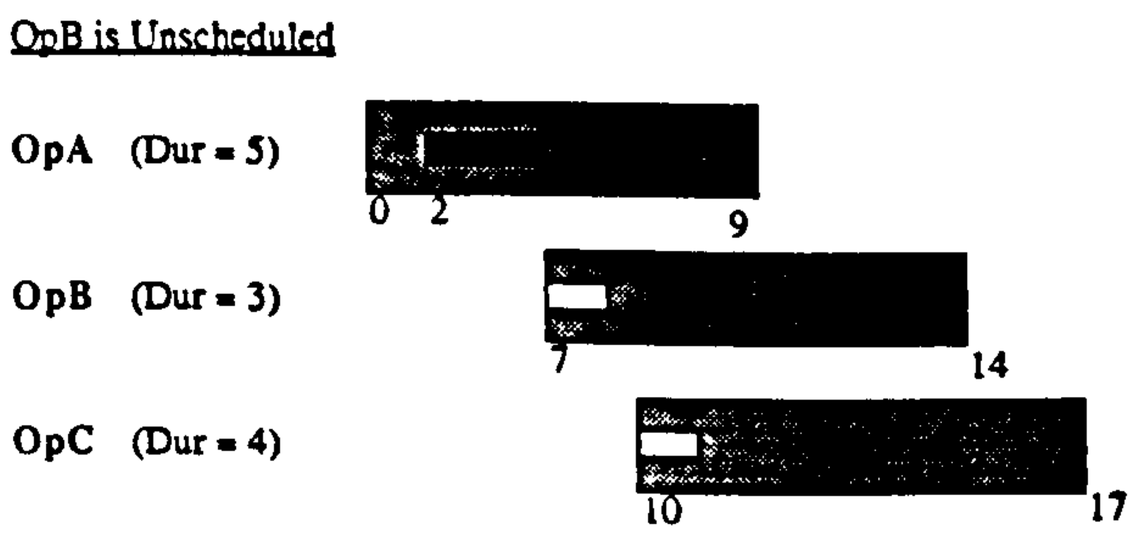

Figure 4.7 (d)

Ideally, the temporal constraints acting on the network should be transformed to look like those shown in figure 4.7 (d), which represent accurately the consequences of current scheduling decisions. Again there are two possible approaches to achieving this. Either the temporal constraints acting on the network can be completely recalculated or, an incremental calculation

based on the old state of the network and the recent change can be performed. In a system in which constraint retraction is extremely common, the incremental approach is likely to be more efficient. On the other hand, when constraint retraction is an unusual occurrence, the additional overhead required to permit incremental recalculation may be more expensive than the cost of occasional complete recalculation. The level of constraint retraction required within DAS certainly favours an incremental approach.

In order to calculate incrementally the consequences of a constraint retraction it is necessary to perform what are apparently redundant propagations. This is highlighted in the example of figure 4.7 in which it is necessary to propagate a constraint precluding OpC from starting before time 10 as a consequence of the scheduling decision on OpA. At the time of propagation this is redundant as OpC is already constrained to start no earlier than time 13. The temporal constraint representation described shortly is designed make this an efficient process.

As discussed in chapter 3, DAS caters for plans which include the *not.during* (*before* OR *after*) relation in order to permit maximum opportunism. Essentially, this permits the scheduler to decide on a particular linearisation of a plan based on a detailed knowledge of existing shop floor work load and current scheduling decisions. This has implications for constraint propagation because, as [Collinot et al '87] points out, " *as soon as disjunctive constraints are considered, the problem of determining whether a given set of constraints is consistent is* NP-hard ". The temporal constraint representation employed within DAS is designed with this in mind.

4.3.5. Constraint Representation

It was stated at the outset of this section that the CMS of DAS had three main features to recommend it. These were noted as being an ability to propagate over networks containing disjunctive constraints in time polynomial with the number of nodes in the network, an ability to deal with constraint retraction efficiently and an ability to assist in conflict resolution. The data structures developed to represent temporal constraints are of prime importance in achieving all

three of these objectives. The first two, polynomial time performance and efficient constraint retraction, had the most influence on the representation selected for temporal constraints.

As noted earlier, determining consistency within a constraint network containing disjunctive constraints is known to be NP-hard [Collinot et al '87]. It would appear from this that the objective of polynomial performance from a consistency algorithm operating on a network containing the *not.during* relation is at best optimistic. However, as [Rit '86] points out, if the constraints involved can be represented by windows and disjunctive relations enumerated, consistency can be achieved using an arc consistency algorithm. If it were possible to cast the problem in this way, the objective of polynomial performance becomes possible as there are several known polynomial algorithms (eg. AC-1, AC-2 and AC-3 from [Mackworth '77]) for achieving arc consistency. The requirement to enumerate disjunctive constraints to their component parts does not pose a problem in the case of the *not.during* relation which expands to *before* or *after*. Neither does the requirement that temporal constraints be represented as a window or a disjunction of windows. This second requirement is perhaps somewhat misleading. It is the ordered nature of the domain rather than that it be represented as a window that is of significance.

The notion that a temporal constraint may be composed of its various component constraints is not peculiar to a representation of disjunctive constraints. In fact, the effective temporal constraint acting on the majority of operations in the system is almost certainly composed of several component constraints. Consider an operation which is restricted to start sometime between time 5 and time 10. It may be the subject of a temporal constraint composed of a release time constraint and a due date constraint. The desire to perform constraint retraction in an efficient manner requires that the various component constraints be tagged with their source constraint. In the event of a constraint retraction all that is required is the removal of the appropriately tagged component constraints.

So far, an argument has been presented in favour of a representation which is made up of multiple windows, each of which is tagged with its source and represents a contiguous interval.

However, when an *O-agent* needs to know when it can schedule an operation, it is not interested in the various sources of the temporal constraint, nor does it require the constraint to represent a contiguous interval. For this reason, it was decided to maintain a dual representation for temporal constraints. Externally, temporal constraints are represented as a list of integer pairs, where each integer pair represents a closed contiguous interval. By permitting any number of pairs within the list, any level of non-contiguity can be represented. An example is shown in figure 4.8.

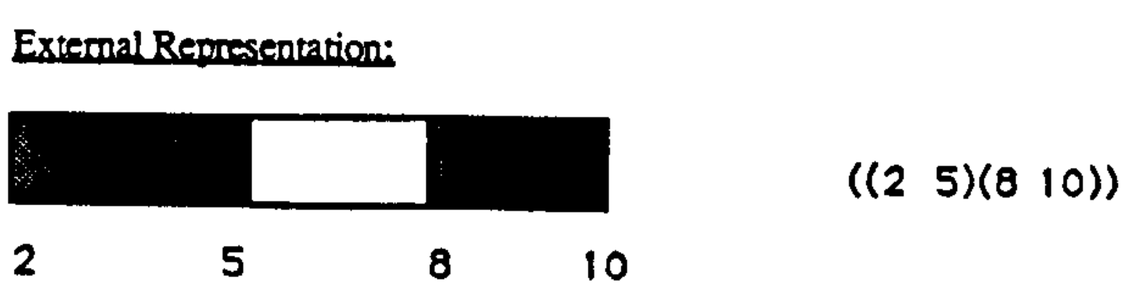

Figure 4.8

Internally, a temporal constraint is represented as a collection of component constraints which are combined to generate the external representation. A component constraint is of the form *(Source (X Y))* in which *Source* identifies the source of the component constraint and *(X Y)* the interval of time allowed by the constraint. The effective constraint on an operation is calculated by intersecting the intervals allowed by all component constraints. The source may be a scheduling decision on another operation, the due date of a lot or indeed any of the sources listed in figure 4.6.

In addition to permitting two of the major objectives of the CMS, the temporal constraint representation described above offers a number of other benefits, the most important being in conflict resolution and computational efficiency. The internal representation of a temporal constraint assists in conflict resolution by identifying the various unary constraints which play a

part in a conflict. In addition to identifying candidate constraints for relaxation, the internal representation makes it simple to determine the local implications of particular constraint relaxations.

Another important feature of this dual representation of temporal constraints is its contribution to computational efficiency in a highly uncertain environment. Rather than calculating a new external representation every time its internal representation is modified, it is more efficient to calculate the external representation on demand. That is, calculate the internal representation during assimilation and only generate an up-to-date external representation in response to a query. In addition to the computational effort required to propagate constituent constraints, the act of calculating a value for the external representation of a constraint incurs significant computational expense. The highly volatile nature of temporal constraints makes it extremely likely that many of the states that a temporal constraint passes through in the course of schedule synthesis will never actually be accessed. In particular the temporal constraints on operations which are not yet at the operational level are likely to take on many values which are never accessed by a scheduling agent. For example, consider the case of the last operation to be scheduled of a five operation process plan. As will be discussed in section 4.4.4, the last operation may remain at the strategic level until after scheduling decisions have been made on the other four. Therefore, the temporal constraint on the last operation will take on at least four values which will never be accessed by a scheduling agent. Additionally, late or early completions of any of the other four operations may add to this figure.

4.3.6. Constraint Propagation Algorithm

The DAS constraint propagation algorithm differs from traditional algorithms in that it intentionally performs apparently redundant propagations. As discussed earlier, this is required to deal with incremental constraint retraction. The algorithm used in the initial implementation of the CMS, in which each constraint is propagated throughout the complete plan regardless of its

current implications, is shown below in figure 4.9.

```
Procedure prop-1(op comp-const)
1. allowed-interval ← nil
   action ← add or retract as specified by comp-const
2. opx ← before(op)
   if action = add
   then allowed-interval ← calc-before(comp-const opx)
   update-legal-starts(opx action source allowed-interval)
3. opx ← after(op)
   if action = add
   then allowed-interval ← calc-after(comp-const opx)
   update-legal-starts(opx action source allowed-interval)
4. loop for opx in not-during(op)
   do begin
      if action = add
      then allowed-interval ← calc-before(comp-const opx)
      update-legal-starts(opx action source allowed-interval)
      if action = add
      then allowed-interval ← calc-after(comp-const opx)
      update-legal-starts(opx action source allowed-interval)
      end
```

Figure 4.9

The procedure PROP-1 is given as arguments an object representing an operation, *Op*, and a component constraint. *Op* is the operation receiving the component constraint. The component constraint is either of the form described in the preceding section or *(Retraction, Source)*. The latter form specifies the retraction of a particular source of constraint propagation. The source of propagation specified in *comp-const* will affect the temporal domains *(legal.starts)* of related operations. In step 2 the interval allowed in the temporal domain of the operation related to *Op* by the *before* relation is calculated. The temporal domain of the operation is then updated, via a call to function *update–legal–starts*. An *active value* is attached to the *legal.starts* slot of each operation. When the *legal.starts* slot of an operation is updated a call is made to the attached procedure, in this case PROP-1. In this way recursion is invoked via access-oriented programming techniques. If the component constraint specifies a retraction, *update–legal–starts* retracts the temporal constraint tagged by *source*. Step 3 deals with the *after* relation in a similar way as step 2 deals with the *before* relation. In step 4 the *not.during* relation is elaborated as the disjunction *before* OR *after*, ie: step 4 resembles the concatenation of steps 2 and 3.

The fact that this algorithm propagates constraints regardless of their current impact on the constraint network gives rise to some concern about its applicability to very large constraint

networks. Within the exemplar site for DAS, the size of the constraint networks being propagated over is sufficiently small for this not to be a major concern. However, a potentially more efficient version of PROP-1 may be encoded (PROP-2) in which propagation halts after the first unaffected node is reached. A node is said to be unaffected by propagation if the change induced in the internal representation of its temporal constraint has no effect on its external representation. PROP-2 is modified such that the call *update-legal-starts* makes a call to PROP-2 if, and only if, the *legal.starts* of the operation changes. Although this is likely to reduce the number of constraint propagations performed, it is not necessarily more efficient. In fact, it introduces two additional sources of computation. The first is a consequence of the fact that it is now necessary to test the effect of a propagation at each stage in the process. The second occurs in the event of a constraint retraction from a node. It is now necessary to determine if there are any propagation processes which halted at this node earlier. If there are any, a further test must be applied to each process in turn to determine if they should be re-initiated. When choosing between the two algorithms it is necessary to consider the size of the constraint network, the cost of testing versus the expected number of redundant propagations and the expected number of constraint retractions.

It is worth noting a few of the more significant implementation details of the CMS for the sake of comparison with other constraint propagation algorithms. The first thing to notice is that algorithms PROP-1 and PROP-2 are both recursive rather than iterative. Traditional constraint propagation algorithms (eg. the algorithm shown in figure 4.4) are iterative in nature and concentrate on maintaining a list of relations which are waiting to be updated. This list grows and shrinks as it is processed during propagation. Within DAS, object-oriented and access-oriented programming techniques have been utilised to provide a simple implementation of PROP-1. Within this implementation it is not necessary to explicitly maintain a list of pending relations. Representing each node in the network as an object and attaching a demon (active value) to the slot on an object representing its domain is sufficient to ensure that all relations requiring modification are modified. The simplicity of this technique is extremely valuable when implementing, testing and extending the constraint propagation mechanism.

4.3.7. Complexity Analysis

It has been stated that the CMS of DAS can determine consistency of a constraint network containing disjunctive constraints in polynomial time. This section presents an analysis of the performance of the PROP-1 algorithm to show that it does achieve consistency in time polynomial with the number of nodes in the network. Like Mackworth and Freuder in [Mackworth et al '85], the time unit used as a measure of complexity is the application of a unary or binary predicate.

The CMS, following the advice of Rit [Rit '86], enumerates disjunctive constraints and determines consistency using an arc consistency algorithm. Although the statement made by Rit is strictly speaking correct it may be somewhat misleading. It does not describe a technique for determining consistency in general constraint networks containing disjunctions in polynomial time. It simply means that consistency can be achieved by invoking a polynomial arc consistency algorithm on the various linear portions of the network generated by the enumeration. The problem remains combinatorial because the number of enumerations required to remove disjunctive constraints remains exponential. Therefore, the following analysis of PROP-1 can show polynomial performance for no more than a restricted class of constraint networks. As the networks being discussed increase in complexity so too does the measure of their computational complexity.

Traditionally, network consistency algorithms have been used to pre-process constraint networks before backtrack search is applied. However, it is intended that the constraint propagation algorithm of the CMS be used alternately with variable instantiation, ie: a scheduling decision is made followed by constraint propagation followed by a scheduling decision and so on. The importance of the distinction here is that any analysis of PROP-1 should take cognisance of its ability to determine consistency within a constraint network incrementally, rather than always starting anew. Most constraint propagation algorithms start by placing all the relations of the network into a queue of relations waiting to be checked. In the worst case scenario, this does not introduce an inefficiency as all relations of the network must be updated. However, in any other

situation this approach necessarily leads to the redundant testing of unaffected relations. Because PROP-1 is intended to be used incrementally, it initially tests only those relations involving the node immediately affected by the unary constraint which has been modified, thus initiating constraint propagation.

Before embarking on an analysis of PROP-1, it is appropriate to highlight two significant points. The first is a feature of the domain (temporal constraint propagation) in which the CMS operates and the second an important implementation detail. Within the CMS it is possible to take advantage of the strict ordering of values in a node's domains to achieve a figure of one predicate application per relation propagated over. For example, if nodes A and B are related by the *before* relation and A is given an end time of 10, it is not necessary to test each value in the domain of B individually to determine whether it should be removed or not. It is sufficient to remove all the values to the left of 10 in the domain of B. This holds for constraint networks representing non-linear process plans if the various intervals representing a disjunctive constraint are ordered numerically.

The second point, an implementation detail, concerns how constraints are propagated over operations related by the *not.during* relation. Consider the portion of a constraint network shown in figure 4.10, in which *OpB* and *OpC* are related by the *not.during* relation and *OpD* is related to both by the *after* relation.

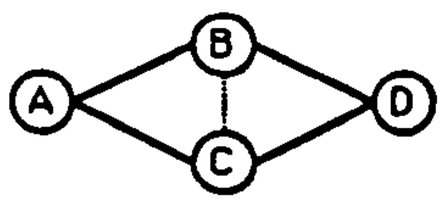

Figure 4.10

The act of giving *OpA* a start time causes a new constraint to propagate directly to *OpB* and to *OpC*. Both constraints will further propagate to *OpD*. However, the two constraints which arrive at *OpD* are equal, and therefore only the first to arrive will propagate further. They are equal because the calculation which determines the constraint to be added takes cognisance of the

duration of other operations related by the *not.during* relation. However, it should be noted that if propagation is initiated at a disjunction, redundant component constraints will be propagated. For example, if *OpB* is given a start time, the component constraint which is propagated directly to *OpA* may not be equal to the one which arrives via *OpC*. Consequently, both component constraints will be propagated further.

In the diagrams which follow, a dotted line denotes the disjunctive *not.during* relation. The integer beside a node indicates the number of component constraints added (predicate applications) to that node in response to the constraint propagation initiated at the node marked with an arrow. The total number of component constraints added throughout the network is used as a measure of computational complexity, *C*. For each case considered *C* is described in terms of *n*, the number of nodes in the network.

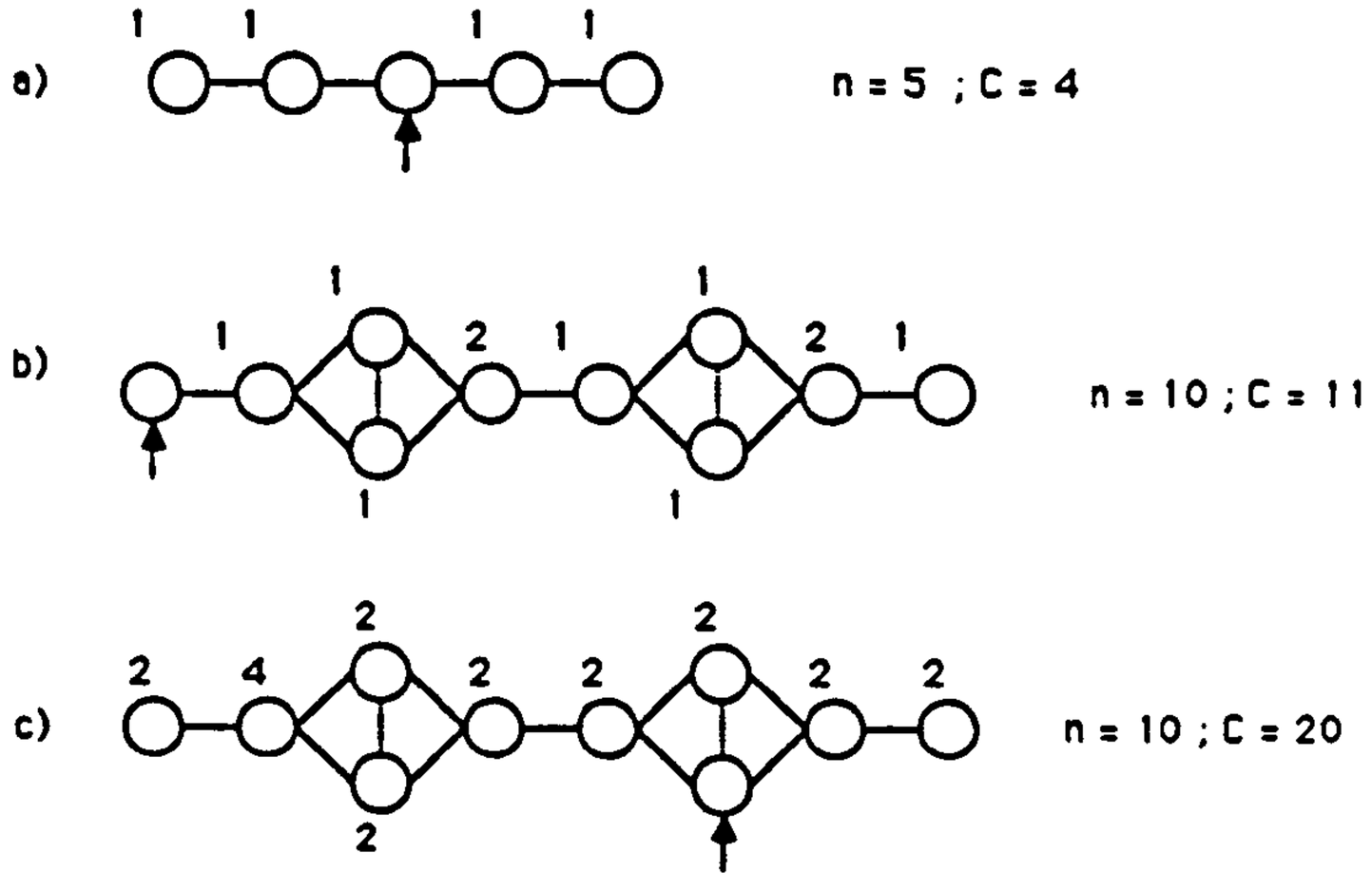

Figure 4.11

The simplest case is the linear network shown in figure 4.11 (a), for which $C = n - 1$. Each node to the left of the source receives one component constraint via the *before* relation and each to the right receives one via the *after* relation. The next case, a network containing non-linearities is shown in figure 4.11 (b). A redundant component constraint is propagated for every disjunction in the network. However, as discussed earlier, the redundant component constraint is not propagated

further, giving a value of *(n - 1) + the number of disjunctions* for *C*. This analysis is only correct if the propagation was not initiated at one of the disjunctions. Figure 4.11 (c) shows an example in which propagation is initiated at a disjunction. In this case, redundant component constraints are propagated further, resulting in a value of 2 {*(n - 2) + number of disjunctions*} for *C*.

