FLEXIBILITY IN A KNOWLEDGE-BASED SYSTEM FOR SOLVING DYNAMIC RESOURCE-CONSTRAINED SCHEDULING PROBLEMS

David W. Hildum

UMass CMPSCI Technical Report 94-77
September 1994

Department of Computer Science
University of Massachusetts
Amherst MA 01003-4610

EMAIL: *hildum@cs.umass.edu*

This research was sponsored, in part, by gifts from Texas Instruments, Inc., the National Science Foundation, under CER Grant DCR-8500332 and contracts CDA-8922572 and IRI-9208920, and the Office of Naval Research, under a Defense Advanced Research Projects Agency Grant (contract N00014-92-J-1698). Machine resources were supported by the Office of Naval Research, under a University Research Initiative Grant (contract N00014-86-K-0764). The content of this dissertation does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

FLEXIBILITY IN A KNOWLEDGE-BASED SYSTEM FOR SOLVING DYNAMIC RESOURCE-CONSTRAINED SCHEDULING PROBLEMS

A Dissertation Presented

by

DAVID WALDAU HILDUM

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial
fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1994

Department of Computer Science

To my parents, Donald C. and Priscilla A. Hildum.

ACKNOWLEDGMENTS

> Well, Pooh was a Bear of Enormous Brain
> (*Just say it again!*)
> Of enormous brain—
> (*Of enormous what?*)
> Well, he ate a lot . . .
>
> A.A. Milne, *Winnie-the-Pooh*

In the process of becoming one of those graduate students who took longer to finish than everybody else, I have benefited from the friendship and support of a large number of people.

Quite simply, none of this would have been possible without the support and guidance of my advisor, Dan Corkill. His ability to grasp and articulate ideas, his insights into the research process, his continuing faith in my abilities, and his friendship and sense of humor enabled me to complete this work. I am extremely fortunate to have been his student.

My thesis committee was very helpful throughout the course of my research, especially in encouraging me to address OR work on scheduling, which has enhanced the scope of this otherwise AI-centered dissertation. Individually, I want to thank Victor Lesser for having supported me financially for most of my years here, and providing many helpful suggestions throughout this project. His interests in various applications of this work have helped to identify a number of important directions for future research. Robbie Moll read (and *reread*) a number of chapters, helping me to clarify my ideas and clean up many an awkward grammatical phrase. Jim Smith was a great help in exposing me to past and present OR techniques and approaches for solving resource-constrained scheduling problems. I thank my entire committee for their assistance and their endless patience in guiding me through this project.

I am grateful to Texas Instruments, Inc., for their initial support of this work. Additional funding came from the National Science Foundation and the Office of Naval Research.

My thanks also to Steve Smith at Carnegie Mellon University for providing the OPIS benchmarking data and many hard-to-find papers.

A number of my friends warrant very special thanks for their extreme generosity over the years. David and Teri (and Brian and Joshua) Westbrook (the Westy's), Alice Julier and Zack Rubinstein (*Keepers of the Vortex*), Alan and Ruth Kaplan, Keith Decker, and Alan "Bart" Garvey, provided the vast bulk of my social existence, and fed me more times in the past ten years than I've fed myself. For their continual (and largely successful) attempts to drag me out of my office and preserve my sanity (often times against my will), I am eternally grateful.

Certain non-local friends, specifically Rick and Joanne Noel, Roger Segal and Sarah Kroloff, and Lewin and Sharon Wright, deserve my special thanks for their endless patience, in light of my frequent habit of holing up in my office and obsessing about my work for years at a time.

A great deal of programming accompanied this dissertation, and the help of certain people was invaluable throughout. I cannot thank Kevin Gallagher enough for his tireless assistance

over the years on an amazingly wide range of problems. David Westbrook and Zack Rubinstein provided valuable coding support for numerous parts of this project, and allowed me to bounce ideas off of them on many occasions. Keith Decker and Alan Kaplan provided assistance on problems ranging from the printing of Postscript files to the operation of my Macintosh.

A number of people in and around the Computer Science department at UMass, going way back, have been valuable sources of information, advice, inspiration, and friendship, specifically: Christy Anderson, Scott Anderson, Malini Bhandaru, Carol Broverman, Claire Cardie, Jody Daniels, Bruce Draper, Ed Durfee, Joe Hernandez, Eva Hudlická, Marty Humphrey, Philip Johnson, Sue Lander, Kathy McAdoo-Whitehair, Dorothy Mammen, Dan Neiman (who provided the headlines for Figure 6.1), Ed Pattison, Anil Rewari, Rob St. Amant, Tuomas Sandholm, Brian Stucky, Bob Whitehair, Jack Wileden, and the Alimonos family at Andy's Pizza (proud sponsors of Amalgams softball).

Within the CS department, I am forever indebted and eternally grateful to Renee Kumar and Sharon Mallory, who processed mountains of administrative paperwork for me during my long tenure in the department (and right up to the last minute). In addition, Priscilla Coe and Nancy Stewart helped to make this department a more enjoyable place to work. Glenn Loud and Judy Ruot of the RCF staff deserve thanks for their many efforts to keep my various machines running over the years. (And my special thanks to Barbara Gould for maintaining the candy jar in the RCF office!)

As an undergraduate computer science major at Brandeis University, I was part of an exciting educational environment that helped spawn my interest in the field of computer science and led me to pursue graduate study. My thanks to Jacques Cohen and Mitch Model for their initial help in preparing and supporting me in this endeavor.

Over the years, my family has provided safe havens for me to escape the endless daily grind of graduate school. Their continued patience and support over the years has been a great help to me. My very special thanks to Nan, Nikki and Stephen, Sue and Tilford (and Adam), Alan and Chris (and Benjamin), Tim and Sarah, Steve, Ted and Syd (and Nathan), Rob and Debi, Bob, and Grandma.

And finally, most of all, I want to thank my parents for their financial, intellectual, and emotional support throughout this long process. I could not have done this without them.

ABSTRACT

FLEXIBILITY IN A KNOWLEDGE-BASED SYSTEM FOR SOLVING DYNAMIC RESOURCE-CONSTRAINED SCHEDULING PROBLEMS

SEPTEMBER 1994

DAVID WALDAU HILDUM

B.A., BRANDEIS UNIVERSITY

M.S., Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Daniel D. Corkill

The *resource-constrained scheduling problem* (RCSP) involves the assignment of a limited set of resources to a collection of tasks, with the intent of satisfying some particular qualitative objective, under a variety of technological and temporal constraints. Real-world environments, however, introduce a variety of complications to the standard RCSP. The *dynamic resource-constrained scheduling problem* describes a class of real-world RCSPs that exist within the context of dynamic and unpredictable environments, where the details of the problem are often incomplete, and subject to change over time, without notice.

Previous approaches to solving resource-constrained scheduling problems failed to focus on the dynamic nature of real-world environments. The scheduling process occurs away from the environment in which the resulting schedule is executed. Complete prior knowledge of the order set is assumed, and reaction to changes in the environment, if at all, is limited.

We have developed a generic, multi-faceted, knowledge-based approach to solving dynamic resource-constrained scheduling problems, which focuses on issues of flexibility during the solution process to enable effective reaction to dynamic environments. Our approach is characterized by a highly opportunistic control scheme that provides the ability to adapt quickly to changes in the environment, a least-commitment scheduling procedure that preserves maneuverability by explicitly incorporating slack time into the developing schedule, and the systematic consultation of a range of relevant scheduling perspectives at key decision-making points that provides an informed view of the current state of problem-solving at all times.

The *Dynamic Scheduling System* (DSS) is a working implementation of our scheduling approach, capable of representing a wide range of dynamic RCSPs, and producing quality schedules under a variety of real-world conditions. It handles a number of additional domain complexities, such as inter-order tasks and mobile resources with significant travel requirements. We discuss our scheduling approach and its application to two different RCSP domains, and evaluate its effectiveness in each, using special application systems built with DSS.

TABLE OF CONTENTS

CHAPTERS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

OVERVIEW

This research describes a knowledge-based approach for solving the important class of *dynamic resource-constrained scheduling problems*. We have implemented our approach in a system called DSS: the *Dynamic Scheduling System*. DSS may be easily configured for solving a wide variety of resource-constrained scheduling problems within dynamic real-world environments, and we have performed an extensive evaluation of its effectiveness within a number of significantly different scheduling domains.

1.1 Introduction

The class of problems we address represents a real-world extension to the standard class of *resource-constrained scheduling problems* (RCSPs) [Davis, 1973]. The generic RCSP may be described in the following manner. Given a set of operation descriptions, a collection of requests to perform sequences of these operations using a finite set of resources, and a network of constraints governing the sequencing of the operations and the use of the resources, the goal is to produce, for each request, a schedule of specific operation sequences that includes explicit starting and finishing times for each activity and satisfies the entire set of relevant constraints. The quality of a completed schedule may be evaluated according to a variety of factors, including the degree to which the due dates of the orders are met, the total amount of time required to complete the operation sequences, and the utilization of the resources. The set of RCSPs subsumes the standard classes of job-shop, project and factory scheduling problems [Bellman *et al.*, 1982, French, 1982].

The class of dynamic RCSPs differs from the class of standard RCSPs in that the problem-solving activity involved in solving dynamic RCSPs is undertaken *within*, and therefore *in reaction to*, an *unpredictably changing* real-world environment. Solving dynamic RCSPs is therefore necessarily a reactive process, because the initial conditions within an environment are not guaranteed to hold throughout the scheduling process nor the execution of the schedule. Real-world environments generate such unpredictable events as resource failures, varying task completion times, and delayed, canceled, modified, or unexpected orders. These events have a substantial impact on the scheduling process. Owing to the changing nature of the environment, decisions made prior to the actual execution of the schedule frequently become obsolete and must then be subjected to later modification.

Because scheduling decisions must be made in response to changes within a dynamic environment, computational time is therefore at a premium, precluding a scheduler from pausing to reevaluate the entire current situation, throw out its existing schedule, modify its original assumptions, and start all over again from the beginning, every time an unexpected difficulty is encountered. A scheduler must instead be capable of adapting to the changing environment by first isolating the impact of unexpected events on its existing schedule, and then efficiently modifying the schedule in response. Great importance is therefore placed on the ability of the scheduler to understand fully the nature and impact of environmental events, react quickly to them, and develop its schedules in such a way as to maintain sufficient flexibility for responding to future difficulties.

The goal of our research has been to implement an intelligent scheduling system that is capable of solving dynamic RCSPs within realistic, broadly constrained, dynamic, and uncertain environments. In the process, we have developed a flexible, domain-independent scheduling approach that combines a well-informed, highly opportunistic control strategy with a least-commitment decision-making process. The opportunism in our approach allows the scheduler to adapt quickly to the changing environment, by focusing on the most tightly constrained subproblems at any point during the scheduling process. The least-commitment decision-making strategy helps to preserve flexibility within the developing schedule to preserve scheduling options for unsolved subproblems, and accommodate anticipated future difficulties [Stefik, 1980, Rickel, 1988]. A frequent, multiple-perspective analysis of the current state of problem solving helps to determine the proper focus of attention and inform the decision-making process [Rickel, 1988].

A secondary objective of this research has been to gain a better understanding of the domain-independent information common to real-world, dynamic RCSP domains, and explore how intelligent methods for representing and manipulating this information may be developed and utilized throughout the scheduling process.

In the following section, we provide a detailed description of the class of RCSPs, introducing some basic terminology, outlining some basic assumptions, and identifying the various computational difficulties presented by this particular class of problems. In Section 1.3, we discuss the special features of the class of dynamic RCSPs. Section 1.4 presents a discussion of the major themes of this research, and Section 1.5 provides an overview of our approach. The contributions of this research are highlighted in Section 1.6. Finally, Section 1.7 provides some useful aids for navigating the rest of the dissertation.

1.2 Resource-Constrained Scheduling Problems

The class of RCSPs (sometimes also called *constrained-resource project scheduling*) has been studied extensively in the field of operations research (OR) since the mid 1950's, and has begun to attract attention from the field of artificial intelligence over the past decade.

RCSPs represent an important class of problems owing to the broad range of real-world scheduling problems they encompass, coupled with the inherent complexity of the solution process. The benefits of an automated scheduling system for solving RCSPs center on the efficient production of high-quality schedules. Early approaches to solving small instances of the RCSP using mathematical programming techniques recorded savings in manufacturing costs resulting from shorter schedule makespans and the maximization of rewards from more on-time project deliveries. A generic scheduling approach designed to handle dynamic RCSPs would provide these same benefits to an even larger range of real-world problems.

1.2.1 Definition of Terminology

We begin with a description of the terms used to discuss RCSPs at a more detailed level. An *order* represents a request for the production or servicing of a particular *product*, which is accomplished through the execution of a sequence of activities called *operations* or *tasks*.[1] The sequence of operations designated for an order is called a *job*, and is instantiated according to the specifications of a *process plan template* which defines a partial ordering for all of the required tasks in the order. A *resource* is a machine or tool used to perform an operation.[2] A *resource plan* specifies a linear sequence of steps that must be taken by a resource to perform an operation. Figure 1.1 presents a visual summary of some of these definitions. We refer to a particular configuration of resources as a *layout*. Such a configuration includes information about the names, types, and locations of all available resources. Figure 1.2 provides some additional scheduling terminology (partially from [French, 1982]) that will be useful throughout the rest of this document.