This analysis does not take account of networks with adjacent non-linearities, shown in figure 4.12 (a), or non-linearities involving more than two nodes shown in figure 4.12 (b).

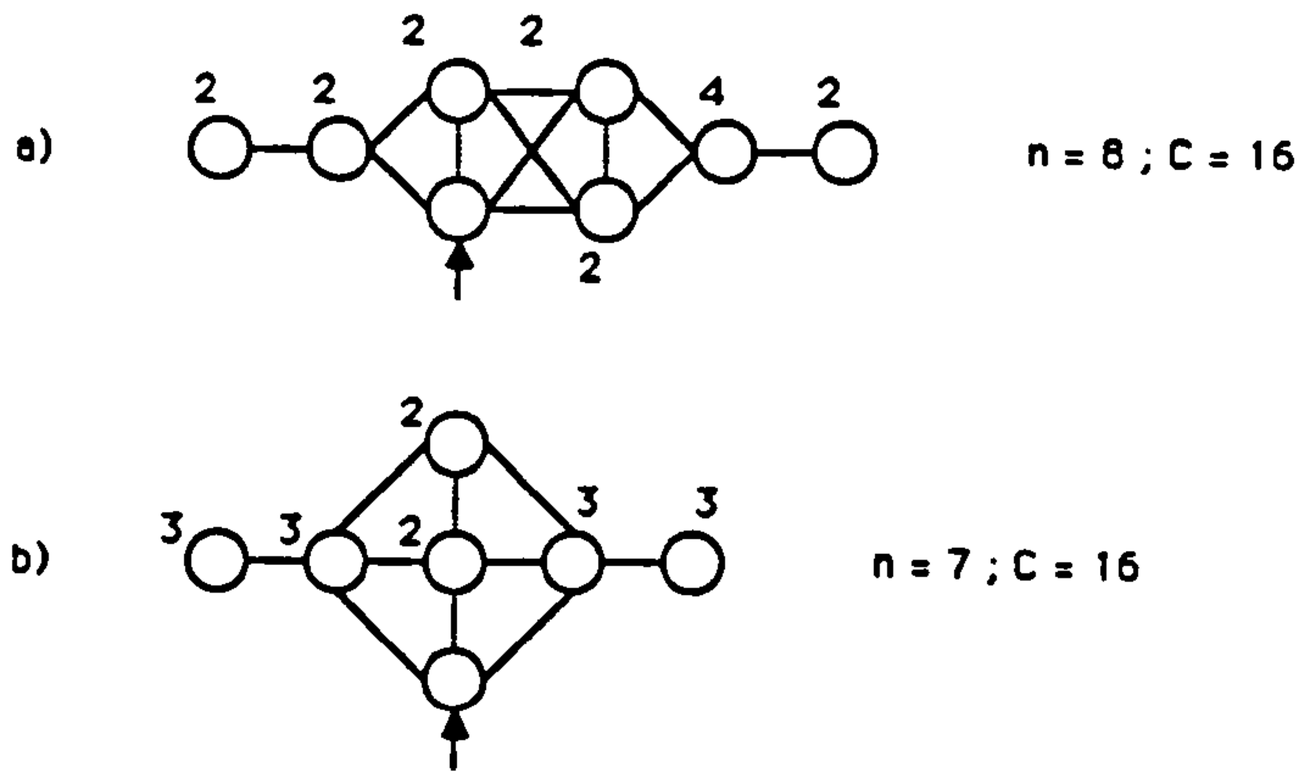

Figure 4.12

It is possible to extend the analysis to cater for such networks. It is obvious however, that such extensions will increase in complexity as the degree of non-linearity allowed in a network increases. Unfortunately, the task of determining consistency in the general case remains exponential. On a more positive note though, the ability to inhibit duplicate propagations does help to reduce the exponential effect in networks which are only "slightly" non-linear. This may be sufficient for many applications, Alcan Plate at Kitts Green for example.

4.4. Conflict Resolution Mechanisms

Realistic scheduling problems are fraught with conflict for a variety of reasons. Executional uncertainty, conflicting scheduling objectives and poorly specified problems are only a few of the factors which make conflict an inevitable feature of schedule generation and maintenance. The

CMS, discussed in the previous section, plays an important role in the handling of conflicts by highlighting areas of conflict within the schedule. This section discusses the mechanisms available to alleviate identified conflicts.

Conflict resolution within DAS is built upon three basic principles. The first is that the action appropriate to resolve a particular type of conflict is independent of whether the conflict occurred during schedule synthesis or as a result of executional uncertainty. That is, exactly the same conflict resolution mechanisms are invoked in both cases. The second basic principle is that conflicts should be dealt with as locally as possible. This is in the interests of computational efficiency and is consistent with the analogy drawn between the DAS architecture and traditional idealised management structures. The third basic principle is that the course of action most appropriate to alleviate a given conflict can only be determined accurately by analysis of the origins of the conflict.

4.4.1. Operation Priority

The ability to recognise areas of high contention within a schedule is essential to the effective focusing of problem-solving effort. Intuitively, it is those areas of a schedule which are highly contended which should receive most effort. Having expended effort both in the identification and subsequent resolution of a conflict it seems appropriate to protect that investment of effort. On the other hand, it is not appropriate to cast any particular solution in stone, as subsequent events may render that solution invalid. DAS has an operation priority mechanism, integral to conflict resolution at all levels in the hierarchy, which addresses both these concerns.

All operations introduced into the current scheduling problem are initialised to have a priority of zero. During the course of schedule synthesis or maintenance, any operation which is found to be in conflict has its priority incremented by one. In this way the priority of an operation acts as a measure of the degree of difficulty experienced in arriving at a particular scheduling decision. The quality of this as a measure of difficulty is dependent on the amount of effort expended in

determining which operations are actually involved in a conflict. How this is utilised to protect partial solutions while retaining a degree of flexibility is discussed in the following sections.

It may be of interest to note that this was not the first attempt at a solution to the problem of protecting expensive scheduling decisions. Initially, priority was associated with resources rather than operations. This is an appropriate method for measuring the degree of difficulty of scheduling decisions only if all the operations at the resource in question are involved in the conflict which is giving rise to increased priority. However, it is often the case that only a small number of the operations within a given subproblem are in conflict, while decisions on the remaining operations can be made with relative ease. The benefits of adopting the finer granularity offered by associating priority with operations rather than resources will become apparent through the following sections.

4.4.2. Operational Level

When change is introduced into the problem of an *O-agent*, whether by executional uncertainty or in the normal course of schedule synthesis, it attempts to reschedule without affecting any other subproblems. If it can find a solution which satisfies the temporal constraints acting on the new problem, it will not cause conflict elsewhere in the schedule. This is not to say that it will have no effect on other subproblems. It may modify the temporal constraints acting on operations in other subproblems while remaining consistent with existing scheduling decisions. Any change to the temporal constraints of operations are made by the CMS, while the appropriate *O-agents* are notified by lateral message passing. It is important to notify effected *O-agents* because despite the fact that their existing solutions remain valid, the notified changes may effect their ability to react to further change.

The priority of an operation is employed within the informal lateral communication between *O-agents* to focus scheduling effort. The type of message used to communicate a change to the temporal constraint of an operation contains a priority field. The value in the priority field

indicates the priority of the operation which has just been given a start time, i.e. the scheduling decision which resulted in the modification message. The receiving *O-agent* will only accept messages concerning an operation if the message is of a higher priority than its effected operation, or if the message relaxes the temporal constraint on the effected operation. In this way, problem-solving effort is focussed on difficult areas of the schedule. For example, consider the situation in which two *O-agents A* and *B* are working on separate subproblems, both involving operations from a common process plan. Assume that OpB has appeared in more conflict sets than OpA and is therefore of a higher priority. It is possible that due to a lack of conflict, a scheduling decision can be made on OpA quicker than on OpB, thus allowing *O-agent A* to dominate a portion of the search space being investigated by *O-agent B*. By allowing *O-agent B* to ignore constraining messages about OpB from *O-agent A*, *O-agent B* can retain its solution space. When *O-agent B* eventually arrives at a scheduling decision on OpB, *O-agent A* must listen to the resulting message and modify its solution to be consistent. At this point, the internal representations of both *O-agents* are consistent with the external representation.

The priority mechanism allows constrained areas of the schedule to dominate less constrained areas. It does however remain flexible because operation priority is always used on a relative basis. Thus, if a previously unconstrained portion of the schedule suddenly becomes highly constrained, due to a machine failure for example, its importance may rise above that of an area which previously dominated it. In addition to being flexible enough to cope with changing conditions, the priority mechanism operates at a level of granularity which allows it to deal with partial subproblem interaction. Had priority remained associated with resources, *O-agent B* would be allowed to completely dominate *O-agent A* in the above example. This may not always be appropriate because there may be other plans with which both *O-agents* must deal and in which *O-agent A* has the more constrained operations.

The CMS helps in the identification of conflict by maintaining the temporal constraints associated with operations in a consistent manner. However, it is the *O-agent*, in the course of its

search for a solution that identifies operations which are in conflict as a result of their temporal domains. Whenever an *O-agent* cannot find a solution to its problem, it does not simply fail. It generates a *conflict set*, a subset of the operations in its problem which it currently believes cannot be given mutually consistent start times. When an *O-agent* fails to solve its problem, it requests help by messaging its superior *T-agent*, giving it this *conflict set* as part of the message. The *conflict set* is a by-product of the mechanism used by the *O-agent* to perform dependency directed backtracking. Dependency directed backtracking requires that the constraints imposed on one operation by another during the process of forward checking be recorded. Forward checking in turn requires the *O-agent* to analyse the topology of the constraint graph of its problem, and apply constraints accordingly. The way in which an *O-agent* analyses its constraint graph is discussed more fully in [Burke et al '89].

4.4.3. Tactical Level

In the majority of cases, a *T-agent* is informed that a conflict is present within its sphere of influence via a message from one of its subordinate *O-agents*. This message takes the form of a plea for assistance by the *O-agent*, in which it specifies a set of operations (a conflict set) which it believes to be at the root of the conflict. Regardless of whether the conflict occurs as a result of a poor tactical level scheduling decision, executional uncertainty or any other reason, the *T-agent* is informed in exactly the same manner. As noted earlier, conflicts may be present in the problem specification, in which case it would be better if a *T-agent* had the ability to detect conflicts prior to delegation to the operational level. In fact, it would be more appropriate for the *S-agent* to detect some built-in conflicts, such as process plans which have due dates preceding their release dates. In the current implementation, conflicts are passed down through the hierarchy, recognised at the operational level and thereafter contained as locally as possible. While this may be in keeping with the traditional idealised management structure analogy, it is highly likely that scheduler performance would be enhanced if an attempt was made to detect and resolve conflicts as early as possible. However, despite the fact that the current implementation is not the most efficient

possible, it does demonstrate an ability to pass conflicts back to the level most appropriate for conflict resolution.

On receiving a plea for help, a *T-agent* has two options available to it. It can either deal with the problem itself or pass it up to the strategic level for resolution. In keeping with the principle of localising conflict as much as possible, a *T-agent* will endeavour to resolve a conflict itself if at all possible. The only local strategy available to a *T-agent* is one of load balancing, that is to redistribute the work load amongst its subordinate *O-agents*. Obviously, there is a danger of falling into a "load-balancing loop" in which a *T-agent* repeatedly swaps work from one resource to another, only to swap it back later attempting to resolve what it considers to be a new conflict. As discussed in chapter 3, each *T-agent* has its own *T-assistant*. One of the tasks of a *T-assistant* is to ensure that looping does not occur, and in fact to guide load balancing decisions. Whenever a *T-agent* receives a conflict set, it informs its assistant of the conflict and asks for a list of possible courses of action it may take to alleviate the conflict. Therefore, it is the *T-assistant* and not the *T-agent* which performs conflict analysis at the tactical level. When queried in this way, the *T-assistant* returns a list of actions in disjunctive normal form which it believes will resolve the conflict. The actions specify either the retraction of certain operations to the tactical level with a view to load balancing, modifications to the temporal constraints acting on certain operations or a combination of both. Essentially the later type of action involves passing an operation to the strategic level for temporal relaxation. The decision-making process followed by the *T-agent* when selecting from the options available to it is a two-stage process. The first stage involves invoking the locality principle by removing courses of action which require temporal relaxation of an operation at the strategic level. The second stage involves selecting from choices which involve only load balancing or alternatively, if all available options involve some temporal relaxation, passing a set of options to the strategic level. If the *T-agent* must select from possibilities involving only load balancing, the priority of the operations involved in the various actions is used to discriminate between them. The combined priority of all the operations involved in the execution of each option is calculated. The option with the lowest combined priority, and

hence "cost", is then selected.

In the course of conflict resolution the *T-agent* sends and receives a variety of messages. The messages it may send are listed below.

<conflict>

A *T-agent* sends this message to an *O-agent* to acknowledge that the *O-agent* is in a state of conflict. This causes the *O-agent* to wait until it receives a <continue> message before attempting to solve its problem again.

<help options>

Sent by a *T-agent* to the *S-agent* when it cannot resolve a conflict within its sphere of influence by load balancing alone. The *options* field, provided by the *T-assistant*, gives the *S-agent* a set of actions to choose from to resolve the conflict.

<continue>

Sent by a *T-agent* to an *O-agent*. This causes the *O-agent* to continue its search for a solution.

Conflict resolution activity by a *T-agent* is always initiated by the arrival of a message from one of its subordinate agents. The *T-agent* employs the suite of messages listed above when dealing with a conflict. The various messages a *T-agent* may receive notifying it of a potential conflict situation are listed below. The action taken in response to each message is also given.

<impossible conf-set>

Sent by an *O-agent* to notify its superior that it has one or more operations, specified by *conf-set*, with a null temporal domain. The *T-agent* responds by putting the appropriate resource in the *res-in-conf* list, incrementing the priority of the operations in the *conf-set* and asking the *S-agent* for assistance. Messages are sent to the *S-agent* only for those operations which are not already in the *conf-ops* list.

<quantum conf-set>

Sent by an *O-agent* to its superior when it has expended a pre-specified amount of search

effort. The *T-agent* checks to see if the subordinate has used up all the quantums it is allowed on its current problem. If not, it sends the subordinate a <continue> message. If it has, it sends it a <conflict> message, adds the resource to the *res-in-conf* list, increments the priority of the operations identified in the *conf-set* and makes a call to DEAL-WITH-CONFLICT. This last action is the one which attempts to resolve the conflict. It involves passing the conflict set to the *T-assistant* and querying it for a set of options which may resolve the conflict. The *T-agent* must then select an option and implement it.

<exhausted conf-set>

The *O-agent* sends this message when it has, through exhaustive search, established that there is no solution to its current problem. The *T-agent* responds by sending it a <conflict> message, adding the resource to the *res-in-conf* list, incrementing the priority of the operations identified in the *conf-set* and calling DEAL-WITH-CONFLICT.

<relaxed op_i>

Sent to inform the *T-agent* that an operation within its problem is part of a process plan which has had its due date relaxed. In effect, this means that op_i has been removed from the *T-agent's* problem. The *T-agent* responds by removing op_i from its internal representation, including its *T-assistant*. It then asks its *T-assistant* if the relaxation has helped any of its subordinates which are currently in a state of conflict. If so, it sends <continue> messages to the appropriate *O-agents*.

<undone op_i>

Sent to notify the *T-agent* that one of its operations has been withdrawn to the strategic level in attempt to resolve a conflict. The *T-agent's* response is exactly the same as its response to the <relaxed op_i> message. That is, both messages could be replaced by a single message.

<benefitted op_i>

Sent to inform the *T-agent* that op_i has been significantly temporally relaxed in an attempt to

resolve a conflict within its problem. The *T-agent* must remove the operation from its *conf-ops* list and notify its *T-assistant*. Having done this, it then asks its *T-assistant* if the relaxation has helped any of its subordinates which are currently in a state of conflict. If so, it sends <continue> messages to the appropriate *O-agents*.

4.4.4. Strategic Level

Like the *T-agent*, the *S-agent* is informed of the presence of a conflict by one of its subordinate agents. A *T-agent* asks for assistance from the *S-agent* by sending it a message detailing the alternative courses of action which its *T-assistant* has advised may resolve the conflict. The various options specified consist of requests for the temporal relaxation of one or more operations. The *S-agent* can achieve temporal relaxation of an operation in one of two ways. It can either invoke inter-agent backtracking, ie: have one *O-agent* undo a scheduling decision in order to benefit another, or perform due-date relaxation, in which case an order is allowed to run late. It is considered appropriate to send conflicts requiring temporal relaxation to the *S-agent* because it is necessary to consider whole process plans in order to make intelligent relaxation decisions. Only the *S-agent* has access to complete process plans.

Given a choice, the *S-agent* always selects a course of action in which it can achieve all necessary temporal relaxations by inter-agent backtracking rather than by due-date relaxation. This is consistent with the view that conflicts should be contained as locally as possible. If the *S-agent* is left with no option other than due-date relaxation, it is required to modify the problem specification. Modifying the problem specification may allow the conflict to influence decisions beyond the scope of the scheduling problem. Containing conflicts at this level is obviously more than just a matter of computational efficiency, but rather concerns the whole subject of schedule acceptability.

As is the case with the other agents in the hierarchy, the *S-agent* must analyse the cause of a conflict before deciding on a course of action to resolve that conflict. Initially it must determine

whether or not inter-agent backtracking is appropriate, and if it is, which decision within a process plan it should undo. Inter-agent backtracking involves two operations from the same process plan, one of which has been identified by a subordinate *T-agent* as being in need of temporal relaxation. Whenever a scheduling decision on an operation is undone during inter-agent backtracking, the operation in question has its start time removed and is retracted to the strategic level until such times as the relaxed operation is actually scheduled. The *S-agent* must consider two factors when deciding whether or not to invoke inter-agent backtracking. Firstly, it must consider the relative difficulty experienced in making each decision in the plan, and secondly, it must consider the temporal constraints acting throughout the plan.

A measure of the difficulty experienced in making each scheduling decision in a process plan is given by its associated priority. Therefore, it seems reasonable that only decisions on operations of a lower priority than the one requiring temporal relaxation should be considered as candidates for being undone. Also, the position of an operation within a process plan can exclude it from being a candidate for inter-agent backtracking depending on the priority of it and other operations. This is shown by example in figure 4.13.

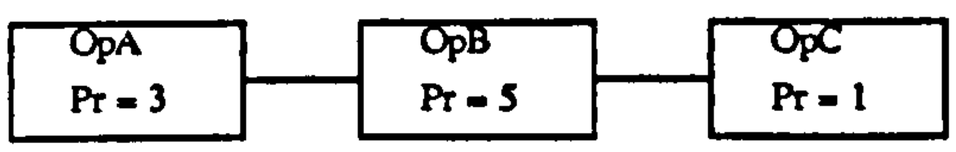

Figure 4.13

In the linear process plan above, if *OpA* is in conflict, it would be possible to undo the scheduling decision on *OpC*. However, this cannot result in the temporal relaxation of *OpA* because, due to the high priority of *OpB*, its decision cannot be undone to benefit *OpA*. Having removed candidate operations which are of a higher priority than the operation to be relaxed, and others which offer no hope of temporal relaxation because of their position with respect to such high priority operations, the *S-agent* next considers the impact of each remaining candidate on the temporal domain of the operation in conflict. This is made possible by the maintenance of a dual

representation of the temporal constraint acting on an operation as described earlier. The *S-agent* elects to undo the decision which will result in the greatest relaxation on the temporal domain of the operation to be relaxed.

Inter-agent backtracking effectively imposes an ordering on decision-making through part of a process plan by allowing a decision to be made on one operation, and its effects propagated, before a decision on another. As is the case with load balancing, there is a danger of entering into an infinite loop in which the *S-agent* undoes a decision on one operation to benefit another, only later to repeat the process in the opposite direction. In order to inhibit looping, the *S-agent* must record the decision-making sequencing enforced as a result of inter-agent backtracking. This record of previously enforced decision-making sequences is inspected by the *S-agent* when considering inter-agent backtracking to ensure that it does not repeat a decision which was enforced earlier and has apparently failed. If at any point in the process of eliminating candidates for inter-agent backtracking there are no remaining candidates, the *S-agent* concludes that the process plan in conflict cannot benefit from inter-agent backtracking and resorts to due-date relaxation.

Due-date relaxation is the ultimate weapon available to the *S-agent*. It guarantees that DAS will find a solution, if not to the precise scheduling problem presented to it, then at least to a very closely related problem. If due-date relaxation is to be an effective conflict resolution mechanism in all situations, it must involve significantly more than a simple extension of the due date on a process plan and propagation of the effects. Certainly, this would be sufficient in some instances, but as in the inter-agent backtracking example of figure 4.13, the position of an operation in a process plan may inhibit this from being an effective solution. For example, extending the due date of the process plan shown in figure 4.14 has no impact on the temporal constraint of OpB as long as OpC remains scheduled.

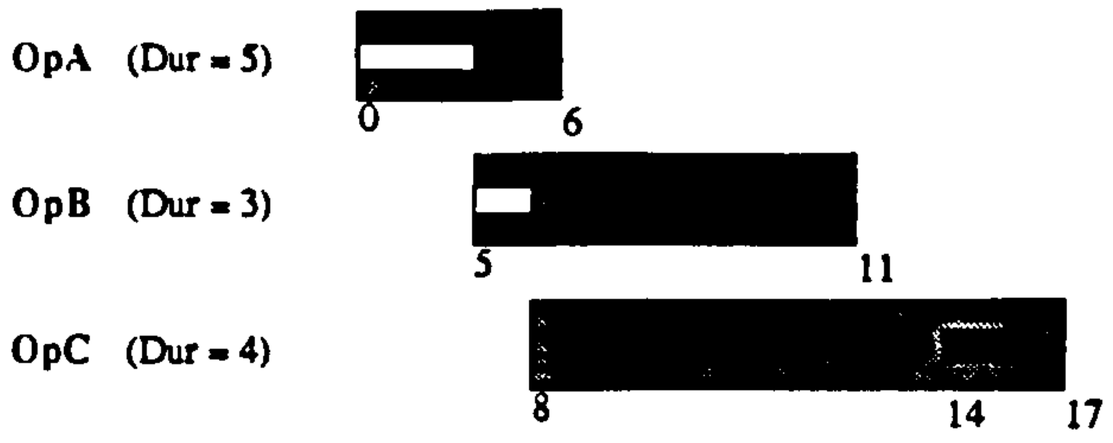

Figure 4.14

What is actually required is that all the operations in the process plan be unscheduled and subsequently re-scheduled in left to right order through the process plan. In the case of non-linearities, an arbitrary decision-making sequence is enforced. The same decision-making sequencing mechanism as is used within inter-agent backtracking is employed to guarantee the required left to right sequencing.

There is naturally concern over the degree of tardiness of an order which has had its due date relaxed. This is particularly true when the due date constraint in question has effectively been removed from the scheduling problem, rather than incrementally extended. This would appear to give *O-agents* operating a JIT strategy a license to schedule relaxed operations some time very far into the future. To counter this possibility, the *S-agent* associates a selection strategy of *earliest dispatch* with each operation in a relaxed process plan. This selection strategy overrides the selection strategy currently active at the resource to which an operation is delegated, thus ensuring that such orders are completed in reasonable time.

Like the *T-agent*, the *S-agent* employs a variety of message types during conflict resolution in order to coordinate its activity with other agents. The messages it may receive inform it of the existence of conflict or of progress towards conflict resolution. They are listed below.

<dec-cancelled op_i>

The *S-agent* updates the appropriate entry in the *startedplans* list.

<dec-made op_i>

The *S-agent* updates the appropriate entry in the *startedplans* list and delegates any operation released by this decision to the tactical level.

<help options>

The *options* field, provided by the *T-assistant*, gives the *S-agent* a set of actions to choose from to resolve the conflict. The *S-agent* must decide which of the options to implement and then implement it.

Messages are sent to subordinate agents informing them of the actions taken by the *S-agent* in an attempt to resolve current conflicts. The messages which may be sent are listed below.

<relaxed op_i>

Sent by the *S-agent* to inform one of its subordinate *T-agents* that an operation within its problem is part of a process plan which has had its due date relaxed. In effect, this means that op_i has been removed from the *T-agent's problem*.

<undone op_i>

Sent by the *S-agent* to notify the *T-agent* that one of its operations has been withdrawn to the strategic level in an attempt to resolve a conflict.

<benefitted op_i>

Sent by the *S-agent* to inform a *T-agent* that op_i has been significantly temporally relaxed in an attempt to resolve a conflict within its problem.

4.5. Predictively Coordinating Problem-Solving Effort

Throughout this thesis it has been stated that due to the dynamic nature of the target environments, DAS is primarily a reactive scheduler and that problem-solving effort should be co-ordinated opportunistically. However, it is also recognised that if an investment is made in some top-down predictive effort, there may be a resulting improvement in computational efficiency when generating rather than maintaining schedules. Additionally, it may also have a positive effect on

the quality of the schedules produced. Because DAS has evolved in a bottom up manner to specifically address reactive issues, much of the work discussed in this section could equally well appear in the future work section of chapter 6. Nevertheless, it was considered appropriate within this chapter to give a brief overview of the features available within DAS which may be utilised in a predictive role.

4.5.1. Strategic Level

Perhaps it is at the strategic level, where the problem is fairly abstract and correspondingly static, that predictive techniques can be applied to greatest effect. This investment in "top-down effort" could take the form of both attempting to detect conflicts as early as possible and, taking an aggregate view of the problem in order to detect trends within the job set. For example, the *S-agent* could fairly easily be modified to detect items of work which cannot possibly satisfy their due date constraint. Having identified such orders, the *S-agent* can perform due-date relaxation immediately, thus avoiding the computationally expensive process of delegating work down through the hierarchy only to have it passed back up for the inevitable relaxation.

The improved performance made possible by the early detection of obvious conflicts such as extremely late jobs may appear relatively small when set against the potential savings offered by tuning the DAS hierarchy to its job set. DAS has several features which are currently configured reactively, but which could be configured predictively in order to tune it to its job set. Some of the features available for tuning DAS include forcing a decision-making sequence onto a plan, the selection strategy of an *O-agent*, the selection strategy of a *T-agent* and the look-a-head facility of a *T-agent*. However, it is not clear that the predictive techniques required to tune DAS to its job set are either available or computationally cost effective.

Assuming that the required predictive techniques do exist, consider the following possibilities. An aggregated analysis of the scheduling problem may identify several bottleneck resources or perhaps that the manufacturing facility as a whole is overloaded. In the later case, the *S-agent*

may decide to relax the due date constraint on an appropriate proportion of the job set, rather than wait for the inevitable requests for assistance from subordinate agents. In the former case, the *S-agent* may choose to ensure, via decision-making sequencing, that bottleneck resources are scheduled before non-bottleneck resources. Before allowing the *S-agent* to take such action, it would be necessary to have a high degree of confidence in the results of the aggregate analysis.

4.5.2. Tactical Level

At the tactical level, *T-agents* could also provide enhancements to the efficiency of DAS if they were to analyse their tasks with a view to early conflict detection. For example, if a *T-agent* could detect a general overloading within its sphere of influence, it could select work to pass back to the strategic level for temporal relaxation without involving the operational level in an expensive conflict detection and analysis exercise.

The current implementation of the *T-agent* does perform some predictive analysis of its problem with a view to saving computational effort. As described earlier, a *T-agent* considers the existence of idle periods due to lack of labour or machine failures at its subordinate *O-agents* and the temporal constraints acting on its operations before making a delegation decision. This decision is geared to avoiding conflict at the operational level.

4.5.3. Operational Level

The scheduling subproblems which reside at the operational level are extremely dynamic and therefore have less to gain from predictive analysis than at other levels. Perhaps the most appropriate form of pre-analysis at this level is one which can determine problem feasibility. This could save the *O-agent* from expending a great deal of effort searching for a non existent solution. The current implementation of the *O-agent* does perform some primitive pre-analysis of this type. This is fully reported in [Burke et al '89].

CHAPTER 5

Case Analysis of DAS

5.1. Introduction

It has been stated that DAS views the tasks of schedule creation and maintenance as being essentially the same. It has also been stated that DAS provides mechanisms capable of managing problem-solving effort effectively in both situations. This chapter substantiates these claims through a series of examples. The following section discusses the issue of schedule creation versus schedule maintenance, and shows them to be the same within DAS. Section 5.3 contains examples demonstrating the various forms of reaction available to DAS whenever change is introduced into the current global hypothesis. The final section describes the scheduling process and shows, again by example, the flexibility of a distributed asynchronous approach during schedule synthesis. Collectively, the examples demonstrate an ability to manage problem-solving effort opportunistically.

5.2. Schedule Creation Vs. Maintenance

The scheduling problem has traditionally been viewed as a predictive task. Recently it has been recognised that there is a need for a reactive capability to repair predictively generated schedules in the face of executional uncertainty. More recently, there has been a great deal of interest in integrating predictive and reactive scheduling techniques. Alternatively, scheduling can be viewed as a problem of continuous reaction. This view is supported by the fact that a schedule is rarely created from an initial condition where all resources are available and the complete job set has been identified. It is more common for work to be on-going with new work being introduced into an existing schedule. In the later case, the introduction of new work becomes merely another source of reaction. Within such a scenario, schedule generation can be viewed as a series of

reactive events executed in order to maintain an existing schedule. DAS holds this view, a fact highlighted by the inclusion of *Introduction of new work* in the list of events requiring reaction given in figure 5.1.

Events Requiring Reaction
Late Completion of Work
Early Completion of Work
Introduction of New Work
Introduction of Maintenance
Machine Failure
Loss of Labour

Figure 5.1

Of the six events listed, two take the form of modifications to the temporal constraints of the operations involved, while the other four appear as new operations. Both the late and early completion of a work item have an effect on the temporal constraints of other operations in the same process plan. The former case represents a possible conflict situation whereas the latter represents an opportunity. The other four events requiring reaction are modelled by the introduction of an operation to the tactical level. The only difference between the way new work is introduced into the schedule and the way the other three sources of reaction are introduced is the level at which they enter. New work is introduced to the tactical level by the *S-agent*, whereas the other types of operation are added directly into the tactical level.

5.3. Reactive Strategies

The fundamental principle behind the mechanisms used to manage problem-solving effort in DAS is that effort should be focused on a problem only to the degree required by the scale of that problem. The decision to associate priority with individual operations rather than resources is a prime example of this philosophy. Whenever DAS reacts to a change in the global hypothesis, it does so in a manner which attempts to ensure that the impact of the change is localised as much as possible. DAS employs four reactive strategies to achieve this. In order of severity they are, local rescheduling within a resource, load balancing between similar resources, inter-agent backtracking within a process plan and finally due-date relaxation of an order. A particular reactive strategy is only invoked to solve a problem after all less severe strategies have failed.

The examples which follow combine to demonstrate the use of each of the four reactive strategies. Two forms of diagram are used for clarification. The first represents a partial schedule at an operational resource and the second, the partial schedule on the process plan of an order. In both cases each operation in the diagram is represented by two boxes; a grey box indicating the temporal domain *(legal_starts)* of the operation and a black or white box corresponding to the operational level scheduling decision on the operation. A white box signifies that no start time has been allocated to the operation, while a black box signifies that a decision has been made and that the box is aligned with the *x-axis* (time) accordingly. When an operation is reported as being complete, it's associated black box turns grey in colour. In the diagrams representing a partial schedule at a resource the names of the operations in the diagram appear at the appropriate point on the *y-axis*. The *y-axis* of a process plan diagram is used to identify processes, some of which may not be present within a particular plan, rather than particular operations. In both cases the *x-axis* represents time, with partial schedule diagrams having an additional summary line. Black areas on the summary line represent periods of time when the associated resource is in use.

Example 1

This example demonstrates both local rescheduling and load balancing. Figures 5.2 (a) and 5.2 (b) show partial schedules on resources *ultrasonic.scanner.1* and *ultrasonic.scanner.2* respectively. The resources are similar in that the work on one could equally well be done on the other.

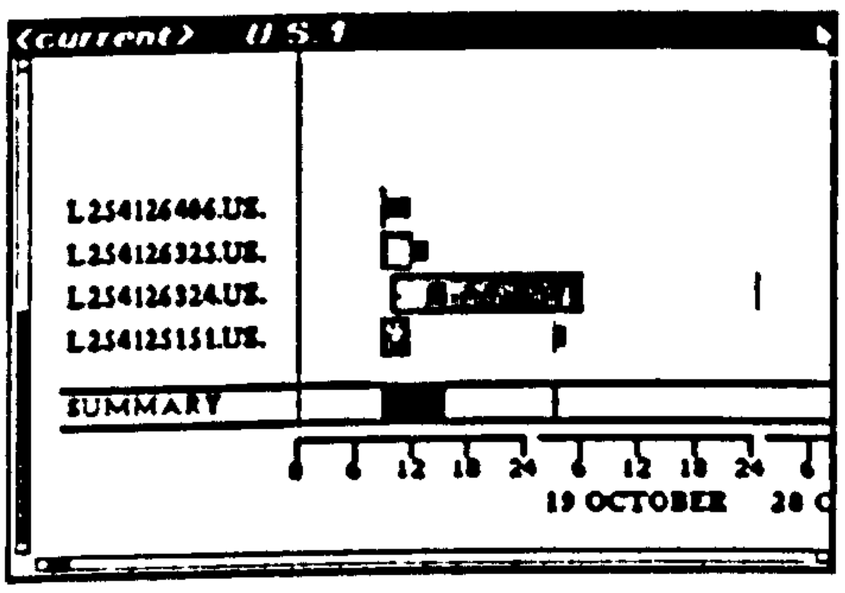

Figure 5.2 (a)

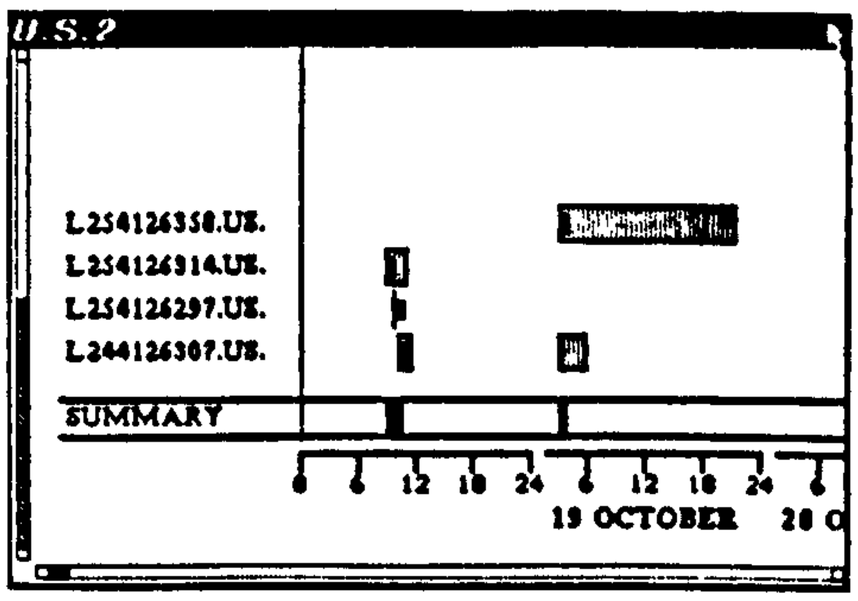

Figure 5.2 (b)

Both diagrams, 5.2 (a) and (b) contain operations from non-linear process plans. The operations in question, L254126324.US and L254125151.US from 5.2 (a) and operation L244126307.US from 5.2 (b) can be identified by the disjunctive nature of their temporal domains. A vertical dotted line in the temporal domain of an operation, eg. L254126324.US and L254125151.US from 5.2 (a), indicates that this portion of the domain is one time unit wide.

The first step of this example is to disrupt the existing schedule by introducing a requirement for maintenance on *ultrasonic.scanner.2*. The maintenance period is to last an estimated twenty four hours and be scheduled to start some time between 18:00 and 23:00 on the 18th of October. This requirement is represented by a *maintenance* operation with the appropriate temporal and technological constraints. The *T-agent* responsible for the ultrasonic scanners delegates the *maintenance* operation to *ultrasonic.scanner.2*. As can be seen from figure 5.3, local rescheduling is sufficient to accommodate the disruption. All that is required is to reschedule operation L254126358.US within its existing temporal constraints. By remaining within the existing temporal constraints the new scheduling decision cannot possibly invalidate any existing decisions at other resources.