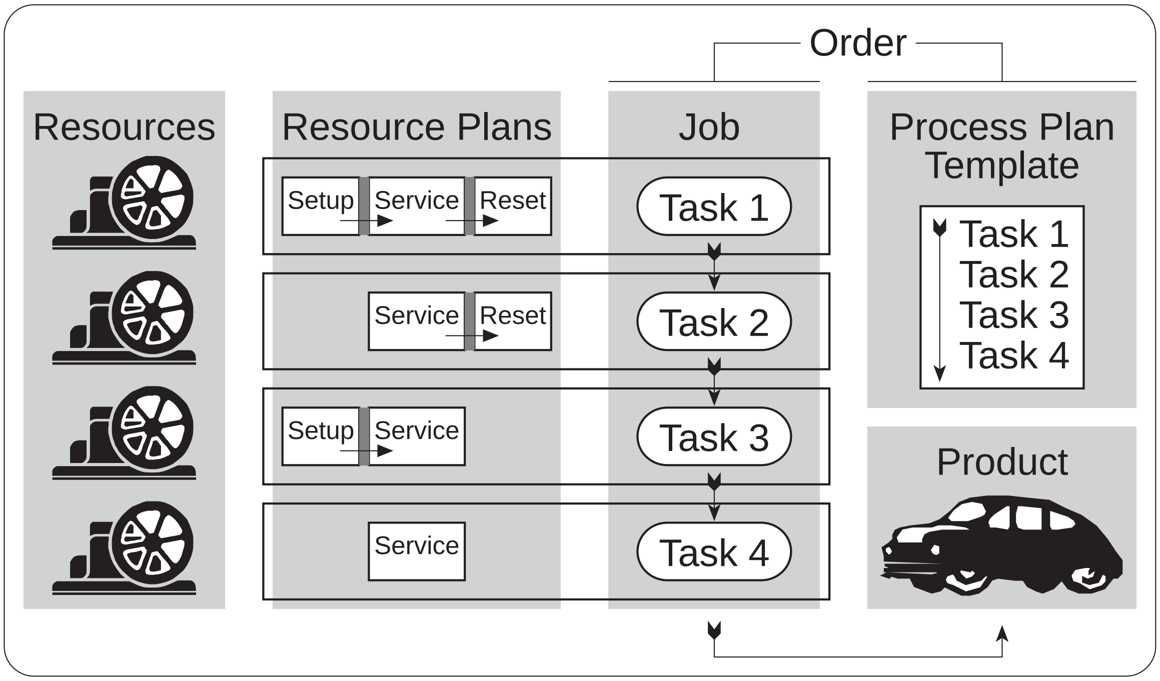


Figure 1.1. Resource-Constrained Scheduling Problem Legend.

The notion of *slack* (sometimes also called *float*) plays an important role throughout this research. For the purposes of this document, and because the term is widely interpreted within the field of knowledge-based scheduling, we now formally describe our use of the term. By performing a *Critical Path Analysis* [Kelley and Walker, 1959] upon the directed graph task sequence that comprises the job for an order O_i, and considering any additional relevant time bound constraints, we can determine, for each task T_{ij} in O_i, its earliest possible starting time (EST_{ij}) and latest possible finishing time (LFT_{ij}). These times define the lower and upper bounds for the region (or *allowance*) within which each product task may be scheduled. Note that this allowance is defined *only* in terms of the main servicing activity that is to be executed

[1] We will use these two terms interchangeably.

[2] Note that some operations may not require the services of a resource.

Processing Time
The amount of time required to perform an operation.
Allowance
The period of time within which an operation is intended to be performed.
Setup Time
The time during which a resource is set up to perform an operation.
Reset Time
The time during which a resource is reset after performing an operation.
Travel Time
The time during which a mobile resource moves between locations.
Idle Time
The time during which a resource is not used.
Ready Time
The desired time at which a job will become available for processing.
Due Date
The desired time at which a job will complete its processing.
Lead Time
The amount of time between the ready time and the due date.
Release Time
The actual time at which a job becomes available for processing.
Completion Time
The actual time at which a job completes its processing.
Flow Time
The amount of time between the release time and the completion time.
Make-Span
The maximum completion time for a set of jobs.
Delay
The difference between the release time and the ready time for a job.
Lateness
The difference between the completion time and the due date for a job.
Tardiness
Equals lateness when lateness is positive, 0 otherwise.
Earliness
Equals lateness when lateness is negative, 0 otherwise.

Figure 1.2. Resource-Constrained Scheduling Problem Terminology.

on a product. It does not include any other supporting resource activity (such as ancillary setup, travel or reset operations) that may be required as part of the resource plan for a resource to perform a particular task.

We assume, for each T_{ij}, the availability of the average expected processing duration d_{ij} for the main servicing activity component. We are now able to define the term SLACK$_{ij}$ for a task T_{ij} as:

$$(LFT_{ij} - EST_{ij} - d_{ij})$$

Slack is an explicit entity that resides within order schedules. It is distinguishable from the idle time that may exist within resource schedules as a byproduct of external fragmentation. Note that as the *EST* and *LFT* values for a task are modified, the value of SLACK for that task and possibly others will also change.

1.2.2 Basic Assumptions

The classical formulation of the RCSP makes a number of strict assumptions about operations, resources, and processing times. These assumptions tightly constrain the problem to facilitate its solution. In general, all required information is assumed to be available from the outset, and to remain fixed throughout the entire scheduling process. The order set and the levels of resource availability are static, and all jobs are released on time (as expected). Figure 1.3 presents a collection of standard RCSP assumptions that are applicable to our work, assembled from [Bellman *et al.*, 1982, Coffman, 1976, French, 1982, Niland, 1970].

1. *Assumptions on Jobs and Operations*
   - 1.1. The complete set of jobs is fixed and known at the outset.
   - 1.2. The time at which each job is released is fixed and known at the outset.
   - 1.3. A definite due date can be assigned to each job. A job must be completed by the beginning of the due date time period.
   - 1.4. Job arrivals occur at the beginning of a time period, and may begin executing in that time period.
   - 1.5. There may be precedence relations between operations within a job.
   - 1.6. There is no preemption. Operations may not be interrupted while executing.
   - 1.7. There is no cancellation. Each job must be processed to completion.
   - 1.8. Completed operations meet all specifications. Operations need never be repeated.
   - 1.9. In-process inventory is allowed. Jobs may wait for their next resource to be free.

2. *Assumptions on Resources*
   - 2.1. The complete set of resources is fixed and known at the outset.
   - 2.2. All resources are *non*-consumable.
   - 2.3. Resources never break down and are available throughout the scheduling period.
   - 2.4. Each resource can process at most one operation at the same time.
   - 2.5. All resources complete their operations satisfactorily. There is no notion of quality.
   - 2.6. Resources may be idle.

3. *Assumptions on Processing Times*
   - 3.1. The processing duration for each operation is deterministic and known at the outset.
   - 3.2. The processing speeds of the resources in a class are identical.
   - 3.3. Processing time includes setup time.
   - 3.4. Setup time is job sequence-independent. The time taken to adjust a resource for a job is independent of the job last processed.

Figure 1.3. General Assumptions for Classical Resource-Constrained Scheduling Problems

Increasingly constrictive assumptions are often used to constrain an RCSP to a more manageable, but also less practical, degree. For example, the sequencing of operations within jobs may be strictly controlled (to preclude simultaneous operations). In addition, each job may require servicing by every resource, and there may be only one of each type of resource available.

1.2.3 The Complexity of the Solution Process

Regardless of the strict assumptions about standard RCSPs, they remain a very difficult class of problems to solve. With the exception of some tightly constrained subclasses, the standard RCSP is NP-hard [Garey and Johnson, 1979].

To illustrate the large number of possible sequences (that is, resource assignments *without* starting and finishing times) that satisfy the constraints of a very basic RCSP, consider a problem consisting of n jobs, each requiring the services of a single machine in each of r resource classes. The total number of possible sequences for this situation is $(n!)^r$ (there are $n!$ orderings of the jobs on each resource class, and the orderings per resource class are independent of one another, hence the $(n!_1 \times n!_2 \ldots \times n!_r)$ total). With even moderate values for n and r, this number is large, and it grows quickly. If we provide m specific machines for each resource class, thereby allowing a choice among m machines to perform each operation, the total number of feasible schedules becomes $(m^n \times n!)^r$ (there are m^n ways to assign one of the m machines to each of the $n!$ job orderings per resource class). Again, even with moderate values for n and r, this number is quite large, and it grows quicker still. For example, given $n = 4$ jobs and $r = 5$ resource classes, and only $m = 1$ machines for each operation, there are $7,962,624$ possible sequences. Given a choice between two machines for each operation, that number rises to $8,349,416,423,424$. The process of assigning starting and finishing times to each reservation in a sequence adds still more significant factors into the equation.

These large sets of possible sequences do include a significant number of sequences that are impractical, such as those that use only one of the m machines from each of the r different resource classes to service all of the jobs. The imposition of temporal sequencing (and other kinds of) constraints serves to limit the number of sequences that may be used to produce feasible schedules. Nevertheless, the total number of feasible schedules for any moderately sized RCSP is still large enough to preclude the use of any kind of scheduling approach that attempts to exhaustively generate all feasible schedules (or even a significant portion of them), and then select the best from amongst them.

A significant difficulty brought about by a dynamic environment is the fact that the constraints on the problem, and the bounds produced by these constraints, are subject to change throughout the scheduling process, to the point where the actual set of feasible schedules changes during the course of producing a final schedule. Both the search space *and* the solution space are modified throughout the scheduling process, as new constraints are introduced and existing constraints are relaxed. Any reasonable approach to scheduling within dynamic environments must therefore adopt a method of progressively exploring the set of possible schedules by extending a feasible developing schedule in promising directions, thereby avoiding decisions that would lead to the development of infeasible or less desirable (low quality) schedules.

An important source of computational complexity involved in the process of solving RCSPs arises from the problem of trying to determine an optimal solution for a particular problem [Fox and Kempf, 1985]. Given the wide variety of metrics that exist for measuring the quality of a schedule, be it the degree to which order due dates are met, the lengths of the processing duration for jobs, or the degree of resource utilization, it is difficult to combine these individual metrics into a single value for determining overall schedule quality. A standard mathematical programming approach requires the definition of a single objective (or a functional combination of single objectives) to be either maximized or minimized. A goal programming approach ranks a collection of single objectives by priority and then invokes a multiple step process of satisfying each objective, in order of priority, using the previous solutions to constrain the continuing solution process, and thus giving each objective complete priority over the next.

The dynamic real world also presents a major obstacle to the problem of assessing schedule optimality [Fox and Kempf, 1985]. The real world does not behave predictably. Decisions based upon assumptions about real-world behavior are therefore not entirely reliable. A schedule that is determined to be optimal prior to its execution in the real world is optimal only to the degree that the real world behaves as expected during the schedule's execution. As a result, it is not practical to devote major effort towards achieving optimality in a schedule, because the true quality of a schedule may only be ascertained in conjunction with its execution in the real world. An apparently optimal schedule may be based on an unreasonable set of expectations about the real world, and may therefore be significantly less optimal when executed. Similarly, a schedule that is originally of less than optimal quality, but which contains some degree of built-in flexibility for dealing with unexpected world events, may actually turn out to be quite good upon its execution.

Simon uses the term *satisficing* to describe solutions that are "good enough" for approximate models of the the real world. Optimal solutions, on the other hand, are generally attainable only within simplified or imaginary domains [Simon, 1981]. The basis of heuristic search is to satisfactorily solve combinatorial problems while requiring less structure in the problem representation. The issue of producing optimal versus satisficing results in the solution of RCSPs is a common topic in OR, while most AI techniques tend to focus on the use of heuristic techniques for finding satisfactory or *quality* solutions.

In discussing past and current approaches to solving the RCSP, McKay claims that the dynamic characteristics of real-world scheduling environments render the bulk of existing solution approaches useless when applied to practical problems [McKay *et al.*, 1988]. The shop floor is inherently unstable, and its state changes very quickly. As a result, long-range scheduling tends to be kept to a minimum, because scheduling decisions so frequently become obsolete in a short period of time. In addition, shop floor dispatchers tend to make use of a wide range of (often unspecific) information and intuition, making the formalization of their decision-making processes essentially impossible.

1.2.4 Relaxed Assumptions

In our attempt to work with more realistic RCSPs, we have altered the standard definition of the problem to include the kinds of real-world details that contribute to the complexity of the solution process. Figure 1.4 presents the relaxed assumptions upon which our model of dynamic RCSPs is based.

1. *Relaxed Assumptions on Jobs*
   1.1. The complete set of jobs is variable and may not be known at the outset.
   1.2. Jobs may arrive ahead of, or behind schedule.
   1.3. The ready time and due date for a job are variable.
   1.4. Jobs may be cancelled at any time.

2. *Relaxed Assumptions on Resources*
   2.1. The complete set of resources is variable and may not be known at the outset.
   2.2. Resources may break down at any time and may not be available during the entire scheduling period.

3. *Relaxed Assumptions on Processing Times*
   3.1. An *approximate* processing duration for each operation is known at the outset.
   3.2. Resources in a class may have different processing speeds.
   3.3. Processing time *does not* include setup time.
   3.4. Setup time is job sequence-*dependent*. The time taken to adjust a resource for a job depends on the job last processed.
   3.5. Mobile resource travel time may represent a *significant* portion of processing duration.

Figure 1.4. Relaxed Assumptions for Dynamic Resource-Constrained Scheduling Problems

1.3 Dynamic Resource-Constrained Scheduling Problem Instances

Our research has focused on a number of different instances of the dynamic RCSP. In this section we formally describe three of these particular problem instances. The *airport ground service scheduling problem* (AGSS) makes heavy use of mobile resources, and includes a great deal of interconnection among the tasks to be scheduled. Some of this interconnectivity derives from the need to schedule significant amounts of travel time as part of the mobile resource reservations. Another source comes from compatibility constraints introduced by technological and travel-related requirements, which affect the pairing of resources, products and orders.

The *job-shop scheduling problem* represents a more generic RCSP that more closely resembles the classical RCSP formulation. While the AGSS problem must deal with significant complexity arising from the interconnectivity of the tasks, the job-shop scheduling problem is faced with a more compartmentalized problem consisting of mostly independent tasks, which helps to reduce the complexity of the solution process by allowing a scheduler to solve each task-specific subproblem without having a substantial impact on the remaining subproblems.

1.3.1 The Airport Ground Service Scheduling Problem

The *airport ground service scheduling problem* is defined as follows. We are given a master timetable of flights $\mathcal{F} : \{F_1, F_2, \ldots, F_I\}$, and a collection of resources $\mathcal{R} : \{R_1, R_2, \ldots, R_R\}$. Each of the flights requires the execution of some sequence of ground-servicing tasks from the task set $\mathcal{T} : \{T_1, T_2, \ldots, T_T\}$, depending on the particular type of ground service requested. Note that $\mathcal{F}$ may experience unexpected additions and cancellations during the course of scheduling. That is, the entire set of flights is not guaranteed to remain stable. A *job* comprised of the appropriate sequence of T_i's, $(T_{i1}, T_{i2}, \ldots, T_{iS})$, is instantiated for each flight F_i in $\mathcal{F}$. All flights have a desired ready time and due date (generally corresponding to flight arrival and departure times), and need not be serviced by every resource. The underlying goal in solving this problem is to produce a schedule of assignments of both processing times and resources to all of the tasks required by the F_i in $\mathcal{F}$, while satisfying all relevant constraints.

Each task T_{ij} entails a particular processing duration d_{ij}, often dependent on the particular product (aircraft) and resource involved. A task may or may not require the use of a resource, and there may be a choice of resource classes to perform a particular task. Each task consists of a `SERVICE` activity, and possibly some form of additional `SETUP`, `RESET` or `EN ROUTE` (*travel*) activity, depending on the selected resource. *Inter-order* tasks (such as the transfer of baggage or passengers between connecting flight aircraft) are used to connect individual flights. An inter-order task originates within a flight F_i and completes within another flight F_j $(i \neq j)$.

A set of technological constraints controls the pairing of the T_{ij}'s with the R_l's. A set of temporal sequencing constraints controls the order in which the T_{ij} may be sequenced within a job. Tasks may be executed serially or in parallel. Additional grouping constraints may require that certain T_{ij}'s be executed within the processing time bounds for some other T_{ik} $(j \neq k)$. The upper and lower processing time bounds for a task T_{ij} may be further constrained to be within a certain amount of time from either the ready time or the due date for F_i, or any flight F_j to which F_i is connected. Finally, certain pairings of tasks and resources may introduce further constraints on other pairings of tasks and resources.

Some of the airport resources are defined to be *mobile*, with each such resource class having a fixed traveling speed at which its members are able to move through the airport environment.

Travel times are dependent on the context within which a resource is used, as determined by the specific locations to and from which the resource must travel. Task processing durations are therefore *not* independent of the schedule. Mobile resource travel routes are unrestricted throughout the airport environment. All resources are subject to permanent failure, and may be periodically and temporarily shut down. Each resource may service only one aircraft at a time, and may not be preempted while executing an assigned task. Both resources and jobs may experience idle time during the course of a schedule.

The primary qualitative objective in the solution of the airport ground service scheduling problem is to limit tardiness across the entire set of flights $\mathcal{F}$, by minimizing the average tardiness cost per flight, and the percentage of tardy flights. That is, the due date for each flight should be met as often as possible. Let D_i be the due date for flight F_i, and let C_i be its scheduled completion time. Let P_i be the numerical ($> \mathit{0}$) priority of F_i.[3] The average tardiness cost can be calculated using the following equation:

$$\frac{\sum_{i=1}^{I} P_i \max(\,0,\ (C_i - D_i))}{I}$$

Note that there is no benefit to be derived from earliness. The percentage of tardy orders can be (partially) calculated by determining for how many flights in $\mathcal{F}$ the equation $C_i > D_i$ holds true. Other qualitative objectives may include the minimization of the total amount of setup activity scheduled for all resources in $\mathcal{R}$, and the minimization of the total amount of travel time scheduled for all mobile resources in $\mathcal{R}$. Maximizing the degree of resource utilization (by minimizing idle time) may also be desired.

As an example of the issues involved in the AGSS domain, consider the problem of servicing a *turnaround* flight at an airport. A gate must be secured to establish the location at which all of the other necessary servicing activities will occur. Various servicing vehicles must be secured to perform baggage loading and unloading, refueling, cleaning and restocking of the aircraft prior to its departure. These mobile resources must report from their previous locations to the location provided by the gate at which the aircraft is to be serviced. Time must be allocated for passenger deplaning and enplaning, to ensure that certain servicing activities are performed without passenger interference. Certain tasks must wait for others to finish before they may begin: arriving baggage should be unloaded before departure baggage is loaded. Furthermore, the loading of the departure baggage must not be initiated too much prior to the flight departure time, to allow for all such baggage to be assembled. Finally, some baggage, cargo and passengers may have to be transferred to other connecting flight aircraft. In this environment, due dates are targets that are intended to be exactly met, to achieve the smooth flow of passengers to flights. Flights that leave early result in missed flights, and late departures increase the discontent of those passengers forced to wait, and possibly miss other connections. A scheduler thus has more flexibility in scheduling the required servicing activities, because compaction of the schedule is not an option.

1.3.2 The Job-Shop Scheduling Problem

Another kind of real-world problem that we address in this work is the *job-shop scheduling problem.* In this case we are given a collection of orders $\mathcal{O} : \{O_1, O_2, \ldots, O_I\}$, a set of tasks

[3] In our implementation of the airport ground service scheduling problem, all flights are given a uniform priority of 1.

$\mathcal{T}$, and a collection of resources $\mathcal{R}$. The ready time indicates the time at which an order O_i is released to the shop for processing. The due date indicates the time at which the processing of the order is to be completed.

The assumptions and constraints described above for the more complicated airport ground service scheduling problem also apply here, with the following exceptions. In a job-shop, the resources are generally stationary, or else their travel times are insignificant. The product is maneuvered through the factory environment from machine to machine. Each task requires the services of a resource, and consists of a `SERVICE` activity and possibly a `SETUP` activity, depending on the prior use of the selected resource.

The qualitative objectives in solving the job-shop scheduling problem include those described above for the airport ground service scheduling problem. In addition, they may also include the minimization of the makespan for the entire schedule and the minimization of the flow (or work-in-process) times for the F_i.

As an example, consider a manufacturing environment where various products are produced via particular routings through the factory. Work areas containing groups of machines are spaced throughout the factory environment, at which the product, or parts of the product, are assembled or processed. Setup operations are required for tasks performed on machines whose prior use involved different kinds of activities. Orders received by the factory often have immediate due dates coupled with high priorities, allowing little or no slack time with which to provide flexibility for the scheduler, and resulting in substantial tardiness penalties. The earlier a job may be completed, the better.

1.3.3 The Transportation Planning Problem

The *transportation planning problem* covers a broad range of problems. In this case, we focus on the particular problem of moving cargo between geographically distributed locations with the help of a collection of mobile resources. We are given a set of *cargo movement requirements* $\mathcal{M} : \{M_1, M_2, \ldots, M_I\}$, a set of tasks $\mathcal{T}$, and a collection of resources $\mathcal{R}$. Each M_i specifies a request to move a particular piece of cargo (the product) from one location to another (possibly via some number of intermediate locations). The ready time indicates the time at which the cargo for a movement requirement is available for shipment from its originating location. A number of different transportation routings may be available, depending on the (bounded) number of intermediate locations that may be visited during shipment by the mobile cargo-transporting resources.

The assumptions and constraints described above for the job-shop scheduling problem apply here as well, with the following exceptions. A significant portion of the resources are mobile, cargo-transporting resources (such as boats, airplanes, trucks, or trains). The travel requirements for these resources are significant, and their traveling speeds are an important consideration in the scheduling process, because they may be sparsely distributed throughout the environment, and some of them are (significantly) faster-moving than others.[4] A shipment task consists of an `EN ROUTE` activity to allow the transporting resource to report to the originating location of the cargo, and a `SERVICE` activity representing the actual shipping

[4] Note that a more realistic model of this problem would allow fuel, crew, and other equipment costs to be associated with the various cargo-transporting resources to augment the reservation-selection process.

phase. Upon completion of a shipment task, the product will be located at its destination location.

The underlying objective in the transportation planning problem is to deliver each piece of cargo to its required destination by its desired due date. A penalty may be incurred for tardy shipments. There may also however, be a limit on how early a shipment may be delivered, and an earliness penalty may be applicable under such circumstances. Strict compaction is therefore not a desired feature of the schedule.

As an example, consider the problem of transporting cargo, such as food, from the east coast of the United States to various locations in Africa. Food cargo becomes ready for shipment at the various originating ports at various times. At each port, the food must be loaded onto either a boat or an airplane for shipment across the Atlantic. Shipments delivered by air arrive faster than those delivered by boat. The decision of which mode to use depends not only on the various compatibility constraints involving resource capacity and the due date for the shipment, but also the availability of boats and airplanes at each originating port. Note also that some shipments may require intermediate stopovers on their way over. Finally, there is clearly a penalty for tardiness in delivering food at the destination ports, but there is also a penalty for delivering food at a time that is too early for the food to be properly stored until it can be put to its intended use. A quality solution to this problem will result in the delivery of each food shipment to its assigned destination port no later than the desired due date and no earlier than some constant time prior to that date (probably depending on the type of food being shipped).

1.4 Research Issues

A number of important issues must be considered in the process of developing a method for solving dynamic RCSPs. The inherent unpredictability of real-world scheduling environments requires a problem-solver to be capable of operating under a significant degree of uncertainty. This existence of uncertainty requires the problem-solver to address the issue of flexibility throughout the entire decision-making process to accommodate unexpected developments. In this section, we will discuss both of these topics in greater detail.

1.4.1 A Sample Problem

To illustrate the degree of interconnectivity among the tasks, and the impact of uncertainty and the value of preserved flexibility on the scheduling process, we first present a simplified problem description from the previously described AGSS problem. For the purposes of this discussion, we will focus on a limited model of the AGSS domain, in which the required ground servicing is quite minimal. We will assume that each aircraft requires the same type of (turnaround) servicing, specifically the use of an assigned gate for docking purposes, the removal of all arriving passenger baggage, the transfer of all connecting flight baggage, and the loading of all departing passenger baggage.

Each separate baggage transfer, unloading, and loading operation requires the use of a single baggage truck, and a separate baggage transfer operation is required for each connecting flight. The baggage unloading operation involves the removal of the arriving passenger baggage from the aircraft, the movement of that baggage to the nearest airport baggage outlet, and the unloading of that baggage from the baggage truck. The baggage loading operation involves the loading of the departing passenger baggage onto the baggage truck at the baggage outlet nearest to the gate assigned to the departing flight, the movement of that baggage to the gate, and the loading of that baggage onto the departing flight aircraft. The inter-order baggage transfer operation involves the removal of all of the baggage for a particular connecting flight, the transferal of that baggage to the gate at which a connecting flight aircraft is docked, and finally the loading of that baggage onto the connecting flight aircraft.

In this simplified AGSS problem formulation, each arriving aircraft reports to its assigned gate shortly after landing at the airport. At the point of arrival, the unloading of all arriving passenger and transfer baggage may begin. After the unloading of all arriving passenger baggage has been completed, but no earlier than fifteen minutes prior to the departure time, the loading of all departing flight baggage may begin. After all of the baggage has been loaded, the aircraft may depart from its gate and proceed to a runway for takeoff. All transfer baggage must be unloaded from an aircraft before it departs from its assigned gate, and no transfer baggage may be loaded onto an aircraft until that aircraft has arrived at its assigned gate. Figure 1.5 provides a visual representation of this problem, showing the gate usage and baggage operations for a series of interconnected flights.