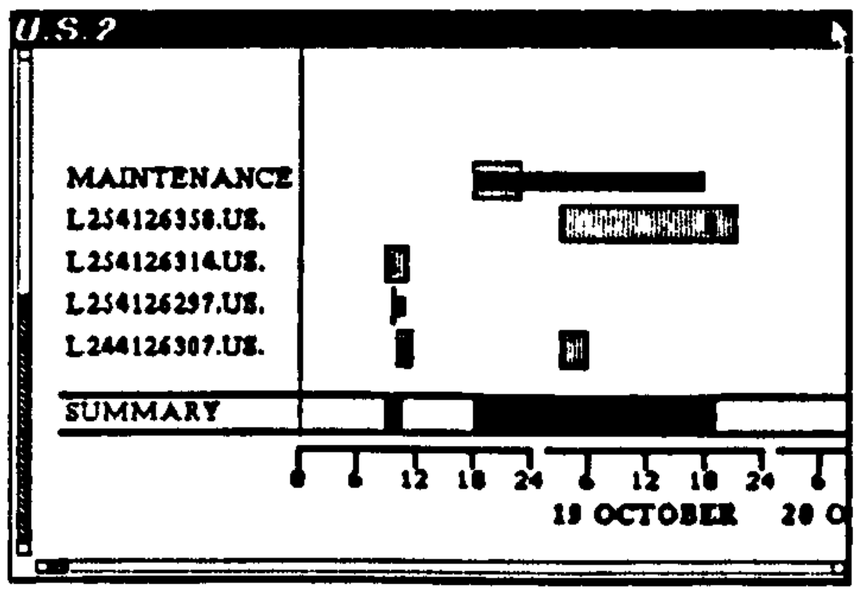

Figure 5.3

The second step of the example is to further disrupt the current hypothesis by reporting that resource *ultrasonic.scanner.2* has failed, and that the task of repairing it is expected to take four hours. This is represented by the introduction of a *repair* operation of appropriate duration and a single point temporal domain. It has a single point temporal domain to represent the fact that the start time of the resource failure is not negotiable. Unfortunately, in the face of this second

disruption there is insufficient room to manoeuvre within the subproblem at *ultrasonic.scanner.2* for local rescheduling to succeed. However, it is possible to contain the disruption within the ultrasonic scanning area of the factory by performing load balancing between the two ultrasonic scanners. This is demonstrated by figures 5.4 (a) and 5.4 (b).

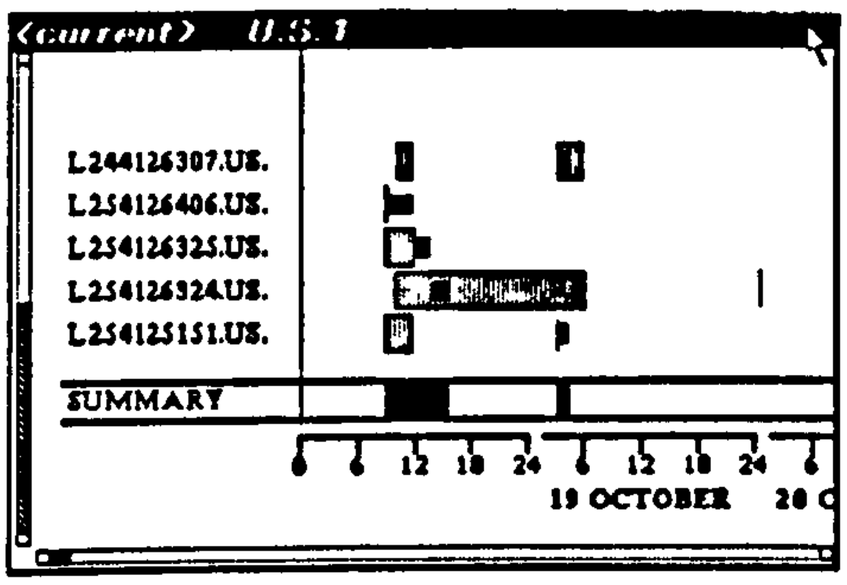

Figure 5.4 (a)

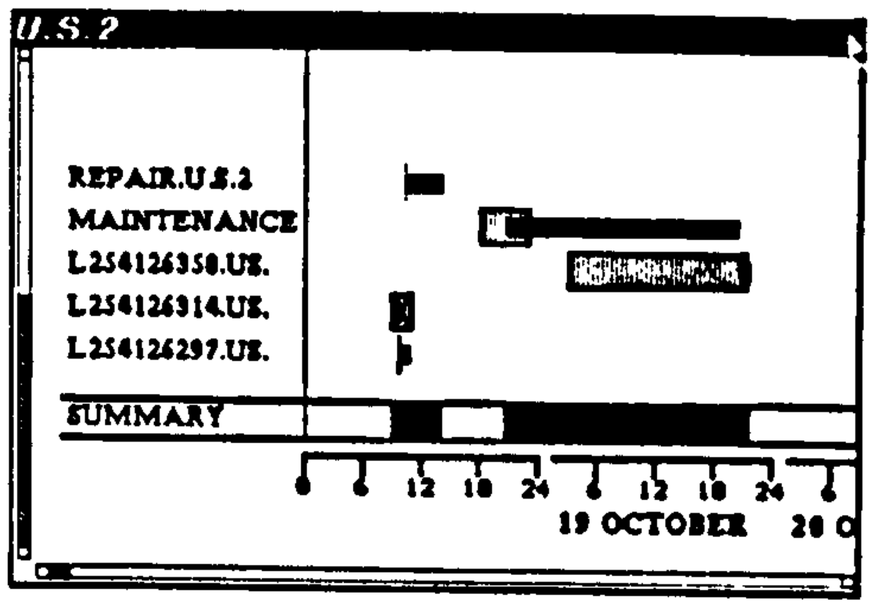

Figure 5.4 (b)

In fact all that is required is to move operation L244126307.US from *ultrasonic.scanner.2.* to *ultrasonic.scanner.1*, thus containing the problem. In addition to altering its operational resource, operation L244126307.US has also had its start time modified. However, the new start time falls within its existing temporal constraints, thus leaving current scheduling decisions at other resources in tact.

Example 2

This example concentrates on the reactive strategy of inter-agent backtracking within a process plan. In the example which follows it is invoked to resolve a conflict caused by the late completion of an operation. It could equally well be invoked to resolve a conflict caused by the introduction of new work, loss of a resource or any of the other events listed in figure 5.1. The process plan of lot L254145825 is shown in figure 5.5.

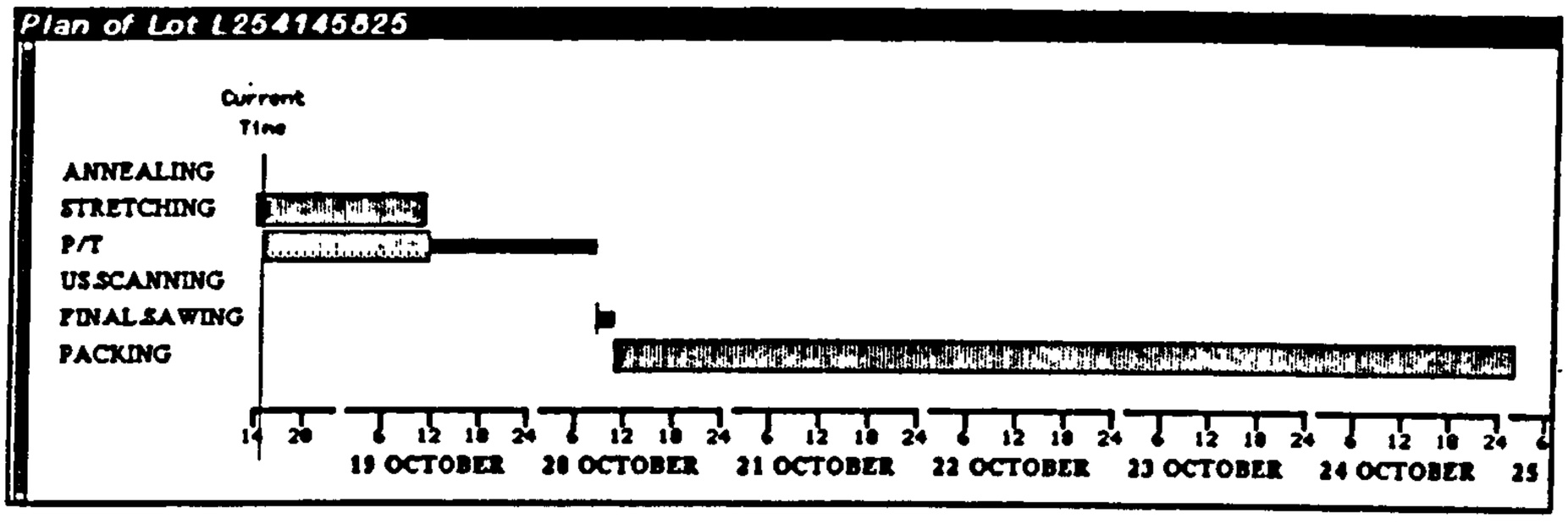

Figure 5.5

Figure 5.5 shows the lot to be completely scheduled and due to complete almost five days ahead of its required due date. Despite this "slack" in the plan, it is obvious that the decision on the start time of the final sawing process is susceptible to executional uncertainty. That is, operation L254145825.FINAL.SAWING has a single point temporal domain leaving no opportunity for a reschedule within the existing temporal constraints if required.

To demonstrate inter-agent backtracking, operation L254145825.P/T is reported to complete one hour late. This has the effect of leaving operation L254145825.FINAL.SAWING with a null temporal domain, and hence in conflict. The *O-agent* responsible for operation L254145825.FINAL.SAWING can notify its superior *T-agent* of the conflict immediately because due to the existing temporal constraints, ie: a null domain on operation L254145825.FINAL.SAWING, the *O-agent* cannot possibly resolve the conflict by local rescheduling. In this instance technological constraints prevent the *T-agent* from resolving the conflict by load balancing. The only option in such a case is to relax the temporal constraints acting on the operation in conflict, operation L254145825.FINAL.SAWING. This can be done either by inter-agent backtracking or due-date relaxation. Inter-agent backtracking is preferable, and from the diagram of figure 5.5 looks likely to succeed. Unfortunately, the diagrams generated to record the completion of an operation are potentially confusing. The black box used to identify the start time and duration of an operation changes colour to grey indicating that the operation has been completed. However, the grey box does not maintain a position which indicates when the operation was processed. Instead it reverts to the position it would occupy if it had not been

scheduled at all, ie: the left hand side of its temporal domain. Despite this, the grey boxes used to represent the temporal constraints acting on operations remain accurate. Figures 5.6 and 5.7 contain operations which have been completed.

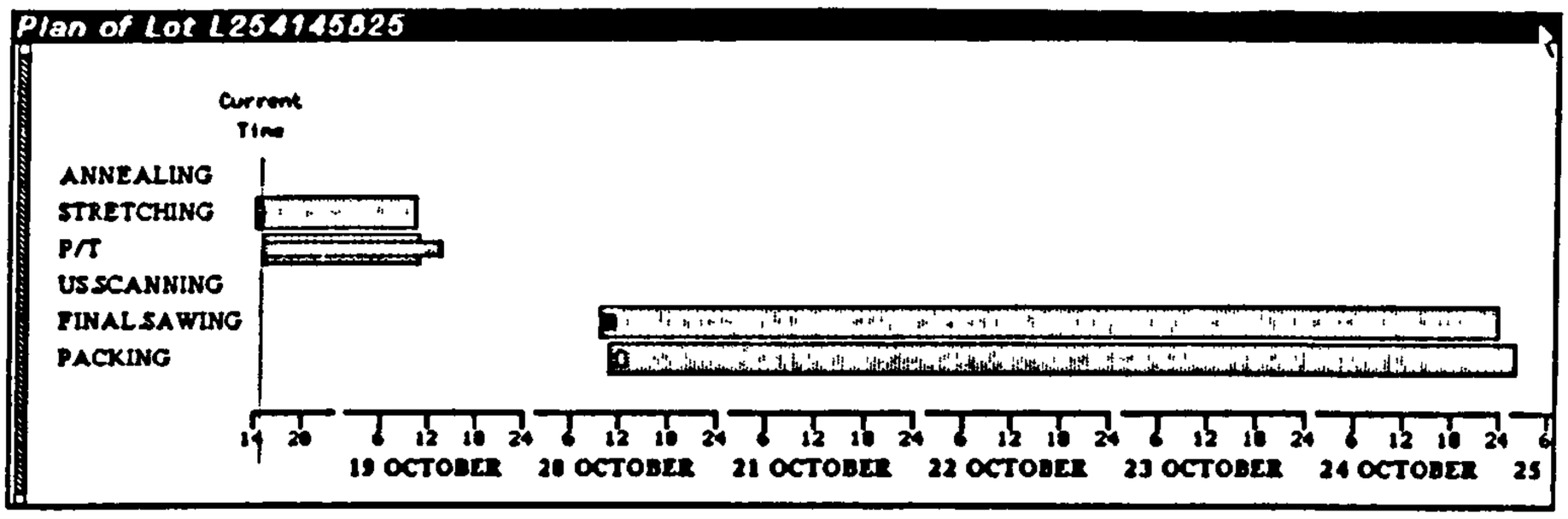

Figure 5.6

The *S-agent* elects to undo the scheduling decision on L254145825.PACKING in order to relax the temporal constraint on operation L254145825.FINAL.SAWING. Figure 5.6 shows the process plan of lot L254145825 after both the stretching and precipitation treatment operations have been completed (one hour late) and the *S-agent* has undone the scheduling decision on L254145825.PACKING. This has significantly enlarged the temporal domain of L254145825.FINAL.SAWING allowing it to be scheduled. Once the *S-agent* has been notified that the final sawing operation has been successfully scheduled it allows a decision to be made on operation L254145825.PACKING. Figure 5.7 shows the resulting process plan with both L254145825.FINAL.SAWING and L254145825.PACKING having been rescheduled one hour forward in time.

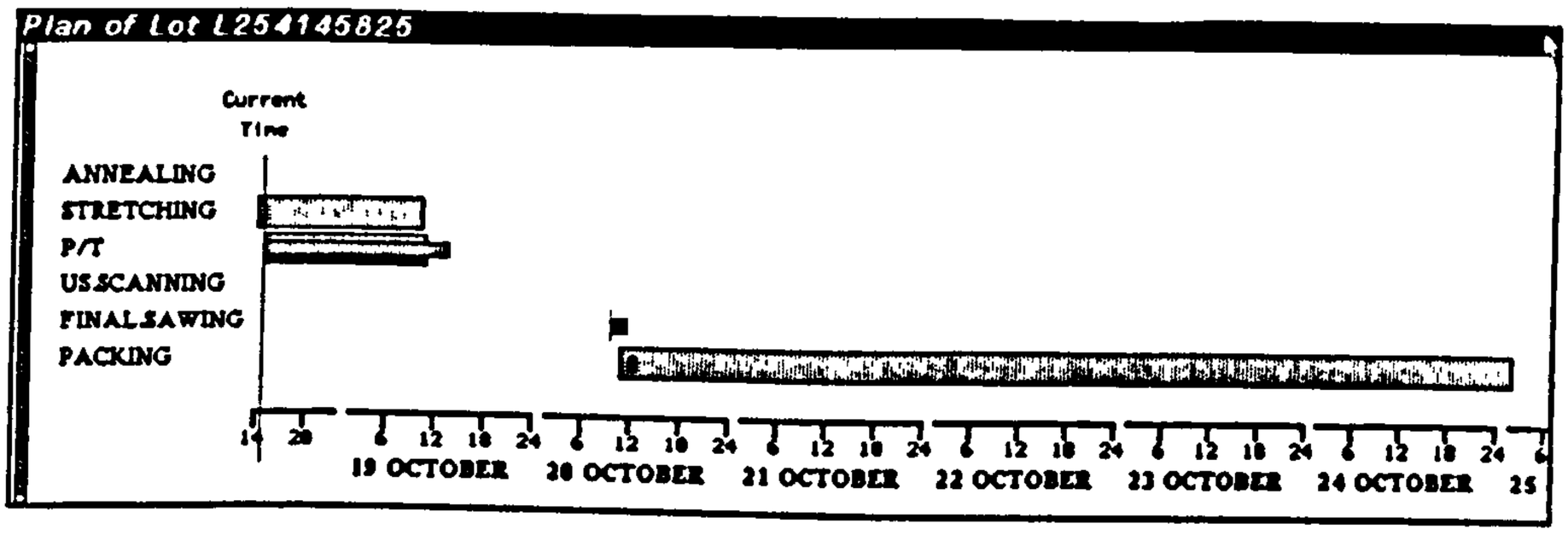

Figure 5.7

Example 3

Example 3 demonstrates both load balancing and temporal relaxation. The temporal relaxation performed includes due-date relaxation. Figure 5.8 shows the partial schedule on *furnace.h*, a batching resource, at the start of the example. It is legal for L234123211.P/T and L254126314.P/T to share the same start time because *furnace.h* is a batching resource . However, operations sharing the same start time must be capable of having the same duration. That is, one of L234123211.P/T and L254126314.P/T may have had its duration extended to permit batching. This fact is reflected in the temporal constraints acting on other operations in the same process plan.

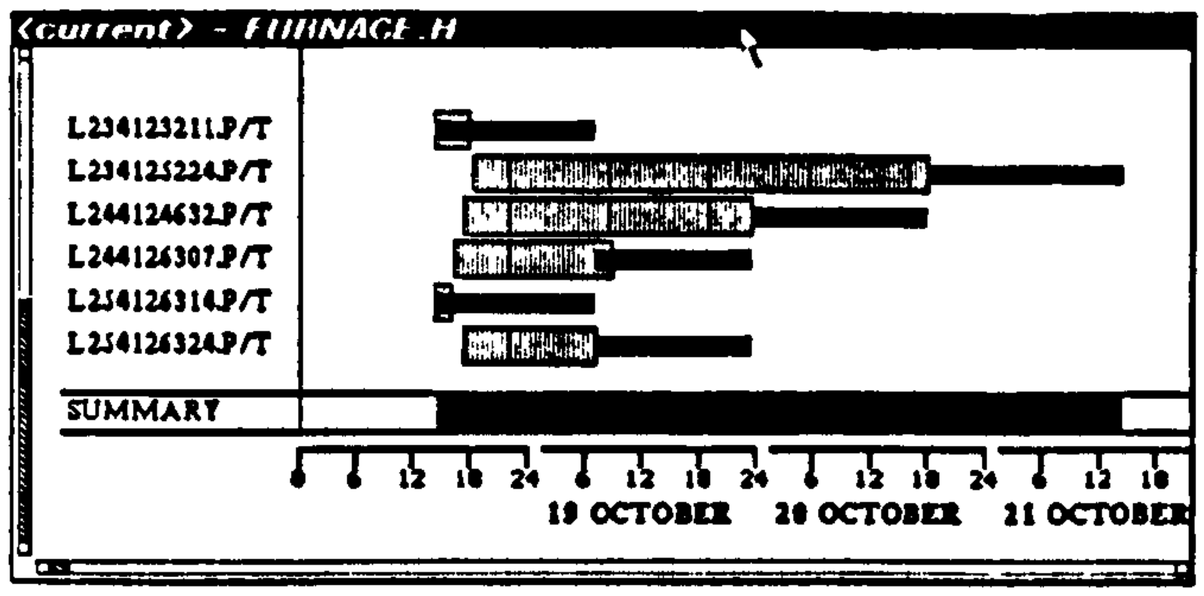

Figure 5.8

Initially, it is assumed that *furnace.h* is manned twenty four hours a day. Introducing a "nine to five" work pattern on *furnace.h* causes a major disruption to the existing schedule. In fact, because an operator is required throughout the precipitation treatment process, none of the work currently scheduled on *furnace.h* can be performed with the new manning level. That is, local rescheduling failed. Fortunately, it is possible to redistribute all the displaced work onto two similar resources *furnace.n* and *furnace.j*. The resulting schedules on *furnace.n* and *furnace.j* are shown in figures 5.9 (a) and (b) respectively.