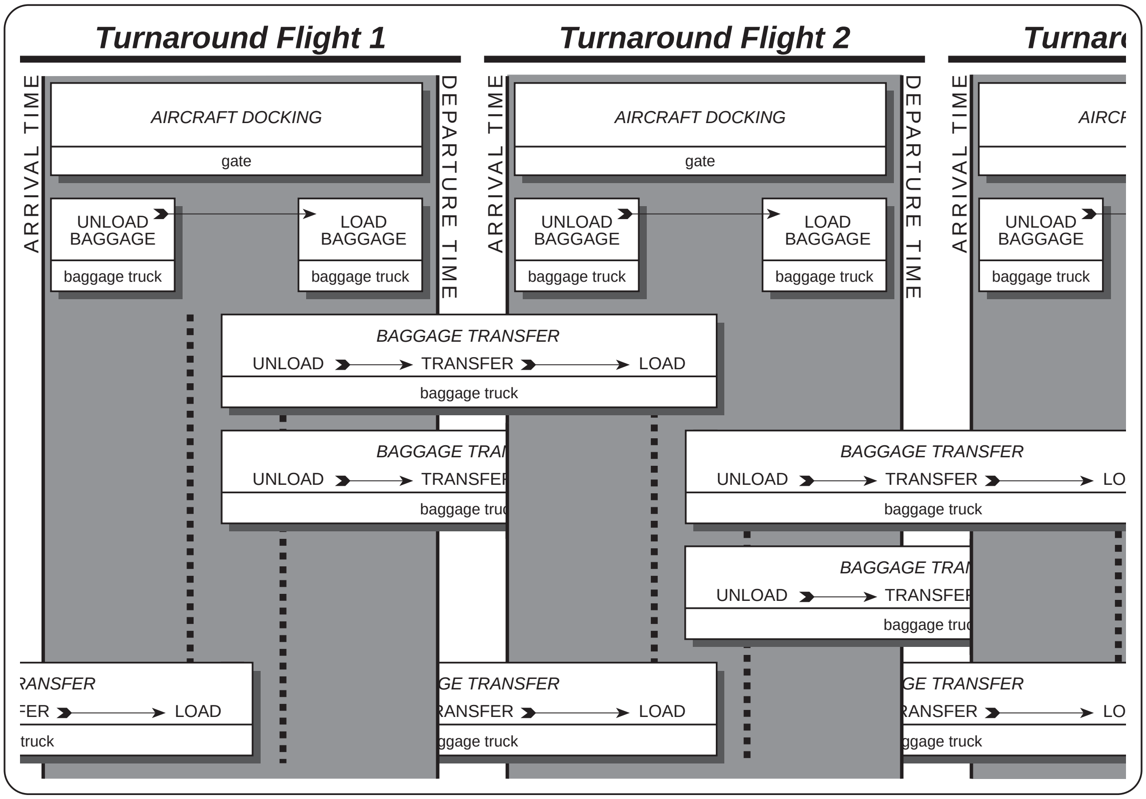


Figure 1.5. Aircraft Servicing Requirements for a Simplified AGSS Problem.

Note that each turnaround flight may be connected to any number of other turnaround flights, and that only a single baggage transfer task may connect the same two flights. The unloading of transfer baggage may begin at any time following the arrival of an aircraft at its assigned gate, as long as the process is completed before the aircraft departs. Similarly, the loading of transfer baggage may be completed at any time prior to the departure of the destination aircraft, so long as it is begun after the aircraft has arrived at its assigned gate.

We now discuss some of the features of real-world dynamic RCSPs that contribute to the difficulty involved in their solution.

1.4.2 Dynamic Environments

Dynamic RCSPs are real-time problems. Unlike the static classical domains where scheduling takes place as an essentially predictive batch process, dynamic and unpredictable domains force a scheduler to monitor and assess the impact of concurrent real-world events throughout the development of an initial schedule and during its execution. A scheduler designed for solving dynamic RCSPs must be able to make decisions about the future at the same time that its previous decisions are being enforced. It must be prepared to repair any previously made decisions that fail during execution. To further complicate matters, the scheduler may not be immediately informed about changes occurring within the environment, and the implementation of its repair actions may also be delayed. Finally, whenever the system pauses to think, the real world situation continues to deteriorate.

In the AGSS domain, we assume that a scheduler receives some degree of advance notice concerning the flights that are scheduled to arrive at an airport during a particular period of time, and that a complete schedule of activities may be produced prior to the arrival of any flights. Airport timetables, however, are subject to fluctuation, as flights are canceled, delayed, or otherwise modified in the course of execution. In a job-shop scheduling domain, factory orders are frequently submitted in batches, often with very little advance notice, precluding the complete production of a schedule before operations on the shop floor begin. Schedulers in these real-world situations are therefore frequently required to perform their scheduling duties in the midst of the on-going execution of a developing schedule, requiring them to incorporate the scheduling of new orders or the modification of previous decisions into their existing queue of scheduling tasks, whenever such developments are finally encountered.

Re-scheduling in a static domain is straightforward. The failure is noted, the problem is reformulated and re-solved using the original methods. In the real world, however, there is often not enough time to stop, think, and then start over again. Every moment of indecision complicates the problem at hand. Time acts as a continuously tightening constraint that limits scheduling options by preventing previous decisions from being reconsidered, and precluding other possible actions from being taken. It forces the scheduler to continuously reevaluate its queue of pending activities to ensure that urgent problems are addressed in time to avoid the delays that would result from last-minute decision making. A scheduler that is aware of a pending shortage of a certain kind of resource should take steps to accommodate the scheduling of the operations that may require that kind of resource, before the desired time for their execution appears.

The dynamic nature of the scheduling environment implies that initial expectations should not be heavily relied upon as a basis for making scheduling decisions. They should only be used to provide information to guide the scheduler until newer information is received from the environment. A scheduler must be able to react to new developments quickly and effectively so that its internal model of the world remains sufficiently informed and its decisions continue to reflect the actual state of the environment. It must also be able to incorporate responses to unexpected world events by determining the nature of each event, and its impact on the current decision-making strategy. It must then formulate an appropriate response to each event and insert the necessary resolution task(s) onto its queue of pending scheduling activities, based on the relative urgency of the response.

Although a scheduler may initially be provided with a set of expected orders, some of these orders may arrive either ahead of, or behind schedule. Anticipated orders may be canceled. New orders may be introduced without advanced notice. Ready times and due dates may be unexpectedly modified. Bottleneck conditions resulting from a high demand for specific resources at specific times may be anticipated based on a preliminary analysis of the expected orders, but the results are not sufficiently reliable owing to the possibility that the order set will not develop as expected, and resources may unexpectedly fail or vary in the amount of time they require to perform their scheduled tasks. This uncertainty complicates the overall process of understanding the current state of problem solving, by continually forcing a scheduler to reconsider the urgency of its pending scheduling tasks and adjust its present focus of attention.

The following two sections highlight two significant sources of environmental uncertainty, and describe their various manifestations.

1.4.2.1 Order Behavior

Whenever an order is received, the scheduler must determine the kind of service required, and the allowances within which the required tasks are to be performed. In our simplified AGSS problem, this involves the determination that basic turnaround ground servicing is required, and that given the flight's arrival and departure times, the gate and various baggage tasks (with consideration of the earliest starting time constraint for loading baggage) must be scheduled within specific time windows.

The scheduler must then determine the availability of the resources required for these tasks, evaluate the anticipated difficulty of securing these resources, and rearrange its current decision-making strategy according to this change in the previous demand for resources. Each flight in our simplified AGSS problem will require a single gate, two separate baggage trucks to perform the unloading and loading of baggage, and an additional baggage truck for each connecting flight. The availability of gates and baggage trucks, coupled with the times at which this new flight will require their services, provides valuable information to the scheduler about which of the new tasks to schedule first.

Whenever the ready time or due date for an order is modified, the relevant task allowances must be updated, using the sequence of steps described above. Each new order or change to an existing order thus has an impact on the current decision-making strategy employed by the scheduler in terms of introducing a new set of unsolved subproblems or altering some number of existing subproblems. The same situation occurs with order cancellations. In this case, the resulting change in resource demand requires the reevaluation of all unsolved subproblems currently requesting reservations for the same resources (during overlapping times) that were required by the tasks in the canceled order. If part of the schedule for a flight in our simplified AGSS problem has already been constructed at the point that a change in arrival time or outright cancellation notification is received, then an existing gate reservation is sure to require some form of modification. If a flight's arrival time is modified enough, then some of the existing baggage truck reservations may also be nullified owing to changes in the allowances within which they must be scheduled.

1.4.2.2 Resource Behavior

An unexpected resource failure affects both the existing schedule and the set of pending operations that still require resources of the class to which the failed resource belongs. The scheduler must first determine the set of all existing reservations that are invalidated by the failure, and then reactivate their previously satisfied subproblems to find replacement resources for the voided reservations. In addition, the scheduler must assess the impact of the resource failure on all subproblems for which the failed resource could have been secured. Resource failures have the potential to quickly and severely alter the bottleneck status for a resource class, thereby seriously affecting the current focus of attention for the scheduler.

In our simplified AGSS problem, the notification of a gate failure forces the scheduler to begin looking for new gate reservations for all of the flights previously assigned to the failed gate, and increase the current level of urgency assigned to each outstanding gate request to reflect the decrease in gate availability. The cancellation and subsequent re-satisfaction of nullified reservations may introduce tardiness into the schedule if flights whose gate reservations were canceled are now forced to wait for those other gates to become available. Such changes to new gates may also cause previously established baggage truck reservations to require modification owing to changes in their allowances or the locations where they are to be used.

Processing time is inherently dynamic and unpredictable in the real world. Different resources require different amounts of time to perform identical operations. A difference in operators, or even the particular time of day during which an operation is performed, may affect processing duration. (In the AGSS domain, weather conditions have a significant impact on the processing durations of the ground-servicing activities.) A delay in the performance of a task by a resource, for whatever reason, requires the scheduler to assemble the entire set of downstream tasks scheduled to use the delayed resource, and check each one to see whether the expected delay will cause that task to increase the flow time of its order. In the case where the completion times for an order will be increased, the scheduler may attempt to find alternate reservations for the tasks involved.

Upon notification of the delayed completion of a task by a particular baggage truck in our simplified AGSS problem, the scheduler will have to determine how many of the downstream reservations for that truck are affected by the delay. Some of the downstream reservations may be able to be shifted within their allowances without incurring any delay to their orders. Others may incur delay, requiring only that an existing gate reservation be extended to include the amount of delay. If such an existing gate reservation is not modifiable, then one of either the offending baggage truck reservation or the unextendable gate reservation will have to be canceled. And, of course, any extension to an existing gate reservation may require the modification of the baggage loading task reservation to ensure that it does not start any earlier than fifteen minutes prior to the newly delayed departure time.

1.4.3 The Importance of Flexibility

We now turn our attention to the actual mechanics of the scheduling process, specifically the issue of *flexibility*. Flexibility plays multiple and extremely important roles in providing the ability to handle the difficulties presented by dynamic real-world environments. Flexibility describes the ease with which a system is able to adapt to the current state of problem solving, as represented by the accumulated results of its previous decisions, its queue of remaining activities to perform, and the present conditions within the environment. In this section, we describe three different ways in which flexibility is involved in the process of solving dynamic RCSPs.

- Flexibility in the control of a problem-solver allows it to react quickly to newly received external information and internal events triggered by previous decisions. Frequent evaluation of current environmental conditions may be used to continually re-focus the attention of the system towards the most urgent subproblem at hand. Rigidity in the control of a system prevents it from adapting quickly to a changing environment.

- Flexibility may be built into, and preserved within, the developing solution by explicitly representing and reasoning about the slack available for each task to provide a wide range options for dealing with anticipated future difficulties. The lack of such flexibility acts to constrain the decision-making process and force a system to devote more effort to backtracking and other constraint relaxation activities when faced with future conflicts.

- Finally, flexibility in the decision-making process allows a problem-solver to be completely informed about the current state of the changing environment, and therefore better able to detect a wider variety of internal and external—and often subtle—developments that contribute to the identification of the most pressing subproblems to be addressed. This flexibility also provides the means to select the best-informed solutions to its subproblems. Rigidity in the decision-making process prohibits a system from considering the widest possible source of inputs at key decision-making points, relating to the control of the system and the actual solution-building process.

We now discuss each of these sources of flexibility in greater detail.

1.4.3.1 Fine-Grained Opportunism

The state of a problem-solving environment is altered, to some degree, by every external real-world event and every internal decision made by a problem-solver. The flexibility of a problem-solving system may be measured by the frequency with which it is able to pause, analyze, and adapt to the internal and external changes that occur during the course of problem solving.

An *opportunity* is a favorable circumstance that arises at just the right moment. Its occurrence is not predictable. In problem-solving situations, an opportunity presents an unanticipated chance to make progress towards a goal. To take advantage of opportunities as quickly as they develop, a problem-solver must have the ability to change course rapidly during its exploration of the developing search space. Sufficiently flexible systems pay close attention to the world and to the effects of their own decisions to facilitate quick reaction to both expected and unexpected developments. These systems are generally described as exhibiting *opportunism* in their control strategies, by continually redirecting their attention to the most urgent or promising issues at hand [Erman *et al.*, 1980, Smith *et al.*, 1990, Sadeh, 1991, Carver and Lesser, 1992]. Less flexible systems tend to wait until a previously determined and possibly extensive course of action has completed before they pause to update their current focus of attention. The impact of their own scheduling decisions may only then be detected as the result of a complete, and only periodically undertaken analysis of the present condition of the problem-solving environment.

At its lowest level, a RCSP may be viewed as a richly connected network of small individual subproblems, each representing a request for a resource to perform some operation within a particular period of time. In dynamic environments, where the changing conditions strongly influence the urgency of the various subproblems, it becomes extremely important for a scheduler to be able to modify its network quickly in response to unforeseen developments. For example, the reception of new orders leads to an increase in resource demand. The introduction of new reservations and the notification of resource failures both act to reduce resource availability and increase resource contention. The cancellation of existing reservations leads to an increase in resource availability. Finally, modifications to existing reservations brought about by the propagation of timing constraints imposed by various scheduling decisions cause the patterns of demand for resources (such as bottleneck conditions) to shift over time. All of these changes have a potentially substantial impact on the current focus of attention employed by the scheduler.

In most problem-solving systems, an overall problem is divided into separate subproblems to be handled atomically, in sequence. We use the term *granularity* to describe the extent of the subproblems handled by a system, and hence the level at which significant decisions are made in a problem-solving process. The granularity of the subproblems determines the frequency with which a problem-solver is able to modify its current state and reevaluate its focus of attention. The granularity of a subproblem is distinct from the granularity of its solution. Any subproblem, regardless of its extent, may be solved in either full detail or not.

A distinction is made between *large-grained* and *fine-grained* approaches. A large-grained scheduling approach might divide the overall scheduling problem into individual order-scheduling subproblems, so that each order is addressed as a single scheduling activity. The solution of each of these order-scheduling subproblems would result in the assignment of both processing times and resources to all of the T_{ij}'s in a job for an O_i. A large-grained approach thus reduces the degree of reactivity that may be achieved by a system by forcing it to wait to complete an entire order schedule before moving on to the next scheduling activity.

Returning to our simplified AGSS problem, we may quickly define two large-grained approaches for its solution. An order-based process would, as mentioned above, address the overall problem in terms of the individual flights. For each flight, the scheduler would produce a complete schedule of gate and baggage truck usage satisfying the timing constraints as controlled by the arrival and departure times (and assorted processing durations). The opportunity to shift the focus of attention to a different flight must wait until all of the tasks for a previous flight have been completely scheduled. On the other hand, consider a resource-based process that would address the overall problem in terms of the individual resource classes. In this case, all of the requests for a particular resource class would be scheduled before the requests for other resources. For example, we might choose to secure all of the gates first, and then proceed to secure all of the necessary baggage trucks. The opportunity to adjust the focus of attention to a different resource class must wait until all of the requests for the previous resource class have been satisfied. The resource-based approach results in a generally larger-grained approach than an order-based approach, considering that the number of orders is generally larger than the number of resource classes.

At the other end of the spectrum, a fine-grained approach, as is often exhibited by heuristic-based scheduling systems, might divide the overall scheduling problem into individual task-scheduling subproblems, such that each problem-solving step would involve the assignment of both processing times and a resource to a single task. Only a single task would thus have to be completely scheduled before the scheduler would be able to reconsider its present course of action. The finer the granularity, as indicated by the scope of the subproblems, the quicker the scheduler is able to react to the changing environment. A fine-grained representation of the subproblems thus produces more frequent opportunities for re-adjustment of the current scheduling strategy [Sadeh, 1991].

A fine-grained approach to handling our simplified AGSS problem would address each individual ground-servicing operation as a separate subproblem, thereby breaking up all flights into their individual resource-requesting components. Each flight therefore produces three or more task-level subproblems, namely a gate request for docking the aircraft, a baggage truck request for unloading baggage, another baggage truck request for loading baggage, and as many more baggage trucks for transferring baggage as there are connecting flights. This fine-grained approach allows a scheduler to select the most urgent task-level subproblem, solve it, and then quickly look for the next scheduling activity. Because of the fine granularity, the impact on the current state of problem solving that results from the solution of a task-level subproblem is quite limited, thereby minimizing the need for major adjustments in the current focus of attention. While the assignment of a gate and two or more baggage trucks to a flight in a single decision-making step introduces a large number of constraints into the developing search space, the impact of assigning a single baggage truck to a ground-servicing task is considerably smaller.

Knowledge-based scheduling systems have run the range of granularity, from large-grained order-based approaches, to fine-grained, task- or activity-based approaches. The issue of granularity plays an important role in the problem-solving process, by defining the scope of the decisions that are made by a problem-solver. In large-grained situations, such decisions may be far-reaching, thereby introducing significant additional constraints into the search space, or otherwise affecting a large portion of the overall problem. This issue is of particular importance in the context of reactive scheduling, where the responsiveness of a system depends on the speed with which it is able to change its current focus of attention and adapt to new information. In large-grained systems, the chance to move to other, more important subproblems is hampered by the need to finish working on the current, and potentially substantial subproblem, while in fine-grained systems, the small size of the individual subproblems permits a problem-solver to quickly move from one portion of the search space to another.

When coupled with a fine-grained decision-making process, an opportunistic problem-solver is able to quickly change its course of exploration through the developing search space, thereby making itself more effective in handling the inherent uncertainty of complex real-world problems, and thus more flexible in dealing with unexpected events.

1.4.3.2 Maintained Slack Time

Our second measure of flexibility involves the degree to which a problem-solving system organizes its decision-making process to avoid over-constraining unsolved subproblems. Successful reactive problem solving in dynamic environments requires flexibility for dealing with unexpected events. As conflicts arise, there must be room for maneuvering to adapt existing plans without greatly affecting the entire developing solution. Problem-solvers that preserve the means of satisfying unsolved subproblems provide future flexibility that may help prevent the difficulties that can arise when attempting to resolve increasingly tighter-constrained situations. Those systems that do not are more likely to have to perform expensive backtracking or costly constraint relaxations to solve increasingly difficult problems.[5]

In this section, we will discuss two different ways that flexibility can be maintained within a developing solution for adapting to unexpected events. The first is a *least-commitment* approach to problem solving that attempts to minimize the introduction of constraints at every decision-making step [Stefik, 1980]. The second is an equally conservative, *worst-case* reservation policy for handling the assignment of mobile resources under conditions of uncertainty. A least-commitment approach can be implemented within a scheduling process by incorporating flexibility, in the form of slack time, into the developing order schedules. In the latter case, flexibility is incorporated within the developing order schedules, but it comes from increased processing durations built into mobile resource reservations. In both cases, the preserved flexibility provides maneuverability for satisfying future reservation requests and assorted conflicts without overly constraining the decision-making process.

We now discuss both of these methods for preserving scheduling flexibility.

1.4.3.2.1 Least-Commitment Decision Making

The least-commitment decision-making approach is similar to the *lazy evaluation* technique employed in some programming languages [Field and Harrison, 1988]. A lazy evaluation approach allows the evaluation of certain expressions to be postponed until absolutely necessary, thereby putting off potentially time-intensive tasks until later to focus instead on more important or immediate work. The least-commitment approach may be thought of as putting off certain decisions that are expected to be easy to make until after other, more difficult (highly constrained), subproblems have been solved.

In many scheduling environments, there is no benefit to completing an order ahead of its desired due date. In fact, there may be a penalty associated with early orders, in the form of additional storage costs. It therefore makes sense to use any existing slack time within an order to help implement a least-commitment decision-making approach that improves the ability of the scheduler to react effectively to a dynamic environment. The incorporation of slack time into the developing schedule preserves the options for resolving future conflicts while also localizing their extent.

A least-commitment scheduling approach for exploiting slack time may operate by avoiding the opportunity to compact the developing order schedule at every decision-making point. Consider the introduction of a resource reservation into the developing schedule. While some constraint propagation is surely required, any remaining slack can be left with the task for which a resource has been secured. The new resource reservation has thus avoided introducing all but the absolutely necessary constraints on the remaining subproblems. The degree of difficulty associated with any related, unsolved subproblems remains relatively unchanged. In addition, this slack may now be used to absorb schedule modifications warranted by future conflicts, possibly without seriously impacting the rest of the schedule.

One particular difficulty presented by this approach involves the detection of conflicts. Because of the increased flexibility represented by the slack time, it becomes harder to determine the true level of resource contention that exists at any point. As a result, enhanced mechanisms

[5] The cost of constraint relaxation consists of a computational expense and a decrease in schedule quality.

are needed for reasoning about the current and anticipated states of resource demand and allocation.

Finally, there is a tradeoff associated with this particular type of least-commitment approach, and it involves the sacrifice of shorter work-in-process (WIP) times to preserve flexibility in the scheduling process. There are certainly RCSPs that have the minimization of WIP times as an important scheduling objective. In such cases, the preservation of slack time may be less helpful towards this end. But reactive processing in dynamic environments depends on the existence of flexibility for adapting to unexpected events and other conflicts, and if slack is available for this purpose, its utilization is certainly warranted.

1.4.3.2.2 Worst-Case Mobile Resource Reservations

Introducing further constraints onto unsolved subproblems limits the number of candidate resources that may be used for their satisfaction, and makes their satisfaction increasingly difficult to achieve, thereby limiting scheduling flexibility. In this section, we describe a situation where the ability to reason with incomplete information and incorporate explicit flexibility into the developing schedule allows a scheduler to solve its subproblems in the order dictated by current environmental conditions, instead of according to some predefined rigid task ordering.

Not all factory resources are stationary machines to and from which products are maneuvered in the course of being serviced or produced. In many factories, a portion of the resources are mobile vehicles that must report to the location of the product they are assigned to service. Such locations are often work areas provided by stationary resources. The movement requirements of mobile resources are an important consideration in the scheduling process, owing to either the magnitude of the distances involved, the traveling speed of the mobile resource, or a combination of the two. Regardless, it is necessary to know the location where a resource is required to perform a particular operation, and from what location it will come. This information, coupled with the individual travel speed for the resource, helps to indicate how much time should be allocated to the resource to perform a servicing task that requires movement through the environment. The selection of a work area constrains the selection of all mobile resources that must report to it, just as the selection of a mobile resource constrains the selection of its required work area. To preserve scheduling flexibility, it is important that the reservation of mobile resources be made in a way that avoids over-constraining future attempts to find work areas for them to use.

Consider the situation where the entire context for an individual subproblem has not been established at the time that a decision involving it must be made. For example, suppose the request for a mobile resource is addressed before the location at which it is to be used has been determined. This situation can be illustrated by returning to our simplified AGSS problem and the process of securing baggage trucks to work at the gates.

If a gate is assigned to a flight before any of the flight's required baggage trucks have been secured, then the reservations for these baggage trucks (which must report to the gate) can be constructed to include exactly enough time to get from their previous locations to the location of the secured gate. But suppose that the scheduler desires to secure the baggage trucks first, owing to their relatively short supply. In this case, a worst-case mobile resource reservation approach results in the construction of baggage truck reservations that include enough time to get from wherever a particular baggage truck may currently reside within the airport, to *any* of the gates that could be assigned in the future. The process of selecting the gate is therefore left unaffected by the decision, because no additional constraints have been imposed as a result of the introduction of a baggage truck reservation.

The decision-making process for such situations could be orchestrated to ensure that all scheduling decisions are made under conditions of complete certainty. For the AGSS example above, this would require all gates to be secured prior to the assignment of any mobile resources (baggage trucks). This approach presents a problem, however, when the mobile resources are in shorter supply than the stationary resources. In this case, a top-down reservation process introduces significant constraints on the mobile resource subproblems that makes them even more difficult to solve.[6] Such an approach clearly violates the principle of maintaining flexibility in the opportunistic control of the system as described in Section 1.4.3.1. The focus of attention should depend on the nature of the scheduling environment, instead of the task hierarchy for the domain.

The issue of flexibility comes into play when mobile resources are allowed to be secured before their required locations have been determined, to permit a greater degree of flexibility in the control of the scheduler. In this case, the inclusion of sufficient maneuvering room as part of all mobile resource reservations made under uncertain conditions preserves the scheduling options for all related unsolved stationary resources. Returning to the AGSS domain, if all baggage truck reservations made prior to any gate assignments include enough time to get to any possible gate candidate, then the process of securing the gates in the future remains unaffected by the baggage truck assignments. An important additional mechanism associated with this process is responsible for handling the refinement of these worst-case allocations as their contexts are further established. The DSS refinement mechanism is described in Section 4.2.6.

While this approach does not explicitly violate the principle of least-commitment from the perspective of the developing order schedules, it does over-constrain the developing resource schedules by allocating more than enough travel time for uncertain mobile resource reservations in a worst-case fashion. But in dynamic environments, the ability to adapt to unexpected developments without severely disrupting the existing schedule warrants the conservatism exhibited by this approach.