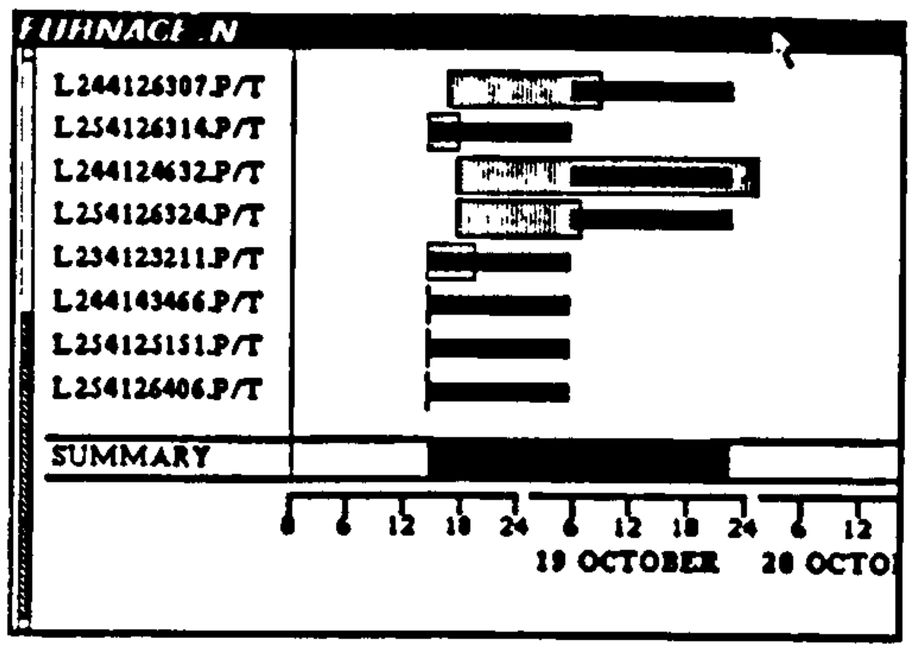

Figure 5.9 (a)

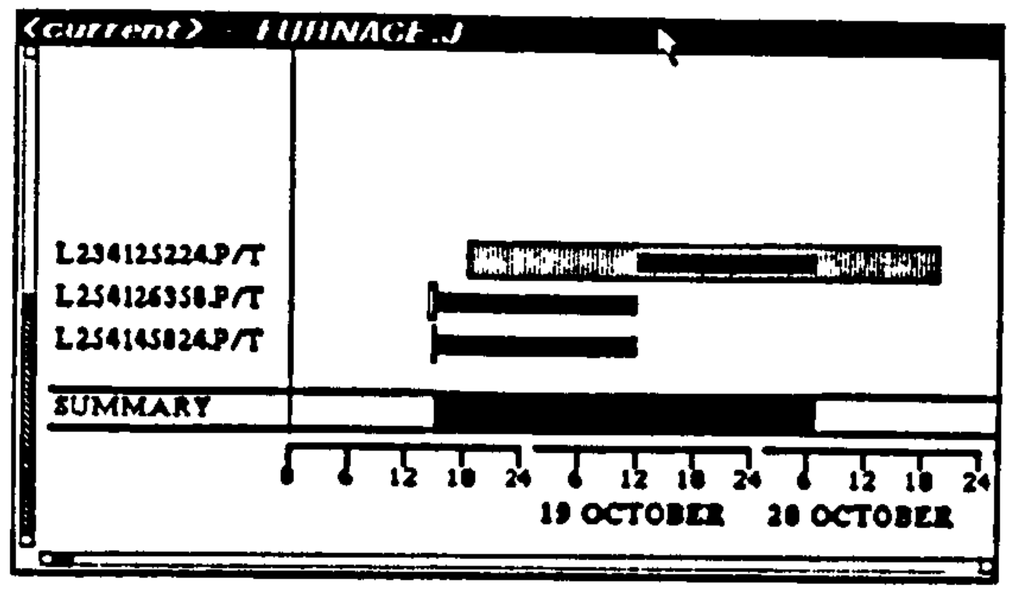

Figure 5.9 (b)

In order to force due-date relaxation, further disruption is introduced to the current hypothesis by requiring maintenance work on *furnace.n*. It is specified that *furnace.n* be maintained, a task expected to last four hours, starting some time between 16:00 and 18:00 on the 18th of October. Once again this is represented by introducing a *maintenance* operation into the subproblem at *furnace.n*. The resulting schedules on resources *furnace.n*, *furnace.j* and *furnace.h* are shown in figures 5.10 (a), (b) and (c) respectively. In figure 5.10 (c) the vertical grey bands identify the periods during which the resource is unmanned.

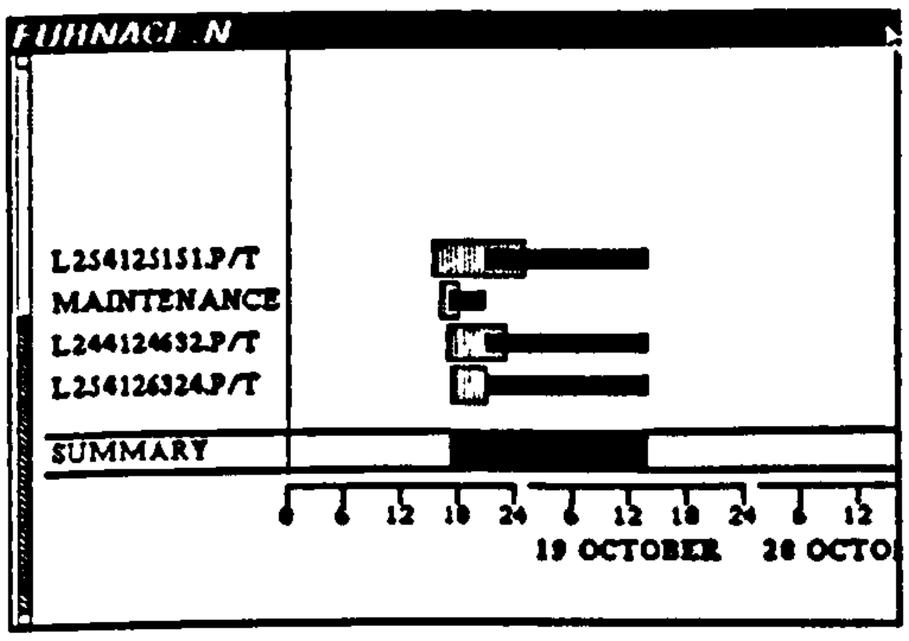

Figure 5.10 (a)

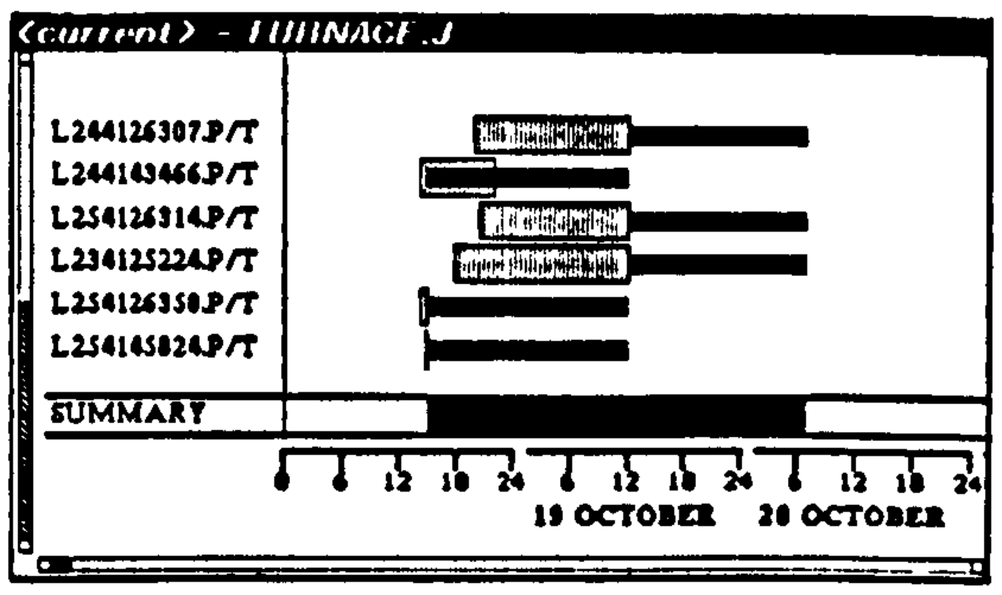

Figure 5.10 (b)

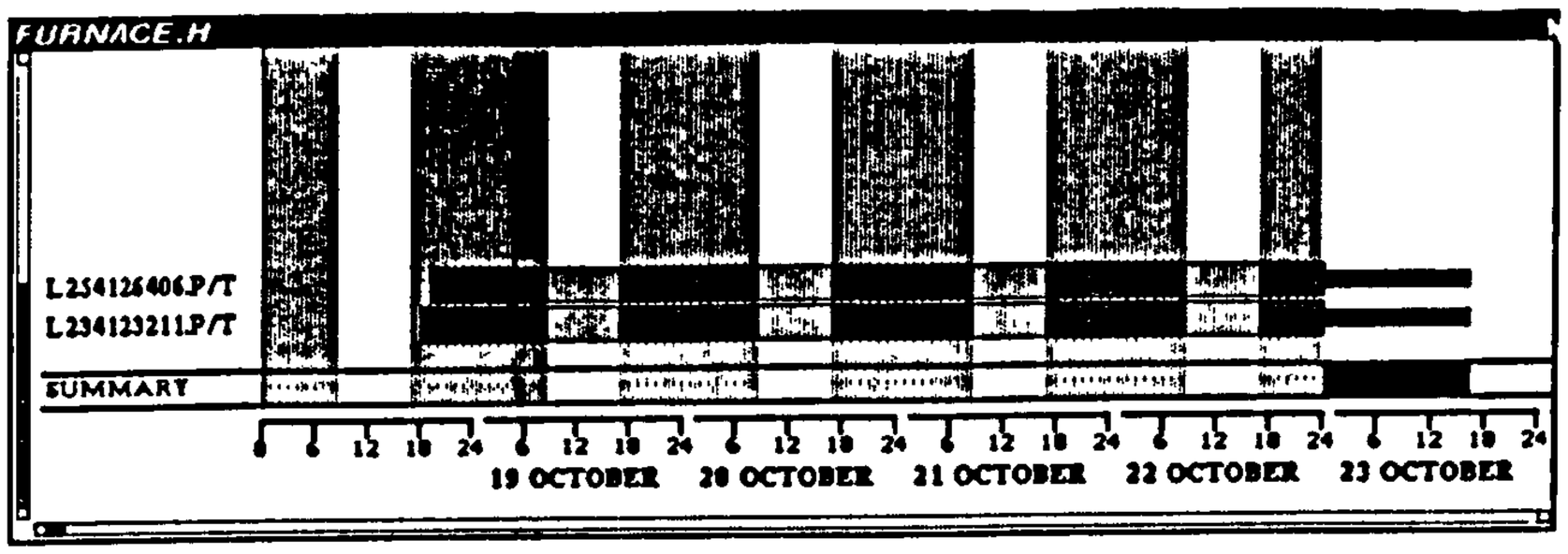

Figure 5.10 (c)

Out of all the operations on *furnace.n* before the introduction of the maintenance period, only two L244124632.P/T and L254126324.P/T retained their existing scheduling decisions. For the remainder, local rescheduling and load balancing proved unsuccessful, leaving temporal relaxation as the only option. For some, inter-agent backtracking provides sufficient temporal relaxation to resolve the conflict, but for others due-date relaxation is necessary. For example, it was necessary to relax the due date constraint on lot L254126406 in order to reschedule operation L254126406.P/T. The resulting process plan is shown in figure 5.11.

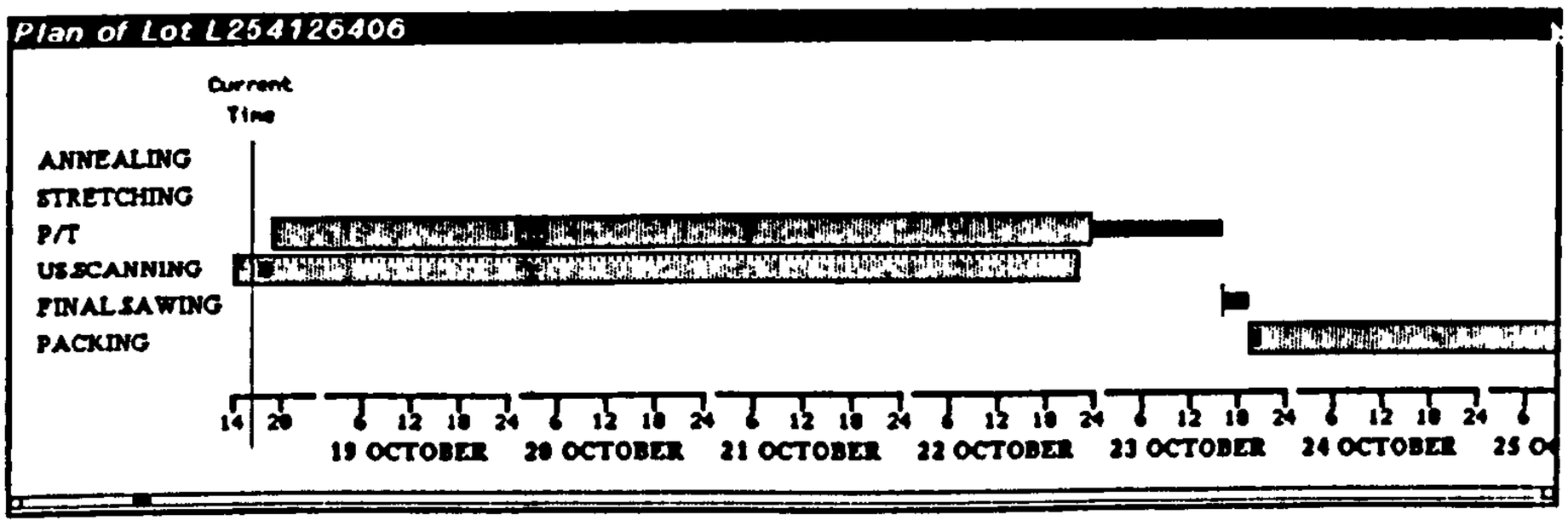

Figure 5.11

The process plan of L254126406 exhibits the consequences of due-date relaxation. There is no longer an effective due date constraint acting on the plan and all the operations are scheduled as early as possible. In addition to this it is necessary to sequence decision-making from left to right through the process plan to guarantee successful conflict resolution. It is not clear from the diagram whether a decision was taken on operation L254126406.P/T or operation L254126406.US first. Either can be the first operation in this process plan, and both have been prevented from

being scheduled as early as they might be by other constraints present at their operational resource. Neither operation L254126406.FINAL.SAWING or L254126406.PACKING have been prevented from being scheduled as early as possible by constraints at their operational resource, hence the single point temporal domain of L254126406.FINAL.SAWING.

5.4. The Scheduling Process

This section is concerned with the asynchronous nature of decision-making within the scheduling process of DAS. Having argued in favour of an asynchronous approach, a description of the scheduling process is presented. This description pays particular attention to the progression from asynchronous to synchronous decision-making as and when it is considered appropriate. The examples of this progression which follow provide further demonstrations of the reactive strategies discussed in section 5.3.

In keeping with the principle that problem-solving effort should only be focused on a problem to the degree merited by that problem, DAS allows decision-making to occur asynchronously unless it has learned of some reason to synchronise. That is, decision-making is synchronised in an opportunistic manner. Initially it allows both inter-plan and intra-plan decision-making to occur asynchronously, and only moves towards synchronisation if necessary. In a distributed system this approach offers the opportunity of maximum concurrency in situations where there is no underlying requirement for synchronisation. It also assists in the problem-solving process by not enforcing arbitrary synchronisations which can make the problem to be solved more difficult by adding artificial constraints. Perhaps more importantly, it acts as a source of learning by allowing conflicts arising out of asynchronous decision-making to guide the search. An asynchronous approach is also required to accommodate various scheduling horizons throughout the hierarchy. It is necessary to permit inter-plan decision-making to occur asynchronously because maintaining a variety of scheduling horizons may make it appropriate to only partially schedule an order.

The scheduling of an order begins when it is introduced into the system at the strategic level. The *S-agent* is notified and proceeds to delegate each operation in the process plan of the job to the appropriate *T-agents*. Each *T-agent* then delegates its operation to a particular resource and informs its *T-assistant* of this decision. This action of delegation generates a message which is sent to the relevant *O-agent*. On receipt of the message, the *O-agent* attempts to introduce the new operation into its local schedule. If it succeeds it writes out a start time for the operation. At this point in the process there is no inter-plan or intra-plan decision ordering, ie: decisions can be made asynchronously. If an *O-agent* cannot produce a consistent schedule it generates an intra-resource conflict set, a set of operations that the *O-agent* believes cannot be given mutually consistent start times on its resource. This conflict set is sent to its superior *T-agent* who informs its *T-assistant* of the conflict set. The *T-assistant* interprets this as a consequence of the delegations made to the subordinate resource. The *T-agent* then asks its *T-assistant* for advice on what load-balancing options remain. The *T-agent* acts on this advice by retracting operations from some resources and delegating operations to others as appropriate. If there are no load-balancing options available, the *T-agent* concludes that its problem is over-constrained and delivers an inter-resource conflict set to the *S-agent*. The *S-agent* analyses the inter-resource conflict sets received from its subordinates and decides upon a course of action. The selected action may be inter-agent backtracking, constraint relaxation or a combination of both. Load balancing involves synchronising inter-plan decision-making by allowing operations from one plan to be scheduled before operations from another. Both inter-agent backtracking and due-date relaxation involve synchronising intra-plan decision-making. Due-date relaxation requires complete synchronisation of decision-making through a process plan, whereas inter-agent backtracking requires only partial synchronisation. An example demonstrating opportunistic coordination of problem-solving effort, ie: the progression from asynchronous to synchronous decision-making follows.

Example 4

When a process plan is first introduced to the strategic level, decision-making may occur on the various operations within it, asynchronously of each other. In fact, decision-making may also

occur asynchronously with respect to operations of other process plans. This example demonstrates an ability to move from asynchronous to synchronous decision-making as required by the current problem-solving state. Figure 5.12 shows the process plan of order L254145825 before any scheduling decisions have been imposed on it.

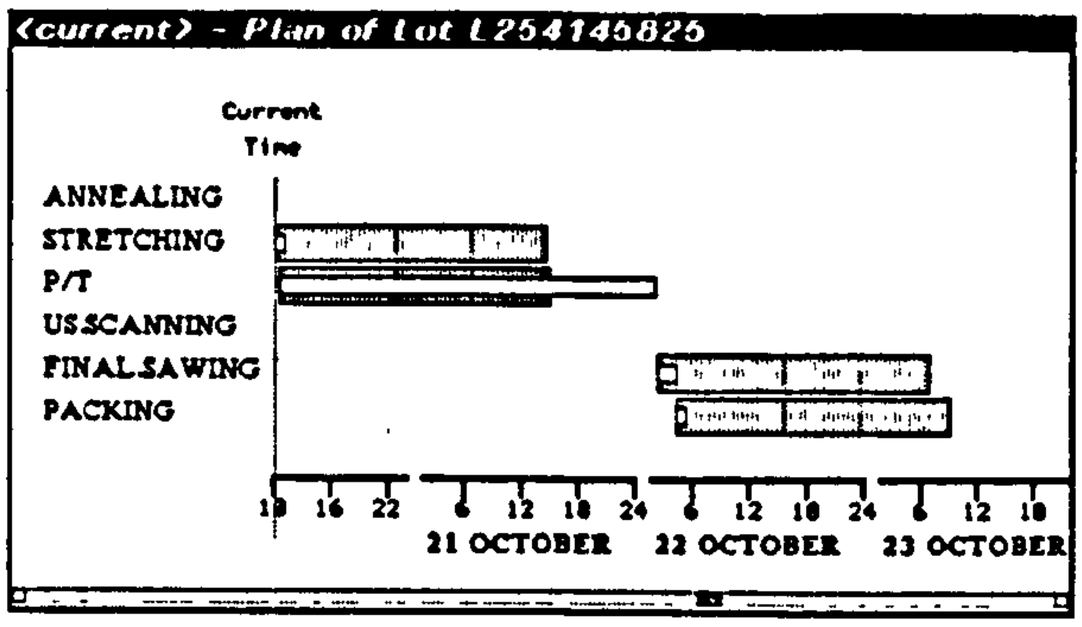

Figure 5.12

Figure 5.13 (a), (b), (c) and (d) trace the decision-making sequence of this plan. They show that the final sawing process was scheduled, then the stretching process, then the precipitation treatment process and finally the packing process. No decision ordering was imposed on this plan because at the time the decisions were being made DAS had no reason to suggest that one ordering would be better than another. The ordering which did evolve is a direct result of the speed with which the relevant *O-agents* made their scheduling decisions. The speed with which an *O-agent* makes a decision is a function of many factors including its quantum, rest interval, difficulty of task, access to the processor in a single processor system and the speed with which its superior *T-agent* delegates the operation to it.