1.4.3.3 Detecting Change in a Dynamic Environment

Our third measure of flexibility derives from the range of perspectives used by a problem-solving system to inform its decision-making process at both the control and domain levels. A problem-solver finds a solution by exploring a search space of partial solutions. The efficiency with which it explores this space directly impacts the amount of effort involved in finding a feasible solution. If it fails to fully understand the texture of the search space and instead depends on an uninformed decision-making process to guide its exploration of the space, then it runs the risk of making frequent misdirected moves within the space that will require it to backtrack in the future or relax constraints to correct itself. This backtracking will impede the progress towards finding a feasible solution.

[6] When the mobile resources are in greater supply, the scheduler can make a decision about a stationary resource and accept the introduction of constraints on any related unsolved mobile resource subproblems, because of the lower urgency for the mobile resource tasks as evidenced by the current and anticipated levels of resource demand.

The wider the range of perspectives on the search space that are considered by a problem-solver, the better it is able to detect important and possibly subtle aspects of the current state of problem solving, and the greater the degree of flexibility it may exhibit in terms of reacting to those developments at the appropriate times. Systems with this ability are described as making use of *multiple perspectives* in their decision-making process. If the perspectives brought to bear on the current state of problem solving supply only a limited view of the state of the world, then a system is forced to act on the basis of an incomplete understanding of the world, and will tend not to detect the appearance of important developments that should be addressed as quickly as possible to avoid turning them into more difficult future conflicts.

Determining task sequences and securing resources in any kind of scheduling problem requires a scheduler to reason about the conflicts that arise in the attempt to satisfy the various constraints among the resources, tasks and orders in the domain [Smith *et al.*, 1986b]. These conflicts arise when the satisfaction of a particular constraint or constraints affects the ability to satisfy other constraints.

1.4.3.3.1 Consulting Multiple Scheduling Perspectives

The process of solving an RCSP comprises two separate *planning* and *scheduling* phases. The planning phase is responsible for selecting and ordering a collection of activities to perform some service or produce some product. The scheduling phase combines the classical OR problems of *sequencing* and *timetabling*, that is, determining an ordering for a collection of activities, and assigning specific start and finish times to each activity in a sequence [French, 1982, Noronha and Sarma, 1991]. The planning phase is often treated as a lookup process where the sequences of required activities for different classes of jobs are prescribed by the domain. A lookup approach limits the overall generation of conflicts, because the scheduler does not have to generate a feasible process plan for each different order from a basic set of constraints.[7] The scheduling phase involves the solution of both the sequencing and timetabling problems together, thereby permitting conflicts to arise from the interaction between the separate sets of sequencing and timetabling constraints.

The sequencing problem is concerned with the ordering of activities on the available machines. The relevant constraints in this problem involve the assorted operational requirements of the individual machines. Activities may be sequenced in such a way as to avoid the need for costly machine setup or retooling operations, to help increase resource utilization. The satisfaction of these constraints impacts the degree to which the final schedule satisfies a machine-centered perspective. The timetabling problem, on the other hand, is concerned with the assignment of specific starting and finishing times to activities. The constraints relevant to this problem involve the time bound requirements imposed on each activity by its corresponding order. The satisfaction of these constraints helps to produce a schedule that reflects an order-centered perspective. Conflicts between sequencing and timetabling constraints occur when the desired sequencing of a particular activity violates the time bounds imposed by the relevant timetabling constraints. Early recognition of such conflicts by a scheduler reduces the need for future backtracking. For a scheduler to reason intelligently about such conflicts and develop the appropriate means for their resolution, it must consider

[7] Note, however, that a lookup approach does limit the ability of a scheduler to adapt to dynamic environments where the response to unexpected events may require the alteration of existing process plans.

the entire set of constraints that enter into each conflict, and understand the full context of the constraints involved.

At any decision-making point, the extent of the information brought to bear on the analysis of the problem and the formulation of its solution defines a particular perspective on the problem that may be used to solve it [Smith *et al.*, 1986a, Smith *et al.*, 1986b]. This degree of perspective provides an indication of how well a problem may be expected to be solved, based on the nature of the information considered in the solution process. If the degree of perspective is fully appropriate to the problem at hand, then a well-informed, and therefore higher quality solution should be expected. If, however, an inappropriate perspective is brought to bear, it should be expected that any solution will be misinformed, and therefore of lower quality. When the degree of perspective used is insufficient for the problem at hand, important aspects of the problem are not considered, and inevitable conflicts may therefore not be prevented.

In RCSPs, both resources and operations have constraints that govern their use and interaction. Resources may require periodic shutdown for maintenance purposes, or prefer that their usage be balanced, or that costly setup activities be minimized. Some operations must be completed before other operations may begin, and some operations may have preferences for particular resources. If a problem is organized in such a way as to focus on satisfying only a limited set of constraints at an important decision-making point, then the ability of a scheduler to address properly the problems that arise out of conflicts with other constraints is lessened.

For example, if the assignment of resources to tasks is performed solely from the viewpoint of the orders involved, then the constraints on the orders are implicitly given priority over the constraints on the resources. The result is sub-optimal resource allocation. Constraints such as balancing resource usage and minimizing resource setup costs are sacrificed to increase the quality of the individual order schedules. If, however, the assignment of resources is performed from the viewpoint of the resources themselves, then it is the constraints on the resources that receive priority instead. The result is that while the utilization of the resources is maximized, orders may miss their due dates or spend excessive amounts of time waiting for the use of particular resources. The proper consideration of all relevant constraints allows the scheduling process to more fully understand the nature of the conflicts that will inevitably occur, make more informed scheduling decisions, and produce better quality schedules.

In our simplified AGSS problem, the reliance on a solely order-based perspective would tend to produce schedules that would unload arriving baggage as close as possible to flight arrival times, and load departing baggage as close as possible to flight departure times, baggage truck supply permitting. An additional result of such an approach, however, would be that the baggage truck schedules would be skewed to favor the orders, thereby producing potentially fragmented usage that would make further order-centered scheduling gradually more difficult, because of the reduced availability of large blocks of available resource time. Reliance on a solely resource-based perspective would minimize such fragmentation by carefully scheduling baggage truck activities in such a way as to increase their utilization, but the effect on the scheduling of the assorted baggage maneuvering tasks would be seen in the failure to get these tasks scheduled as early or late as would be preferred by the flights.

Dynamic RCSPs introduce another dimension of scheduling perspective that involves real-world timing constraints on the subproblems that must be solved by a scheduler. As real-world events occur, and scheduling decisions are executed, time steadily marches on, continuously affecting the urgency of all pending scheduling tasks. The decision-making process must therefore consider, not only the various sets of time-independent scheduling perspectives involved in each particular situation, but also the relative urgency of the situation as defined by the current real-world time.

Highly constrained subproblems involving requests for bottlenecked resources are urgent, as are conflicts related to unexpected resource failures. A scheduler working within a dynamic environment must exhibit a broad understanding of urgency through consideration of a variety of perspectives on the state of problem solving to manage the smooth integration of all scheduling tasks. It should be capable of assessing quickly the urgency of each new scheduling task, and determining the importance of attempting to handle that task ahead of all others. This determination requires an evaluation of the relevant portions of the current state of problem solving. Such careful consideration of all important scheduling perspectives enables a scheduler to integrate effectively the performance of both predictive and reactive tasks.

For example, consider a reactive scheduling situation involving the failure of a resource in high contention, which voids a number of existing reservations. The already high level of contention for the affected resource class will increase as a result of the failure, thereby increasing the difficulty that may be expected in the process of re-satisfying all of the reactivated subproblems. Under ordinary predictive circumstances, these newly reactivated subproblems would increase in urgency owing to the greater contention resulting from the resource failure. A resource-based perspective on the state of problem solving would strongly contribute to this increase. Now, consider the sudden reception, immediately following the notification of the resource failure, of a batch of new orders with very early due dates. Even if these new orders require resources that are in abundant supply, the various subproblems they create may warrant a higher level of urgency than some of the failure-reactivated subproblems because of their more rapidly impending due dates, regardless of the levels of contention for their required resources. The existence of the real-world clock in this situation, a factor of the reactive operation of the scheduler, may force the scheduler to address the problem of satisfying the newly created subproblems ahead of the reactivated subproblems, because of the potential for substantial delays to be incurred by the new orders if their subproblems are not immediately addressed. The real-world clock perspective may therefore occasionally take precedence over other resource- and order-based scheduling perspectives, depending on the current state of problem solving.

1.4.3.3.2 Variable-Ordering and Value-Ordering Heuristics

Previous work in solving constraint-satisfaction problems has represented constraint networks using graph structures, where the nodes represent the variables in the problem, and the arcs connect these variables to the sets of possible values that may be used for their satisfaction [Dechter and Pearl, 1988]. The solution process proceeds by alternately selecting a variable to initialize, assigning it a value, and propagating the constraints imposed by the assignment. When a dead end is encountered, because of the lack of a feasible value for a variable, some form of backtracking must be invoked. The solution process completes when all variables have been assigned values, or an infeasible situation is recognized. The heuristics designed for informing variable and value selection phases are called *variable-ordering* and *value-ordering* heuristics [Sadeh and Fox, 1990].

The amount of backtracking required as part of the solution process may be lessened by means of an informed ordering of the variable and value selection phases. This ordering, which is achieved through a periodic analysis of the search space, organizes the solution process so that the variables that are most heavily constrained, that is, the variables with the fewest possible satisfying values, are handled as early as possible in the scheduling process [Haralick and Elliott, 1980]. Each value-to-variable assignment acts to further constrain the remaining satisfaction possibilities for the outstanding (uninitialized) variables. By making assignments as early as possible to those variables having the fewest possibilities for satisfaction, the need for costly backtracking is lessened.

The AGSS domain illustrates a number of variable-ordering perspectives that can be used to determine the relative urgency of scheduling subproblems. If gates are in greater supply than baggage trucks, given the number of outstanding requests for each, then the process of finding a gate should be expected to involve less difficulty than finding a baggage truck. Resource supply is determined by a number of factors. An airport provides a certain number of baggage trucks and gates, but the occurrence of resource failures modifies this supply. In addition, some resources cannot be used for certain tasks owing to compatibility issues, so the pool of resources that can satisfy a request may be further constrained. The amount of slack time accompanying a resource request influences its urgency by providing an idea of how many possible reservation times may be considered for a particular resource. If one baggage task has more slack than another, the one with less slack should receive greater priority because there are fewer times at which its request may be satisfied. The size of the expected processing duration for a task plays a role in that shorter reservations have less of an impact on the resource schedules. If one baggage task is expected to take more time than another, the larger of the two should be handled first, to take advantage of whatever large blocks of available resource time still exist. The role of time in the determination of subproblem urgency gives priority to tasks with lower time bounds on their allowances (earlier start times), such that baggage trucks for baggage unloading tasks should be secured prior to securing baggage trucks for the following baggage loading tasks. Finally, it is often the case in dynamic environments that some kinds of scheduling tasks may take temporary precedence over others. For example, when a sudden resource failure voids a number of existing reservations, the need to re-satisfy those canceled reservations may be more important than satisfying other downstream scheduling tasks. These variable-ordering perspectives are summarized in Table 1.1.

The AGSS domain also illustrates a number of value-ordering perspectives that can be used to inform the domain-level decision-making process. In our simplified AGSS problem, baggage trucks are in demand because each flight includes at least two baggage maneuvering tasks as part of its required ground-servicing activities. If the supply of baggage trucks is sufficient to handle such demand, then the scheduler need not take special care in the assignment of baggage trucks to explicitly compact their schedules to preserve large blocks of available time (as with a resource-based approach). In such cases, the scheduler may decide to concentrate on the satisfaction of the order preferences instead (using an order-based approach). If the demand for baggage trucks gets too high, the scheduler may begin attempting to produce baggage truck reservations that avoid introducing fragmentation into their resource schedules, and force the baggage operations to be scheduled at times that are not optimal for the flights. Reservations should be introduced in such a way as to minimize the upset to the existing schedule. Preemption and right shifting of existing reservations should be avoided until absolutely necessary, as should any introduction of tardiness into a flight's ground-servicing schedule. Ideally, once a scheduling decision has been made, it should remain unchanged

Table 1.1. Perspectives on the Variable-Ordering Process

Tightness of Constraints	The complete set of constraints currently imposed on a subproblem as the result of previous scheduling decisions provides a strong indication of how difficult its solution is expected to be. Information provided by these constraints indicates the current demand for and availability of the resource classes involved, the size of the pool of candidate resources, and a particular locality within which a successful reservation may be found. Constraints on subproblems limit the options that are available for their satisfaction. As a result, subproblems should be addressed before becoming too heavily constrained so that their means of satisfaction are as abundant as possible.
Temporal Urgency	The proximity of a subproblem's allowance to the actual current world time strongly affects the urgency with which that subproblem should be addressed by the scheduler. Subproblems should be solved in time for their required operations to be performed within their desired allowances.
Type of Subproblem	The specific nature of a subproblem, within the context of the current state of problem solving, plays an important role in determining its urgency. Some problems are more serious than others. For example, it may be more important to satisfy outstanding requests for bottleneck resources than to go back and refine previous scheduling decisions made under uncertain conditions (an activity described in Section 4.2.6). Note that these priority relationships among different kinds of subproblems tend to vary over time with the state of problem solving. In situations where refinement involves reservations for bottlenecked resources, the process of refining those task-nodes and freeing up valuable time within those resource schedules to satisfy remaining demand may become the most urgent scheduling task on the queue.

by future decisions. Changing gate assignments may be problematic if the existing mobile resource reservations for the baggage trucks that must report to a gate include only enough time to get to that particular gate location. Because the AGSS domain contains a large number of mobile ground-servicing vehicles (such as baggage trucks), it is also desirable to minimize the amount of time that these vehicles spend moving throughout the environment, to provide more time for performing servicing tasks. Finally, the nature of a flight also contributes to the reservation-selection process. The priority of a flight may dictate that it receive the best possible reservations available, so that it departs on time and has all of its required ground-servicing operations completed according to its specific operation preferences. These value-ordering perspectives are summarized in Table 1.2.

1.5 Research Overview

Our approach to solving dynamic RCSPs is based on providing four important capabilities, described as follows:

- Address the dynamic nature of the problem using a fine-grained, opportunistic decision-making process.
- Use slack time and least-commitment decision making to preserve flexibility throughout the scheduling process.
- Use multiple, relevant perspectives to inform all control and scheduling decisions.
- Provide generic applicability for a variety of RCSP domains.

Additionally, we have sought to deal with the additional complexity that arises from handling RCSPs that involve mobile resources with significant travel requirements, and common activities that are shared among separate orders.

To test our approach, we have designed and implemented a generic reactive knowledge-based scheduling system called DSS (the Dynamic Scheduling System) which implements our approach. DSS provides a foundation for representing a wide variety of real-world RCSPs. Its flexible scheduling approach is capable of reactively producing quality schedules within dynamic environments exhibiting unpredictable resource and order behavior.

DSS attempts to satisfy the following scheduling objectives:

- Minimization of the average tardiness cost per order
- Minimization of the number of tardy orders
- Maximization of scheduling efficiency
- Minimization of the total duration of all resource setup activities (including mobile resource travel time)

While DSS does not specifically attempt to minimize order flow times, our experiments have indicated that the resulting flow times are not substantially worse than some other scheduling systems that do attempt to satisfy this objective. The objective of minimizing the amount of

Table 1.2. Perspectives on the Value-Ordering Process

Resource Contention	The current level of contention for a particular class of resource provides important information to the scheduler about the impact of its value-ordering decisions on future attempts to secure resources of the same class. In situations where the contention for a resource class is sufficiently high, it is desirable to compact the resource schedules by minimizing the introduction of external fragmentation, to leave as much contiguous free time available for satisfying the many outstanding reservation requests. When contention values are lower, the compaction of the resource schedules is less important, owing to the decreased value of the free time for the resource class. Decisions made without consideration of the current level of contention are liable to contribute to the creation of future conflicts that will be increasingly difficult to solve, or to produce schedules of lesser quality.
Schedule Disruption	The potential for the disruption of other areas within the developing schedule weighs heavily on the value-ordering process. Preemption of existing reservations requires consideration of the impact of reactivating previously satisfied subproblems. The introduction of delays requires consideration of the impact of having to shift other existing reservations and increase the tardiness of one or more jobs. The impact of these decisions extends to both the variable-ordering *and* value-ordering processes. Disruption may alter the queue of pending scheduling activities, and significantly affect the quality of the developing schedule.
Preferences and Objectives	It is important to consider the particular preferences of resource classes, orders and operations, as part of the value-ordering process. It may be desirable, perhaps globally, to minimize the amount of time that resources spend traveling within the factory environment, or the number and processing duration of the setup operations required to be performed by a resource. The satisfaction of the shift preferences of individual operations and the completion times of orders are similar concerns that are affected by a scheduler's reservation-securing process.

computational effort that goes into the scheduling process is achieved by exploiting different areas of flexibility within both the developing schedules and the scheduling process itself.

Figure 1.6 provides an overview of the basic components of DSS, as well as the various domain-specific information that is required for describing a particular RCSP. DSS is implemented as an agenda-based blackboard system [Erman *et al.*, 1980]. It maintains a blackboard structure upon which the developing schedule is constructed and the sets of orders and resources are stored. A set of generic domain-independent knowledge sources is provided for making the assorted variable-ordering and value-ordering scheduling decisions. These knowledge sources are triggered as the result of developments on the blackboard, and by an event processor that receives (simulated) world events and activates the appropriate knowledge sources for handling their response. Triggered knowledge sources are placed onto an agenda and executed in order of priority.

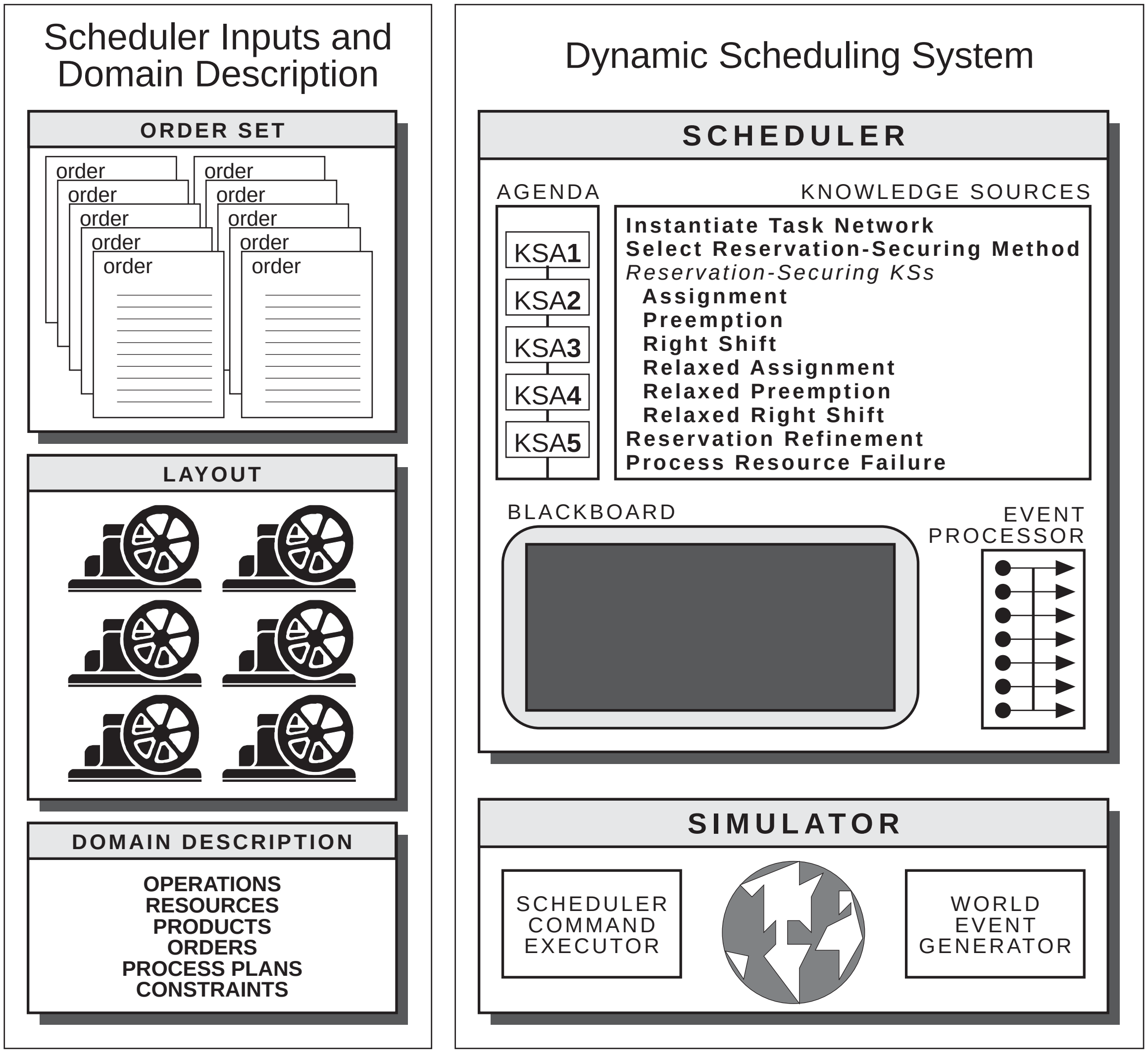


Figure 1.6. Scheduler Inputs, Domain Definitions, and Basic System Components of DSS.

DSS is also equipped with a simulator that implements a real-world, dynamic scheduling environment. The simulator executes the commands issued by the scheduler and generates occasional unexpected events. It also serves as a means of checking the integrity of the schedules produced by DSS, by ensuring that they can be successfully executed.

The domain-specific information required by DSS includes a knowledge base that describes the operations, resources, products, orders, process plans and constraints involved in a particular RCSP. An order set and layout define an instance of an actual RCSP for which DSS may produce a schedule.

1.5.1 Important Architectural Features and Functionality

DSS is distinguishable from other scheduling systems in a number of ways that provide important functionality throughout the entire scheduling process. We describe these architectural features below.

- **Explicit Representation of Slack Time**

  We have designed and implemented a scheduling approach that maintains flexibility within the developing schedule in the form of explicitly represented slack time. Such flexibility acts to preserve the options available to the scheduler for resolving future scheduling conflicts. Our least-commitment approach contributes to the production of quality schedules in dynamic environments where unexpected conflicts develop frequently. This capability has not been addressed by other scheduling systems.

  The slack time associated with a task-level subproblem represents a localized decision-making area for scheduling decisions to be made without seriously impacting other subproblems. These areas help to indicate the range of scheduling options that are available to an individual subproblem at any time during the scheduling process. This information is extremely valuable in helping to determine subproblem urgency.

  As the amount of slack incorporated within the developing schedule is affected by scheduling decisions, the sizes of all related slack nodes are modified, and the urgency of each unsolved subproblem with which they overlap (in time and resource class) is updated in response. Changes (often subtle) in the supply of slack within the developing schedule are thus immediately detected by the scheduler.

  The slack time that is available to the individual scheduling subproblems is explicitly represented within DSS by special *slack node* units built into the developing order schedule. These units are indexed onto a particular blackboard space according to the subproblem's allowance and the resource class(es) desired by the operation.

  Explicit representation of slack time allows DSS to maintain a well-informed understanding of the relative urgency of its subproblems, thereby providing it with the means to react quickly and properly to unforeseen developments in dynamic environments, and improve its path of exploration through the search space using least-commitment techniques. The existence of such explicit slack time also broadens the range of perspectives that may be consulted throughout the decision-making process.

- **Fine-Grained, Multiple-Perspective, Reactive Problem Solving**

The breakdown of the RCSP into finely grained, individual subproblems provides the basis for implementing a reactive, operation-based scheduling approach that provides maximum flexibility for responding to dynamic environments The power of such a representation has been previously demonstrated, in terms of minimizing the number of search states generated in the course of problem solving [Sadeh, 1991]. But the power of this approach in the context of reactive scheduling has not been explored.

Coupled with a multiple-perspective analysis of the problem-state that considers fully the relevant scheduling perspectives on the current state of problem solving at key decision-making points, the ability to adapt both quickly and effectively to a dynamic environment has been achieved. The urgency of each individual subproblem, inherently dependent on the current state of problem solving, is used to organize the queue of scheduling tasks that must be executed by DSS at any time. DSS is therefore able to shift its focus of attention quickly from subproblem to subproblem as the changing conditions within the environment warrant.

- **Maintaining an Informed View of Resource Contention Levels**

  Each subproblem requiring a resource for an operation is matched with a *service goal* that becomes responsible for managing its solution. By linking these service goals together according to overlaps in their allowances and sets of potentially satisfying resources, DSS is able to construct a high-level view of the current level of resource contention that exists within the environment at any particular time [Corkill *et al.*, 1982].

  As scheduling decisions are made, the allowances and sets of potentially satisfying resources for these service goals change, which impacts the degree to which they are linked to each other. Changes in these link sets are then used to modify the urgency of their corresponding subproblems.

  The high-level view of problem solving created by this interconnected service goal network provides DSS with a valuable understanding of the topology of the developing search space, which is then applied to the process of determining subproblem urgency. The ability of DSS to react to a rapidly changing environment is therefore significantly enhanced.

- **Refining Previous Uncertain Decisions**

  The ability of DSS to refine previous scheduling decisions in response to new information is achieved by means of a special reservation refinement process controlled by *refinement goals* that are triggered for previously solved but currently unrefined subproblems whose contexts have been further defined as the result of subsequent scheduling decisions. The ratings of these refinement goals are determined by the current level of contention and anticipated demand for the class of resource involved, across the entire processing duration for the existing reservation.

  The network of refinement goals provides further information about the kinds of tasks remaining to be processed by the scheduler, and indicates areas within the developing schedule that have the potential for producing resource availability to help satisfy other outstanding resource requests.