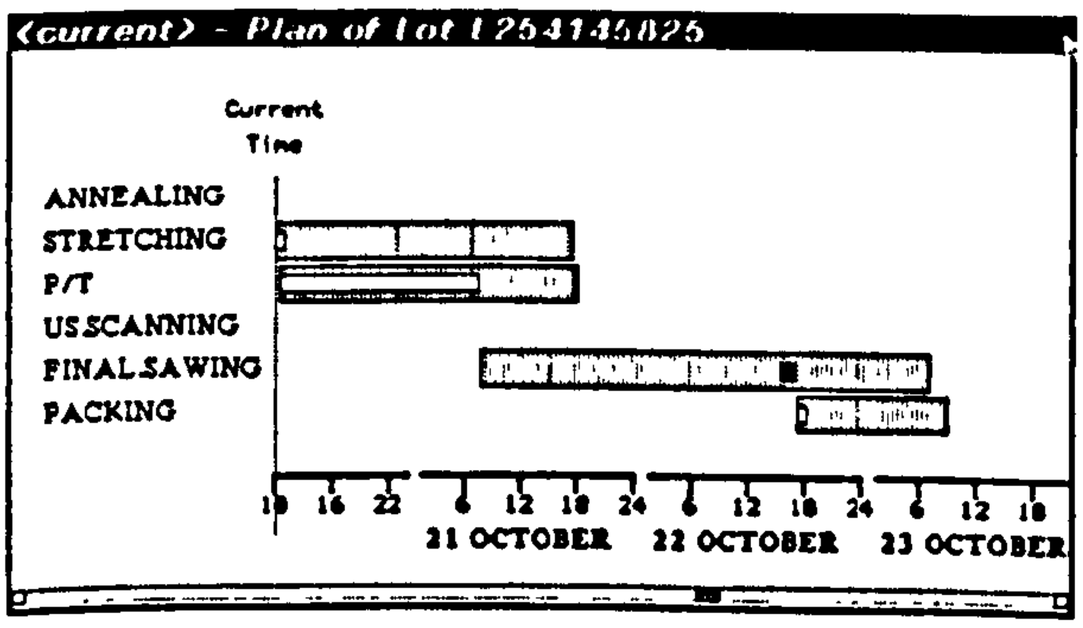

Figure 5.13 (a)

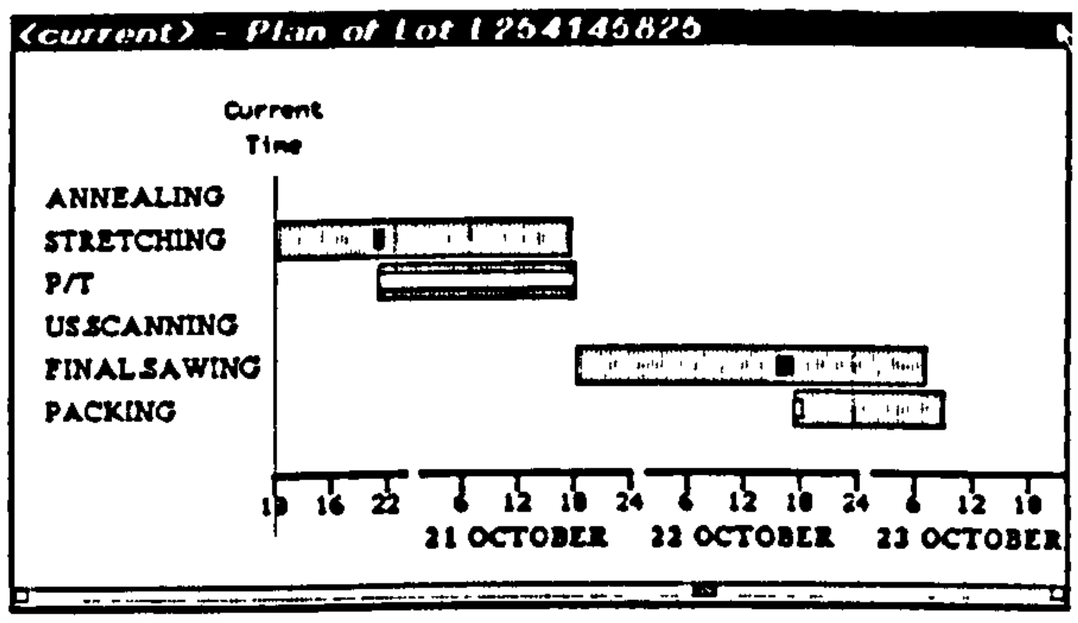

Figure 5.13 (b)

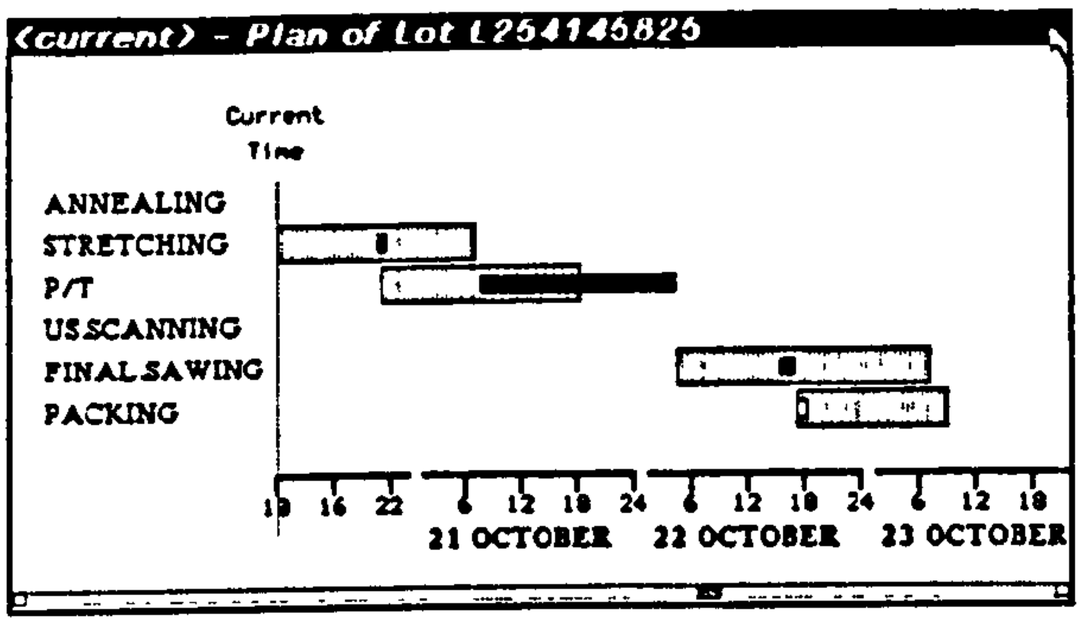

Figure 5.13 (c)

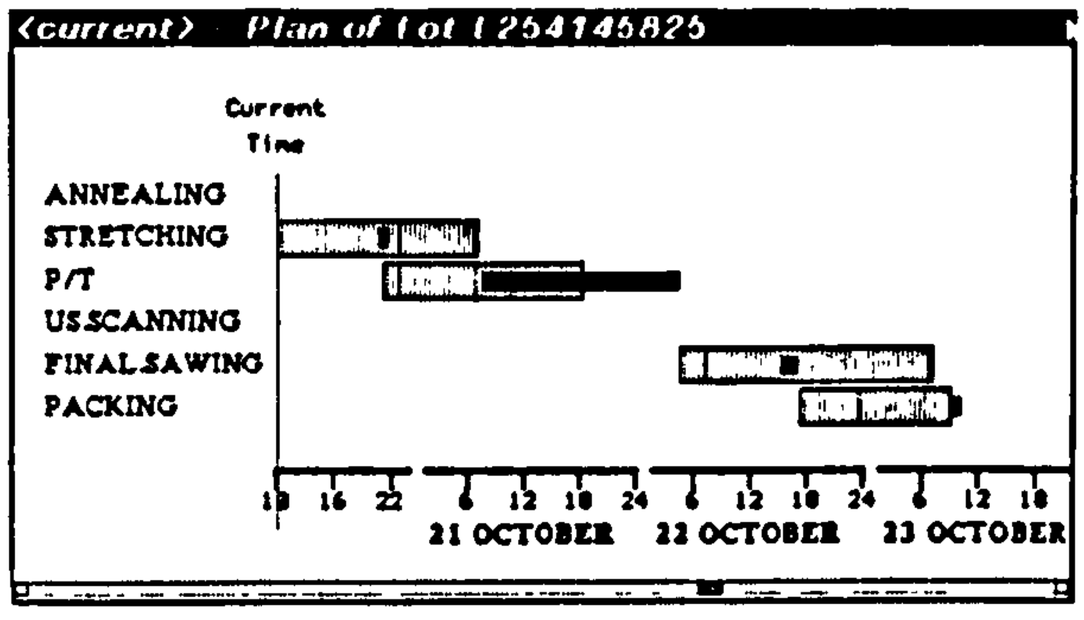

Figure 5.13 (d)

In many instances no synchronisation of decision-making is required to schedule a process plan. However executional uncertainty and the introduction of new work may generate a requirement for a degree of synchronisation. Throughout this example, decision-making on the process plan of lot L254145825 will become progressively more synchronised. In figure 5.13 the precipitation treatment operation of the plan is scheduled on *furnace.j*. The introduction of new work causes DAS to remove operation L254145825.P/T from *furnace.j* and place it on *furnace.n* (ie: load balancing). This is viewed as inter-plan synchronisation as a decision on an operation from one plan is undone in order to allow a decision on an operation from another plan. Figure 5.14 (a) shows the plan after the precipitation treatment operation has been unscheduled from *furnace.j* and 5.14 (b) shows it after it has been rescheduled on *furnace.n*.

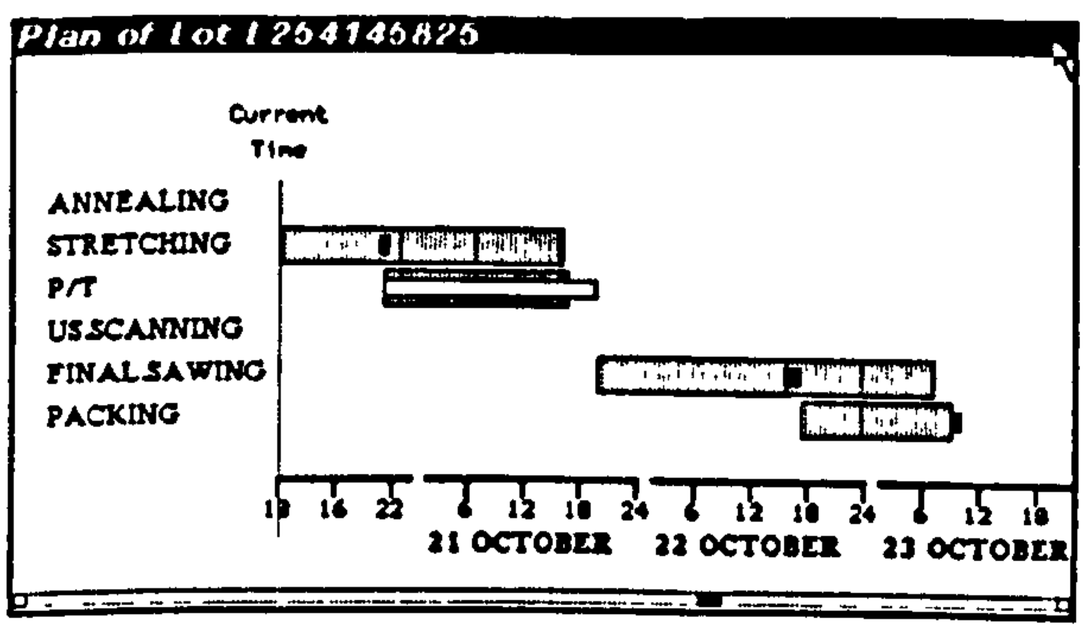

Figure 5.14 (a)

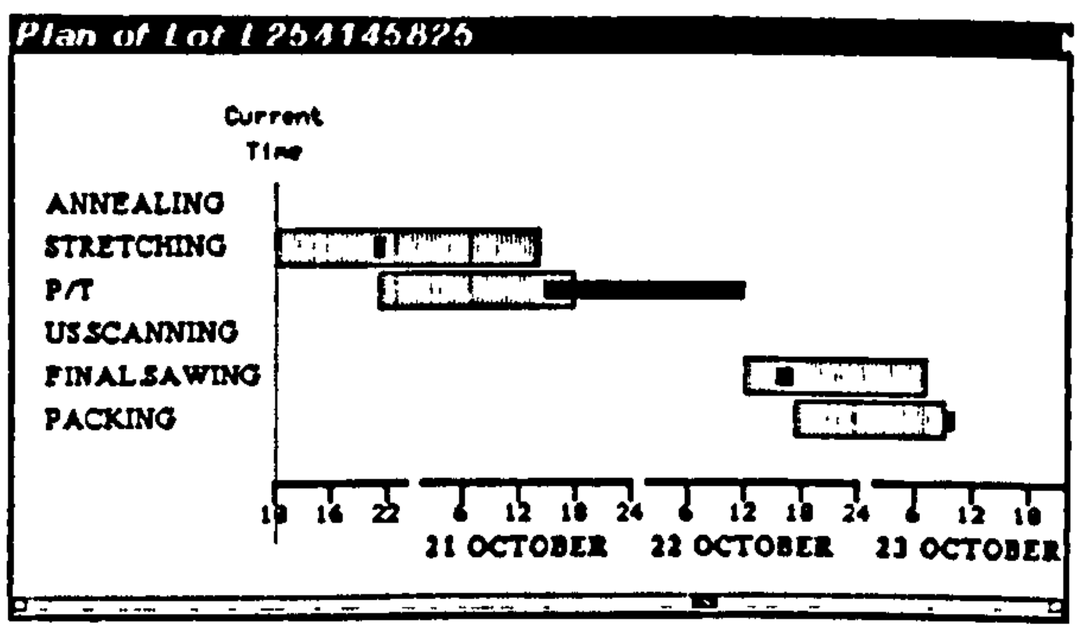

Figure 5.14 (b)

At this stage in the example there is still no intra-plan decision-making synchronisation. The next step of this example involves invalidating the schedule shown in figure 5.14 (b) by causing a disruption, at the resource of operation L254145825.PACKING. An artificially severe

disruption was introduced for the sake of example. The disruption eliminated all three packing resources from use any time after 17:00 on October the 22nd. This was necessary to exclude load balancing as an effective method of rescheduling operation L254145825.PACKING. Despite the severity of this disruption it was still possible to reschedule the operation within the existing problem constraints. If a decision on operation L254145825.PACKING is allowed to take place before a decision on L254145825.FINAL.SAWING the problem can be resolved. This partial synchronisation of decision-making through the plan is viewed as inter-agent backtracking. Figures 5.15 (a), (b), (c) and (d) trace the sequence of events during inter-agent backtracking on plan L254145825.

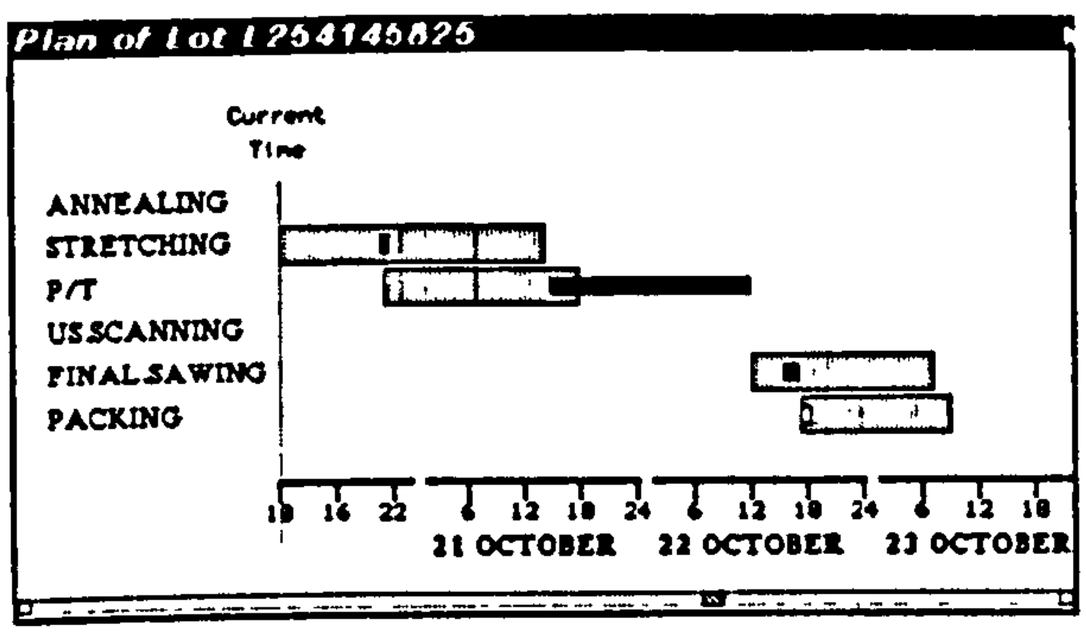

Figure 5.15 (a)

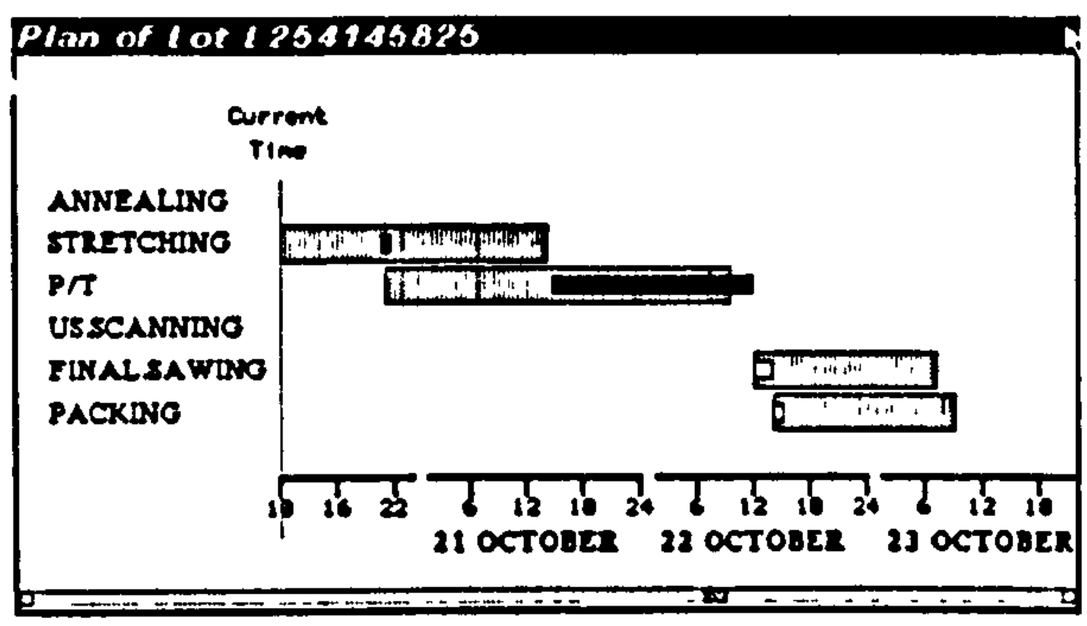

Figure 5.15 (b)

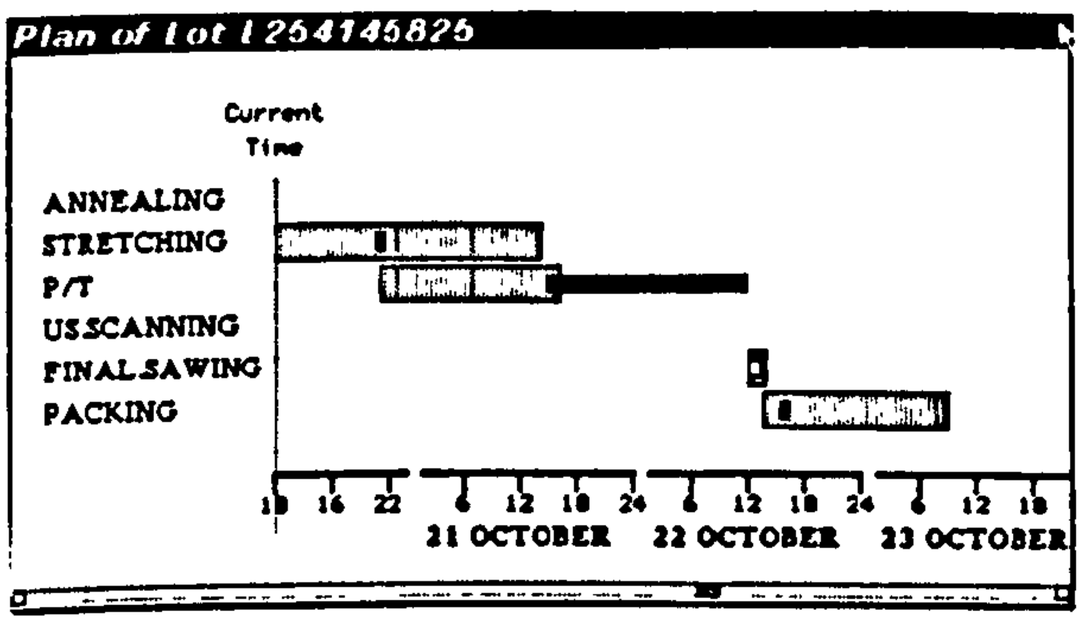

Figure 5.15 (c)

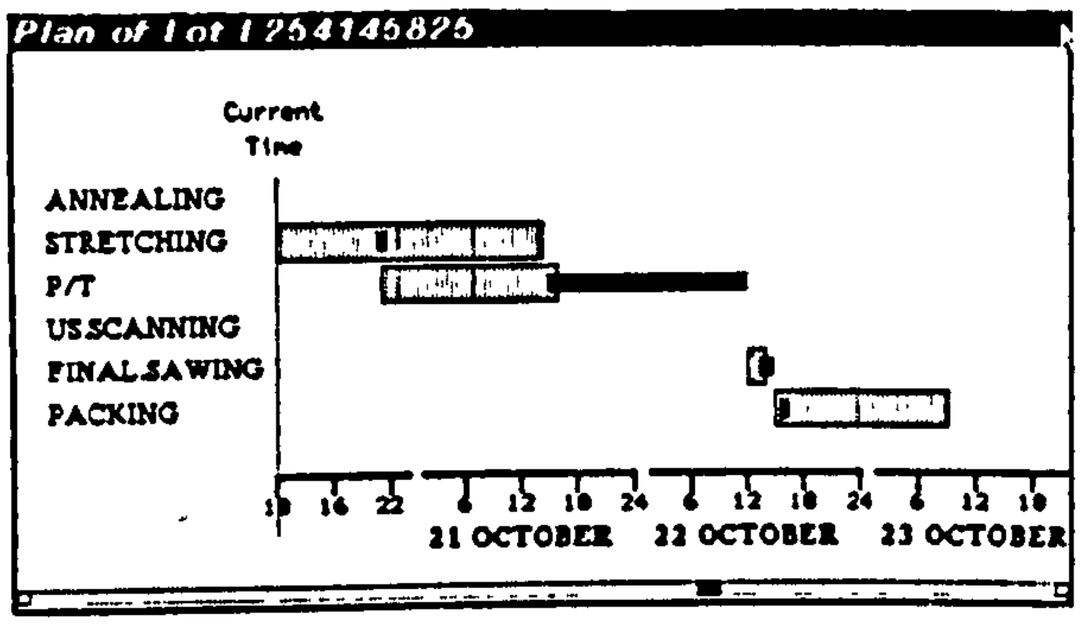

Figure 5.15 (d)

Figure 5.15 (a) shows the plan after operation L254145825.PACKING has been unscheduled to alleviate the conflict situation at its operational resource. Figure 5.15 (b) shows the first step in the inter-agent backtracking process with operation L254145825.FINAL.SAWING unscheduled. This relaxes the temporal constraints acting on operation L254145825.PACKING sufficiently to

allow it to be scheduled. This is shown in figure 5.15 (c). In response to a scheduling decision being made on the packing operation, the *S-agent* allows a decision to be made on operation L254145825.FINAL.SAWING. This completes the inter-agent backtracking process demonstrated by the fully scheduled process plan in figure 5.15 (d).

Further intra-plan decision-making synchronisation may occur in the form of inter-agent backtracking in response to other conflict situations. However, if a large enough disruption occurs, or if too much new work is introduced it may not be possible to resolve the situation by synchronisation alone. As discussed in section 5.3, it is possible to invoke temporal constraint relaxation in conjunction with complete synchronisation through a process plan. To demonstrate the effects of this, all packing resources were reported to be out of commission for an estimated three days. In order to resolve the resulting conflict it was necessary to relax the due date constraint on the plan of lot L254145825 and synchronise decision-making from left to right through the plan. This is shown in figure 5.16.

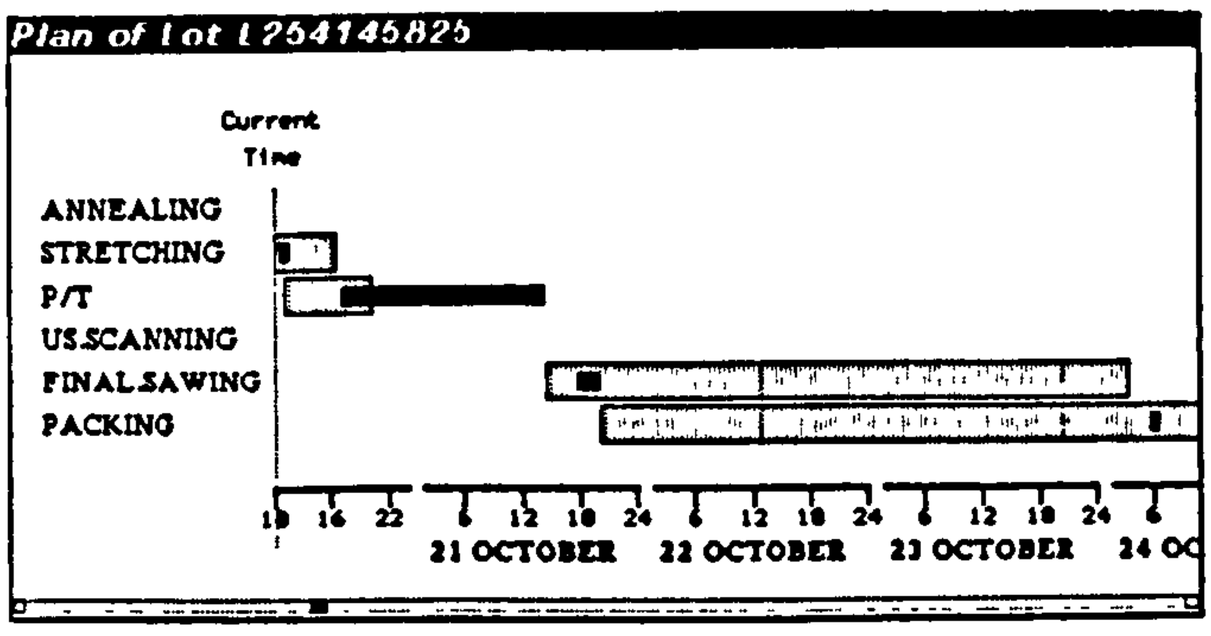

Figure 5.16

The diagram shows that each operation in the plan was prevented from starting as early as allowed by the intra-plan temporal constraints acting on it. This first three operations have not fared too badly, but the severity of the disruption at the packing resources is witnessed by the fact that the packing operation is scheduled to start two and a half days later than permitted by intra-plan temporal constraints.

CHAPTER 6

Future Work and Conclusions

Chapter 6, the final chapter of this thesis, has three purposes; to summarise the preceding chapters, to consider areas of future work and to present conclusions. The summary presented closely follows the structure of the thesis and aims to bring out the most significant points made in each chapter. The sources of future work range from identifiable weaknesses with the current implementation of DAS, through to work only made possible by the existence of a system such as DAS. Consequently, the future work section has been divided into two sections; Extensions and Empirical Analysis. Finally, the concluding remarks are intended to reaffirm the statement being made by this thesis as a whole.

6.1. Thesis Summary

Chapter 1

Scheduling is a difficult task for both theoretical and practical reasons. Theoretically, the combinatorial complexity of the task makes it a challenging one, while executional uncertainty and the difficulties associated with identifying a clear scheduling objective make it difficult from a practical point of view. Traditional approaches within manufacturing include techniques based on inventory control, MRP systems, OPT and recently JIT. Unfortunately, none of these techniques provide a satisfactory solution in real-world environments.

This thesis holds the view that in general, the scheduling task is a dynamic and stochastic satisfaction problem. Further, scheduling in a manufacturing environment must be an on-line task in order to cater for the presence of executional uncertainty. The contributions of the work presented in this thesis are a problem-solving architecture suitable for addressing the scheduling problem when viewed in this way, and more particularly, the mechanisms required to manage

problem-solving effort effectively within such an architecture.

Chapter 2

Chapter 2 presents a two-part review of work considered necessary for a full appreciation of the remainder of the thesis. The first section presents a summary of existing techniques which are intended to enhance the performance of heuristic search, and the second the application of these techniques within particular scheduling systems. It concludes that there is an opportunity to combine recent advances in scheduling techniques with advances in distributed computing technology. DAS evolved out of a recognition that this combination could offer a partial solution to both the theoretical and practical difficulties associated with scheduling in manufacturing domains.

Chapter 3

Chapter 3 introduces DAS, a distributed aysnchronous scheduling system. It deals with the DAS philosophy, representations employed and gives a detailed account of the DAS architecture.

DAS recognises the scheduling environment to be dynamic and stochastic in nature. It views satisfaction, rather than optimisation, as its primary objective. Within this framework, reactive techniques geared to schedule maintenance are considered more appropriate than off-line predictive techniques. Due to its intended operating environment, DAS approaches its problems in an opportunistic manner. Another consequence of the nature of its intended operating environment is that, in the presence of conflict, problem-solving activity is organised in order to localise the effects of the conflict.

A frame-based representation is used to model the environment, the current set of problem constraints and the schedules produced by DAS. The main representational components of the model are resources, operations and plans. Resources are modelled at two levels of abstraction; individual resource level and aggregate resource level. Plans represent particular orders from the order book and consist of temporally-related operations. An operation represents a particular process which must be performed during the manufacture of a specific work item. Constraints are

attached to these major components in order to describe the scheduling problem, the order book and the current hypothesis. In addition to this frame-based representation, scheduling agents have their own internal representation of the world.

The DAS architecture is based on an analogue of an idealised management structure in that it is hierarchical, distributed, and depends heavily on communication for success. It is a three-tier hierarchy consisting of strategic, tactical and operational levels. The distributed, asynchronous and hierarchical features of the DAS architecture offer many benefits when addressing the fundamental difficulties of the scheduling problem. Each level in the hierarchy defines one or more scheduling focal points, within which decision-making may proceed asynchronously with respect to all other focal points. Each focal point has an associated scheduling agent responsible for all activity at that point in the hierarchy. The activity performed by an agent is dependent on its level in the hierarchy. Communication between agents is achieved via message passing.

Chapter 4

Chapter 4 presents the mechanisms required to manage problem-solving effort in DAS in a globally consistent manner. The major components are a constraint maintenance system (CMS), the conflict resolution strategies available to the various classes of scheduling agent and inter-agent communication via message passing. The chapter begins by presenting information considered necessary for a full understanding of the CMS.

The CMS performs two very important roles in DAS. Firstly, it maintains the global hypothesis in a consistent manner, thus allowing scheduling agents to share a common view of the world and secondly, it assists in highlighting conflict within a schedule. The three most important features of the CMS are its ability to deal with constraint retraction correctly and efficiently, to propagate over certain classes of constraint network in polynomial time and to assist in the task of conflict resolution. A complexity analysis of the performance of the CMS concludes that in general the task of determining consistency in a network containing disjunctions is NP-hard. However, for a restricted class of networks the CMS can determine consistency in time polynomial

with the number of nodes in the network.

Conflict resolution strategies exist at all three levels of the hierarchy. DAS has an operation priority mechanism, integral to conflict resolution throughout the hierarchy, aimed at protecting parts of a solution which were difficult to generate. The priority attribute of an operation allows problem-solving effort to be focused opportunistically. At the operational level, conflict resolution takes the form of local rescheduling. At the tactical level, conflict resolution is achieved via load balancing between similar resources. Conflict resolution at the strategic level involves either partial or complete temporal relaxation of a process plan. This is performed via inter-agent backtracking or due-date relaxation respectively. Whatever the level of conflict resolution, message passing is used to co-ordinate the activity of agents to ensure that they operate in a globally consistent manner. The various conflict resolution strategies act to coordinate problem-solving effort opportunistically.

Chapter 5

Chapter 5 presents a case analysis of DAS which justifies, by example, the claims made by this thesis.

6.2. Future Work

6.2.1. Extensions

An obvious area for future work on any project is to correct weaknesses which have become apparent during system development and testing. There are two known weaknesses with the current implementation of DAS. Both are considered weaknesses because they result in an unnecessary waste of problem-solving effort. The first concerns the way in which agents throughout the hierarchy process their message buffers. An agent processes its message buffer by reading and acting on the message at the top of the list, ie: it processes messages in strict chronological order. There are opportunities at all levels in the hierarchy to reduce the amount of problem-solving effort required to solve a problem by performing an analysis of all the messages

currently in the message buffer. Consider for example an O-agent which has a message buffer containing the messages shown in figure 6.1.

1) <modify op_i priority>

.

.

n) <delete op_i >

Figure 6.1

In the current implementation, the *O-agent* in question would update its internal representation in accordance with message 1, try to solve the modified problem, and then continue to process the remaining messages in chronological order until it arrives at message n. On processing message n, it will remove op_i from its internal representation and try to solve the resulting problem. If the contents of the message buffer were analysed as a whole, it would be possible to detect that message 1 is nullified by message n and is therefore redundant. Any problem-solving effort expended in an attempt to solve the problem containing the modified op_i is obviously wasted effort. There are potentially greater savings at the strategic and tactical levels. In addition to detecting redundancy, messages could be processed together with a view to identifying symptoms of a common conflict. This offers the possibility of an improvement in the efficiency of conflict resolution.

The second known weakness concerns the way in which the priority of an operation is calculated. The present method encourages relatively unconstrained areas of the schedule to temporarily dominate more constrained areas. This is quickly corrected when the priority of the operations in the highly contended areas rises sufficiently to allow them to dominate the schedule. The problem-solving effort wasted in arriving at and correcting the temporary solution gives cause for concern. The root of the problem is that it takes longer for an *O-agent* to request and receive assistance with its problem than it takes another *O-agent* to schedule a relatively unconstrained operation. As a result of this, decisions on unconstrained operations are written out and can constrain operations which have not yet had their priority increased by the relevant *T-agent*. A better arrangement would be to have individual *O-agents* maintain the priority of an operation

during the course of search. For example, it could increment the priority of an operation after every 2000 comparisons during search. In this way, highly constrained operations would receive the protection of an increased priority at an early stage, thus avoiding the temporary situation discussed above.

Perhaps the most natural extension of the work presented here would be to re-implement it as a multi-processor system. DAS is designed to be a multi-processor system capable of taking advantage of the benefits offered by concurrent processing. It already has in place the mechanisms required to manage problem-solving effort across a collection of processors. Such an implementation would provide a useful vehicle for experimental analysis of both distributed problem-solving and distributed processing.

There is a growing feeling within the AI community (eg. [ESPRIT '89]) that future research in the problem-solving area should consider a marriage of numeric and symbolic techniques. The ideal situation is one in which each technique is applied to the tasks to which it is best suited. By providing an architecture and the mechanisms necessary to coerce its active components, scheduling agents, into generating a globally consistent solution, DAS offers an opportunity to experiment with a mix of symbolic and numeric techniques. Problem-solving agents could easily be replaced by functionally similar components implemented using alternative technology. Another potential area of future work is to augment DAS with more predictive scheduling capabilities. Perhaps numeric techniques provide the most appropriate implementation of these functions.

Two other prominent candidate areas for further work involve extensions to the CMS. The first is to enhance the range of temporal relations which can be propagated over, and the second is to cater for capacity constraint propagation. It is desirable to extend the range of temporal relations in order to allow DAS to represent, and therefore cater for, a wider range of problems. Ideally the range of temporal relations should be extended to include all seven (thirteen including inverses) identified in [Allen '84]. In the current implementation, DAS is unable to represent the need for secondary resources within a process plan. The fact that a particular process requires an

operator in attendance at the beginning of the process to initiate it cannot be represented within the existing range of temporal relations. The two processes, one representing an action on a work item and the other the presence of an operator, should be linked by the *starts* temporal relation. Another example, not involving secondary resources, concerns the requirement to take samples of a product at intermediate stages of the manufacturing process in the interests of quality control. Often all that is required is that a sample be taken sometime *during* a process, thus requiring inclusion of the *during* relation in the CMS. Extensions to the range of temporal relations permitted would enable DAS to cater for activities such as quality control and secondary resource processes, thus allowing it to schedule at a lower level of detail.

In the current implementation, scheduling agents have the sole responsibility for satisfying capacity constraints. The fact that the CMS does not support them in this task is considered acceptable at the operational level because only the *O-agent* making decisions at a resource needs to know the consequences of those decisions. This is also considered acceptable at the tactical level because the *T-agent* has access to the current schedule at each of its subordinate resources. The strategic level is not concerned with capacity constraints at individual resources. However, it is potentially useful to propagate the capacity constraint consequences of scheduling decisions directly to operations by modifying their *possible.resources* slot in accordance with current scheduling decisions. Whether this form of constraint propagation saves more problem-solving effort than it expends is an area for investigation. It may be appropriate in certain areas of a factory model and not in others, or possibly not at all.

6.2.2. Empirical Analysis

It has been argued that, in the majority of real world scheduling applications, the scheduling task is predominantly one of maintenance rather than creation. Furthermore, it has been demonstrated that it is possible to approach the task, posed as a satisfaction rather than an optimisation problem, in an entirely opportunistic manner. While this thesis has demonstrated "proof of concept" and argued in favour of this approach, it has not investigated performance-

related issues. The two most interesting performance indicators are problem-solving efficiency and the quality of solutions produced.

As noted in section 6.2.1, there are known weaknesses in the current version of DAS which if corrected would certainly enhance its problem-solving efficiency. However, there are other aspects of its operation where relationship to problem-solving efficiency is not so easily identified. For example, the degree to which scheduling agent behaviour is synchronised up and down the hierarchy affects problem-solving efficiency. This should not be confused with decision-making synchronisation which deals with synchronisation across the hierarchy. One extreme is to allow all agents to operate completely asynchronously at all times, while the other is to allow only one agent to be active at a time. While the second extreme defeats the purpose of having a distributed, asynchronous architecture, the first is not without fault. Is it wise to allow subordinate agents to continue working on a difficult problem at the same time as a superior agent is in the process of simplifying the problem? Similarly, is it wise to introduce new work to lower levels of the hierarchy until the work already there has been scheduled? Various degrees of agent synchronisation should be experimented with, to identify the best point of balance between completely asynchronous behaviour and completely synchronous behaviour.

Another important trade-off worthy of empirical analysis concerns the distribution of effort between pre-analysis of a problem and the actual search for a solution. For example, how much effort should a *T-agent* expend when deciding to which resource to delegate an operation ? It may go to a great deal of trouble to select a resource which it thinks is marginally more suitable than the rest, only to have its subordinate inform it that it cannot be scheduled. Alternatively, it would be equally unsatisfactory to have a *T-agent* delegate work to a resource it knows is currently being repaired. Similarly, how much effort should an *O-agent* expend trying to determine whether or not there is a solution to its problem before actually trying to solve it? How much effort should an *O-agent* expend trying to solve a problem before asking for help? The selection of a conflict resolution strategy presents itself as another example of this trade-off. Are there efficient methods

for identifying reactive strategies which will not succeed in certain situations?

The other major performance indicator of a scheduling system is the quality of the solutions it generates. DAS has many variable parameters which effect the nature of the solution it generates. As discussed earlier, it is very difficult to identify a single measure with which to judge the quality of a schedule. It is not being suggested that any attempt be made to identify a measure of schedule optimality, simply that the consequences of various parameter settings on the nature of the solution generated be analysed. Each scheduling agent has its own suite of parameters which can be configured to strive for a local objective. It would be interesting to investigate the effects of conflicting objectives and how they are resolved. The task of selecting one option from a number of options during conflict resolution is another degree of freedom with which to experiment.

6.3. Concluding Remarks

The work presented in this thesis was motivated by a desire to address the practical as well as theoretical difficulties of the scheduling problem. The practical difficulties originate mainly from the dynamic and stochastic nature of typical scheduling environments. It has been argued that within such environments the task is predominantly one of schedule maintenance rather than creation, and that the objective is satisfaction rather than optimisation. Viewing the task in this way has led to the conclusion that an opportunistic approach to both problem-solving and the management of problem-solving effort is appropriate. If an opportunistic approach is not employed, it is difficult to see how else to manage problem-solving effort without the introduction of unwarranted artificial constraints.

This thesis has presented a problem-solving architecture suitable for addressing the scheduling problem when viewed in this way. It has been recognised that if any benefit is to be gained from the architecture introduced, it is necessary to supplement it with mechanisms capable of managing problem-solving effort. DAS, a scheduling system based on this architecture, views the tasks of schedule creation and schedule maintenance to be essentially the same. It has been

demonstrated that DAS has the mechanisms necessary to manage problem-solving effort effectively in an opportunistic manner. In conclusion, both the architecture and the mechanisms presented lend themselves very well to the opportunistic approach required to address the problem of scheduling in real world environments.

While DAS has demonstrated "proof of concept", it leaves a great deal of scope for future work. It provides a good starting point for an investigation into the combination of numeric and symbolic problem-solving techniques. It also provides a vehicle to examine the trade-off between pre-analysis and search in problem-solvers operating in uncertain environments. Additionally, there are a number of natural extensions and areas of experimental analysis worthy of future work.

References

(1) [Allen '81]

Allen J.F.

An Interval Based Representation of Temporal Knowledge.

Proceedings of the Seventh International Joint Conference on Artificial Intelligence (pp. 221-226), Vancouver, Canada, August 1981.

(2) [Allen '83]

Allen J.F.

Maintaining Knowledge About Temporal Intervals.

Communications of the ACM, Vol. 26, No. 11, November 1983, pp. 832-843.

(3) [Allen '84]

Allen J.F.

Towards a General Theory of Action and Time.

Artificial Intelligence Vol. 23 (1984) pp. 123-154.

(4) [Berliner '73]

Berliner H.J.

Some Necessary Conditions for a Master Chess Program.

Proceedings of the Third International Joint Conference on Artificial Intelligence (pp. 77-85), Stanford University, California, August 1973.

(5) [Berliner '79]

Berliner H.J.

The B* Tree Search Algorithm: A Best First Proof Procedure.

Artificial Intelligence Vol. 12 (1979) pp. 23-40.

(6) [Burke et al '89]

Burke P. and Prosser P.

A Distributed Asynchronous System for Predictive and Reactive Scheduling.

Technical Report: AISL-42-89, Knowledge Engineering Group, Department of Computer Science, University of Strathclyde, Glasgow U.K..

(7) [Collinot et al '87]

Collinot A. and LePape C.

Controlling Constraint Propagation.

Proceedings of the Tenth International Joint Conference on Artificial Intelligence (pp. 1032-1034), Milan, Italy, August 1987.

(8) [Collinot et al '88]

Collinot A., Le Pape C. and Pinoteau G.

SONIA: a knowledge-based scheduling system.

Artificial Intelligence in Engineering, Vol. 3, No. 2, 1988, pp. 86-94.

(9) [Davis '83]

Davis R.

Diagnosis Via Causal Reasoning: Paths of Interaction and The Locality Principle.

Proceedings AAAI-83, Washington, DC (1983) pp. 88-92.

(10) [Davis et al '83]

Davis R. and Smith R.G.

Negotiation as a Metaphor for Distributed Problem Solving.

Artificial Intelligence Vol. 20 (1983) pp. 63-109.

(11) [Davis '87]

Davis E.

Constraint Propagation With Interval Labels.

Artificial Intelligence Vol. 32 (1987) pp. 281-331.

(12) [Dean '85]

Dean T.

Temporal Imagery: An Approach to Reasoning About Time for Planning and Problem Solving.

Research Report: 433, Yale University, New Haven, CT..

(13) [de Kleer et al '84]

de Kleer J. and Brown J.S.

A Qualitative Physics Based on Confluences.

Artificial Intelligence Vol. 24 (1984) pp. 7-83.

(14) [de Kleer '86]

de Kleer J.

An Assumption-Based Truth Maintenance System.

Artificial Intelligence Vol. 28 (1986) pp. 127-161.

(15) [Doyle '79]

Doyle J.

A Truth Maintenance System.

Artificial Intelligence Vol. 12 (1979) pp. 231-272.

(16) [Duncker '45]

Duncker K.

On Problem Solving.

Psychological Monographs, Vol. 58 (270).

(17) [Elleby '87]

Elleby P.

Problem Solving With Temporal Constraints.

Technical Report, Knowledge Systems Group, Department Of Computer Science, Reading University U.K..

(18) [Elleby et al '88]

Elleby P., Fargher H.E. and Addis T.R.

A Constraint-Based Scheduling System for VLSI Wafer Fabrication.

Technical Report, Knowledge Systems Group, Department of Computer Science, Reading University U.K..

(19) [Erman et al '80]

Erman L.D., Hayes-Roth F., Lesser V.R. and Reddy D.R.

The HEARSAY-II Speech Understanding System: Integrating Knowledge To Resolve Uncertainty.

Computing Surveys, Vol. 12 No. 2, June 1980, pp. 213-253.

(20) [ESPRIT '89]

Commission Of The European Communities.

1989 ESPRIT Work Programme, Brussels, 25 July 1989.

(21) [Fikes '70]

Fikes R.E.

REF-ARF: A System for Solving Problems Stated as Procedures.

Artificial Intelligence Vol. 5 (1970) pp. 1-50.

(22) [Fikes et al '71]

Fikes R.E. and Nilsson N.J.

STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.

Artificial Intelligence Vol. 2 (1971) pp. 189-208.

(23) [Fox M.S. '83]

Fox M.S.

Constraint-Directed Search: A Case Study of Job-Shop Scheduling (Phd. Thesis).

Technical Report: CMU-RI-TR-83-22, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa..

(24) [Fox B.R. et al '85a]

Fox B.R. and Kempf K.G.

Complexity, Uncertainty and Opportunistic Scheduling.

IEEE International Conference on Artificial Intelligence (1985), pp. 487-492.

(25) [Fox B.R. et al '85b]

Fox B.R. and Kempf K.G.

Opportunistic Scheduling For Robotic Assembly.

IEEE International Conference on Robotics and Automation (1985), pp. 880-889.

(26) [Fox B.R. et al '85c]

A Representation for Opportunistic Scheduling.

Robotics Research, The Third International Symposium (Editors Faugeras O.D. and Giralt G.)

The MIT Press, Cambridge Massachusetts.

(27) [French '82]

French S.

Sequencing and Scheduling.

John Wiley and Sons.

(28) [Haralick et al '80]

Haralick R.M. and Elliot G.L.

Increasing Tree Search Efficiency for Constraint Satisfaction Problems.

Artificial Intelligence Vol. 14 (1980) pp. 263-314.

(29) [Harrison '85]

Harrison M.C.

The Concepts of Optomized Production Technology OPT - The Way Forward.

BPICS CONTROL, June/July 1985, pp. 7-15.

(30) [Hart et al '68]

Hart P., Nilsson N. and Raphael B.

A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

IEEE Transactions on Systems and Science Cybernetics, Vol. SSC-4, No.2, pp. 100-107, July 1968.

(31) [Hayes-Roth et al '79]

Hayes-Roth B. and Hayes-Roth F.

A Cognitive Model of Planning.

Cognitive Science, Vol. 3, No. 4, 1979, pp. 275-310.

(32) [Hewitt '77]

Hewitt C.

Control As Message Passing.

Artificial Intelligence Vol. 8 (1977) pp. 323-363.

(33) [Intellicorp '86]

KEE 3.0 Technical Manuals Volumes (1-3).

(34) [Johnson '54]

Johnson S.M.

Optimal two- and three-stage production schedules with set up times included.

Naval Research Logistics, Quarter 1, pp. 61-68.

(35) [Johnson et al '74]

Johnson L.A. and Montgomery D.C.

Operations Research in Production Planning, Scheduling, and Inventory Control.

John Wiley and Sons.

(36) [Knuth et al '75]

Knuth D. and Moore R.

An Analysis of Alpha-Beta Pruning.

Artificial Intelligence Vol. 6 (1975) pp. 293-326.

(37) [Land et al '60]

Land A.H. and Doig A.G.

An Automatic Method for Solving Discrete Programming Problems.

Econometrica Vol. 28, 1960, pp. 497-520.

(38) [Lawler '73]

Lawler E.L.

Optimal Sequencing of a single machine subject to precedance constraints.

Management Science, Vol. 19, 1973, pp. 544-546.

(39) [LePape '85]

LePape C.

SOJA: a Daily Workshop Scheduling System.

Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems, Warwick, Great Britain.

(40) [LePape et al '87]

LePape C. and Smith S.F.

Management of Temporal Constraints for Factory Scheduling.

In Proceedings IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems (TAIS 87) (Editors Rolland C., Leonard M., and Bodart F.).

Held in Sophia Antipolis, France, May 1987.

Elsevier Science Publishers.

(41) [McCarthy et al '69]

McCarthy J. and Hayes P.J.

Some Philosophical Problems From The Standpoint of Artificial Intelligence.

In Machine Intelligence 4 (Editors Meltzer B. and Michie D.)

Edinburgh University Press.

(42) [McDermott '80]

McDermott D.V.

Spatial Inferences With Ground, Metric Formulas on Simple Objects.

Research Report: 173, Yale University, New Haven, CT..

(43) [Mackworth '77]

Mackworth A. K.

Consistency in Networks of Relations.

Artificial Intelligence Vol. 8 (1977) pp. 99-118.

(44) [Mackworth et al '85]

Mackworth A.K. and Freuder E.C.

The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems.

Artificial Intelligence Vol. 25 (1985) pp. 65-74.

(45) [Malone et al '83]

Malone T.W., Fikes R.E. and Howard M.T.

Enterprise: A Market-like Scheduler for Distributed Computing Environments.

Working Paper: CISR WP 111 (Sloan WP 1537-84), Center for Information Systems Research, MIT.

(46) [Mellor '66]

Mellor P.

A Review of Job Shop Scheduling.

Operational Research, Vol. 17, 1966, pp. 161-171.

(47) [Montanari '74]

Montanari U.

Networks of Constraints: fundamental properties and applications to picture processing.

Information Science, Vol. 7, 1974, pp. 95-132.

(48) [Moore '68]

Moore J.M.

An n-job, one machine sequencing algorithm for minimising the number of late jobs.

Management Science, Vol. 15, 1968, pp. 102-109.

(49) [Mott et al '88]

Mott D.H., Cunningham J., Kelleher G. and Gadsden J.A.

Constraint-Based Reasoning For Generating Naval Flying Programmes.

Expert Systems, Vol. 5, No. 3, August 1988, pp. 226-246.

(50) [Newell et al '57]

Newell A., Shaw J.C. and Simon H.A.

Empirical Explorations of the Logic Theory Machine.

In Computers and Thought (Editors Feigenbaum E. and Feldman J.)

New York, McGraw-Hill.

(51) [Newell et al '59]

Newell A., Shaw J.C. and Simon H.A.

Report on a General Problem Solving Program.

Proceedings of the International Conference on Information Processing (ICIP), 1959 (pp. 256-264), Paris: UNESCO HOUSE.

(52) [Nilsson '71]

Nillson N.J.

Problem Solving Methods in Artificial Intelligence.

McGraw-Hill, New York.

(53) [O' Grady '88]

O' Grady P.J.

Putting The Just In Time Philosophy Into Practice.

Kogan Page, 120 Pentonville Road London N1.

(54) [Orlicky '75]

Orlicky J.

Material Requirements Planning.

McGraw Hill.

(55) [Ow et al '88]

Ow P.S., Smith S.F. and Howie R.

A Cooperative Scheduling System.

In Expert Systems and Intelligent Manufacturing (Editor Oliff M.D.).

Elsevier Science Publishing Co., Inc..

(56) [Parunak et al '86]

Parunak H.V.D., White J.F., Lozo P.W., Judd R., Irish B.W. and Kindrick J.

An Architecture for Heuristic Factory Control.

Proceedings of the American Control Conference, Seatle, 1986.

(57) [Ramsay '88]

Ramsay A.

Formal Methods in Artificial Intelligence, Cambridge Tracts in Theoretical Computer Science 6.

Cambridge University Press.

(58) [Rit '86]

Rit J-F.

Propagating Temporal Constraints For Scheduling.

Proceedings AAAI-86, Philadelphia, Pa. (1986) pp. 383-388.

(59) [Sacerdoti '74]

Sacerdoti E.D.

Planning in a Hierarchy of Abstraction Spaces.

Artificial Intelligence Vol. 5 (1974) pp. 115-135.

(60) [Sacerdoti '75]

Sacerdoti E.D.

A Structure for Plans as Behaviour (Phd. Thesis).

Computer Science Department, Stanford University.

(61) [Sacerdoti '77]

Sacerdoti E.D.

A Structure for Plans and Behaviour.

New York: Elsevier, North-Holland.

(62) [Sadeh et al '88]

Sadeh N. and Fox M.S.

Preference Propagation in Temporal/Capacity Constraint Graphs.

Technical Report: CMU-CS-88-193, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa..

(63) [Sanborn et al '88]

Sanborn J.C. and Hendler J.A.

A Model of Reaction For Planning in Dynamic Environments.

Artificial Intelligence in Engineering, Vol. 3, No. 2, 1988, pp. 94-101.

(64) [Smith '83]

Smith S.F.

Exploiting Temporal Knowledge to Organise Constraints.

Technical Report: CMU-RI-TR-83-12 , The Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pa..

(65) [Smith et al '86a]

Smith S.F., Fox M.S., Ow P.S.

Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems.

AI Magazine Vol. 7 No. 4, Fall, 1986, pp. 45-61.

(66) [Smith et al '86b]

Smith S.F., Ow P.S., LePape C., McLaren B. and Muscettola N.

Integrating Multiple Scheduling Perspectives to Generate Detailed Production Plans.

Proceedings SME Conference on AI in Manufacturing (pp. , Long Beach, CA, September '86.

(67) [Smith '87]

Smith S.F.

A Constraint-Based Framework for Reactive Management of Factory Schedules.

Proceedings International Conference on Expert Systems and the Leading Edge in Production Planning and Control Charleston, South Carolina, May, 1987

(68) [Smith et al '87]

Smith S.F. and Hynynen J.E.

Integrated Decentralisation of Production Management: An Approach for Factory Scheduling.

Proceedings 1987 Symposium on Integrated and Intelligent Manufacturing, ASME Annual Winter Conference, Boston (Massachusetts, U.S.A.), December 1987.

(69) [Steele '80]

Steele G.L.

The Definition and Implementation of a Computer Programming Language Based on Constraints (Phd Thesis).

Technical Report: AI-TR-595, MIT, Cambridge MA..

(70) [Stefik '81]

Stefik M.

Planning With Constraints (MOLGEN: Part 1).

Artificial Intelligence Vol. 16 (1981) pp. 111-140.

(71) [Tate '75]

Tate A.

Using Goal Structure to Direct Search in a Problem Solver (Phd Thesis).

Machine Intelligence Research Unit, Edinburgh.

(72) [Tate '76]

Tate A.

Project Planning Using a Hierarchic Non-linear Planner.

Technical Report: 25, Department of Artificial Intelligence, University of Edinburgh U.K..

(73) [Tsang '87]

Tsang E.P.K.

The Consistent Labeling Problem In Temporal Reasoning.

Proceedings AAAI-87, Seattle, Washington (1987) pp. 251-255.

(74) [Ullman '76]

Ullman J.D.

Complexity of Sequencing Problems.

In Computer and Job-Shop Scheduling Theory (Editor Coffman E.G.).

Wiley and Sons, New York.

(75) [Valdes-Perez '87]

Valdes-Perez R.E.

The Satisfiability of Temporal Constraint Networks.

Proceedings AAAI-87, Seattle, Washington (1987) pp. 256-260.

(76) [Vere '83]

Vere S.A.

Planning In Time: Windows and Durations for Activities and Goals.

IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, pp. 246-267, May 1983.

(77) [Vilain et al '86]

Vilain M. and Kautz H.

Constraint Propagation Algorithms for Temporal Reasoning.

Proceedings AAAI-86, Philadelphia, Pa. (1986) pp.377-382.

(78) [Waltz '72]

Waltz D.L.

Generating Semantic Descriptions From Drawings of Scenes With Shadows.

Technical Report: AI-TR-271, MIT, Cambridge MA..

(79) [Whitin '57]

Whitin T.M.

The Theory of Inventory Management.

Princeton University Press.

(80) [Wight '81]

Wight O.W.

Manufacturing Resource Planning: MRP II - Unlocking Americas Productivity Potential.

Oliver Wight Ltd Publications Inc.

(81) [Winston '77a]

Winston P.H.

Artificial Intelligence (Second Edition) pp. 96-97.

Addison-Wesley Publishing Company.

(82) [Winston '77b]

Winston P.H.

Artificial Intelligence (Second Edition) pp. 119-122.

Addison-Wesley Publishing Company.