  The reservation-refinement process provides the clean-up mechanism required by a decision-making strategy that seeks to build flexibility into its schedules. When DSS makes a scheduling decision under conditions of uncertainty, it relies on the fact that at some later date, after more complete information regarding such a decision has been obtained, it may go back and refine that decision to account for its newly enhanced view of the world. Without such a refinement mechanism, DSS would be forced to accept lower-quality, worst-case schedules as the price of its attempt to explicitly maintain scheduling flexibility.

- **Anticipating the Impact of the Decision-Making Process**

  The projection technique utilized by the various knowledge sources to determine the complete impact of their potential scheduling decisions on the current state of problem solving allows DSS to organize its decision-making process to help prevent future conflicts from developing.

  Special *projector units* that mimic the defined resources are provided for this technique. At particular points during the scheduling process, these projector units may be manipulated by the DSS knowledge sources to determine the various effects of possible scheduling decisions. The effects of these projected scheduling decisions are simulated by the KSs in the course of their invocation.

  For example, in the course of attempting to preempt an existing reservation to satisfy some other subproblem, the actual preemption of each relevant existing reservation is simulated using the corresponding projector units. The KS considering the preemption is therefore able to understand completely the impact of each potential preemption case, and make its eventual scheduling decision based on that information.

- **Scheduling Mobile Resources with Significant Travel Requirements**

  In an attempt to handle a wider range of real-world RCSPs, we have extended the RCSP model to include mobile resources with significant travel requirements. Quite often, the amount of travel required of a particular resource to service a product represents a significant portion of the overall processing duration for the assigned operation. Required travel time based on location thus provides an important distinguishing factor among otherwise similar resources, and has a considerable impact on the decision-making process.

  This extension to our RCSP model warrants additional attention from our scheduling approach. It must be possible to make scheduling decisions under conditions of uncertainty, when relevant locations are undetermined. Additional mechanisms are required for refining previous decisions as more contextual information becomes available.

  Existing scheduling systems have avoided this situation by assuming that mobile resource travel durations are insignificant, thereby precluding their use in many real-world scheduling domains (such as transportation planning).

- **Accommodation of Inter-Order Tasks**

  We have further extended the class of RCSPs that we are able to handle to include the sharing of common inter-order tasks among otherwise independent jobs. This capability allows us to represent more realistic scheduling problems, and forces the scheduler to consider a broader range of perspectives at certain decision-making points.

> The interaction between subproblems within the same order, and among separate orders, is achieved by linking together individual subproblems according to their preceding and succeeding tasks, possibly across orders. Sets of orders defined by inter-order tasks are therefore represented as composite connected graphs containing the subproblems for a number of orders.

1.5.2 Evaluating The Applicability of Our Approach

DSS has been used successfully as the base system upon which a number of applications have been built to solve interesting RCSPs. These applications include: an airport ground service scheduling domain, a simplified turbine component production plant, and a large-scale transportation planning environment. Each of these domains and applications are described in greater detail in Appendix A.

1.5.2.1 The Airport Ground Service Scheduling Domain

A significant portion of our work has focused on the airport ground service scheduling domain. The AGSS domain differs from the standard job-shop environment in that the minimization of order flow times, as represented by the interval between the gate arrival and departure times for an aircraft, is not an objective for the scheduler. The goal instead is to develop a schedule of ground-servicing activity for each aircraft that allows it to meet exactly its targeted arrival and departure times.

The AGSS domain has provided us with the opportunity to better understand the important role that slack time plays throughout the scheduling process. Furthermore, it has given us experience in scheduling a greater variety of activities, many of which can be performed simultaneously. Mobile resources may have significant travel requirements, forcing them to report to various locations in the course of performing their assigned duties. Quite often, these locations may not be known at the time that the mobile resources are being secured. In addition, some AGSS tasks have a preference for being scheduled later in the process plan, instead of being shifted as early as possible. Finally, the AGSS domain domain involves inter-order tasks (in the form of baggage and passenger transfer operations) that connect otherwise independent jobs, requiring a scheduler to be able to represent and propagate constraint information across separate orders.

The airport ground service scheduling RCSP presents an interesting scheduling problem, owing to the tendency for airport timetables to pack a great deal of activity into a number of short time periods throughout an entire day. These surges (also called *banks* or *pushes*) constitute groups of flights that either arrive or depart together, in such a way as to maximize the connections met by passengers that pass through the airport on their way to other destinations. An arrival push is generally followed by a corresponding departure push within 30–45 minutes. At large hub airports, where one or more airline companies routes a significant portion of its flights, the degree of activity occurring within a single push becomes even higher. In addition, the AGSS domain involves a highly dynamic environment. Weather conditions, both local and remote, have a severe impact on airport activities. Many ground-servicing tasks take longer to complete under difficult weather conditions, and additional servicing is often required in such situations. Delays in anticipated timetables resulting from problems at other airports tend to complicate the local scheduling problem by rendering previous scheduling decisions obsolete.

Our experiences with this particular domain are discussed further in Section 5.6.

1.5.2.2 A Simplified Job-Shop Scheduling Domain

The job-shop scheduling domain that we have implemented is a test domain described in [Chiang *et al.*, 1990] which was used for evaluating the OPIS system [Smith *et al.*, 1986a, Ow, 1986, Ow and Smith, 1988]. The (simplified) shop produces three kinds of turbine propeller blades (two of each kind). The production of a propeller blade consists of either three or six sequential machining operations, performed on a variety of stationary resources. Notable in this shop model is the fact that the factory operates for only five days at a time, often requiring jobs to remain idle during the time that the factory is shut down. Setup operations are required at any time a resource must shift from working on one kind (family) of propeller blade and shift to another. The processing durations of these setup operations are constant for each type of resource.

The most important feature about this domain is the fact that one of the scheduling objectives is to deliver each job as quickly as possible, that is, to minimize its flow time through the shop. Such a requirement presents a challenge for DSS in that it is not designed to specifically contract its schedules in an attempt to deliver jobs as early as possible.

In addition, this turbine component plant, simplified though it is, presents a realistic RCSP for use in evaluating the applicability and performance of DSSin common job-shop domains. While the process models for the blade families are rather flat, the processing durations and the shop model are based on features of a real-world domain.

Our experiences with this particular domain are discussed further in Section 5.6.

1.5.2.3 A Large-Scale Transportation Scheduling Domain

Our third domain for evaluating applicability is a large-scale transportation scheduling problem. In this domain, the main activity is the transfer of cargo between ports by boat. Bundles of cargo are delivered at different times to various ports around the globe. Each bundle of cargo must first be loaded onto a ship, transferred to a destination port (possibly via a sequence of intermediary ports), and finally unloaded. The domain model uses dock resources (equipped with cranes) to perform the loading and unloading of the cargo.

A collection of ship types is defined, each with varying travel speeds and cargo capacities. Each port provides some number of docks, each constrained in the size and type of ship it may service. An important simplification in our implementation of this domain is the requirement that the capacity of each ship is limited to a single job, that is, a single ship cannot be used to transport the cargo of more than one job between ports.[8]

This particular transportation environment represents a practical, real-world RCSP that involves mobile resources with significant travel requirements. Because of the distances involved, the duration of any transfer operation represents a major portion of the entire project duration for each job. The speed of each individual ship therefore can play an important role in the reservation-securing process. In addition, there are a number of important time bound constraints on the various activities, in that penalties may be incurred as the result of jobs being delivered ahead of (in addition to behind) schedule.

Our experimentation with this particular RCSP was basically limited to the evaluation of the applicability of the DSS representation schemes and the problem-solving approach. All

[8] This constraint is imposed by the current implementation of DSS.

indications were that DSS was entirely capable of representing the features of this domain and producing quality schedules for realistic order sets.

1.6 Contributions

This work represents a significant advancement in the development of reactive, knowledge-based systems for solving dynamic RCSPs. We have implemented a working system, DSS, which is able to generate feasible, quality schedules for a variety of (often complex) domains under a broad range of unexpected environmental conditions.

The specific contributions of this work relate to the development of a generic, multi-faceted and flexible approach to solving dynamic RCSPs, and the extensive evaluation of the performance of this approach in a number of different scheduling domains. The specific contributions are highlighted below.

- **Development of a Multi-Faceted Approach to Solving Dynamic Resource-Constrained Scheduling Problems**

  The reactive scheduling approach implemented within DSS manages to preserve and exploit flexibility throughout the scheduling process while achieving the following simultaneous objectives:

  – *Use of Slack Time to Preserve Flexibility and Limit Schedule Disruption*

    By maintaining slack time within developing schedules, our least-commitment decision-making process helps preserve options for solving outstanding subproblems. This explicit slack time also provides the means for adapting to unexpected developments in a way that avoids severely disrupting the existing schedule. The degree of pliancy exhibited by a scheduler in reacting to unforeseen events is a measure of flexibility that is increasingly important when operating within dynamic environments. Unlike other systems that intentionally compact their developing schedules throughout the scheduling process [Fox, 1983, Smith *et al.*, 1990, Sadeh, 1991], leaving very little room for dealing easily with future conflicts, DSS provides for such contingencies while still managing to produce quality schedules.

  – *Quick and Effective Reaction to Dynamic Environments*

    Effective problem solving in dynamic environments requires the ability to choose between working on predictive tasks and reacting to unexpected events. Our focus on the kinds of problems that arise in these environments has guided the development of a fine-grained, opportunistic scheduling approach that interleaves the execution of predictive scheduling tasks with the resolution of conflicts and other unexpected developments. By avoiding the common tendency to rigidly alternate between predictive and reactive scheduling modes [Smith, 1987, Ow *et al.*, 1988, Sadeh *et al.*, 1993], we provide DSS with the ability to put each scheduling task in its proper perspective within the overall scheduling process, and explore the developing search space as conditions within the environment warrant.

  – *Consultation of Multiple Perspectives in All Scheduling and Control Decisions*

    The ability to understand the impact of changes brought about by internal and external events is considerably important in the process of reacting to a changing environment. To inform the decision-making process under these circumstances, we have equipped the control (variable-ordering) and scheduling (value-ordering) heuristics in DSS with the ability to consider, at their individual decision-making points, a comprehensive range of appropriate perspectives on both the current state of problem solving and any relevant, anticipated or pending scheduling activities. This capability is absolutely necessary within dynamic scheduling environments to guide the scheduler and minimize the need for backtracking and constraint relaxation. Existing systems tend to rely on a narrow consultation of perspectives on the current problem [Fox, 1983, Smith *et al.*, 1990, Sadeh, 1991], thus limiting their ability to fully comprehend the current state of problem solving and react accordingly.

- ***Accommodation of Additional Domain Complexities***

  Many real-world scheduling problems introduce additional complexities to the class of canonical RCSPs. Our experience with the AGSS domain has led to the development of mechanisms to deal with the following two such features:

  * *Inter-Order Tasks*
    Tasks that extend across otherwise separate orders effectively broaden the extent of their corresponding subproblems. The decision-making process involving such inter-order tasks at any time must therefore consider a wider range of information to avoid settling for lesser-informed decisions. DSS is capable of representing inter-order tasks and understanding their implications at all key decision-making points throughout the scheduling process.
  * *Mobile Resources with Significant Travel Requirements*
    Standard RCSPs tend to downplay the issue of significant travel requirements for mobile resources. Movements required of mobile resources are generally treated as insignificant, in comparison with the processing durations of their main servicing tasks. Many RCSP domains, however, require a scheduler to honor the (often significant) travel requirements of mobile resources in the process of developing a schedule. Large-scale transportation problems, for example, require a great deal of mobile resource movement. Schedulers unable to accommodate such requirements are of little use in such domains. DSS is capable of not only managing such situations, but also devotes its efforts to the minimization of the total amount of mobile resource travel costs incurred by a particular schedule.

- **Implementation of Our Approach in a Knowledge-Based Scheduling System**

  We have implemented our scheduling approach in a knowledge-based scheduling system called DSS (the Dynamic Scheduling System). DSS is capable of representing and solving a broad range of real-world RCSPs.

- **Extensive Evaluation of the DSS Scheduling Approach**

  We have undertaken an extensive evaluation of the DSS scheduling approach, organized into two important sets of experiments.

  - *Comparisons with Common Benchmarks*

Using a set of benchmark data designed for testing the OPIS system [Ow, 1986, Ow and Smith, 1988, Chiang *et al.*, 1990], we have been able to compare (favorably) the scheduling performance of DSS with results from a version of OPIS (OPIS 0) [Smith *et al.*, 1986a], ISIS [Fox, 1983], and a modified version of the COVERT dispatch rule [Vepsaleinen, 1984] in a simplified job-shop domain. These experiments confirm the improvements in reduced tardiness that are achieved by the scheduling approach implemented in DSS.

– *Execution under a Range of Reactive Conditions*

Using our own AGSS domain implementation, and the simplified job-shop domain described above, we have also evaluated the performance of DSS in a range of dynamic scheduling situations that include unexpected resource failures and surprise order receptions. These experiments further confirm the success of the DSS scheduling approach in reacting to uncertain environmental conditions.

- **Demonstration of the Generic Capabilities of DSS**

  The final contribution of this work is the development of a generic decision-making framework for solving dynamic RCSPs that manages to preserve and exploit simultaneously the various sources of flexibility inherent in the problem while providing a foundation for representing a broad range of real-world RCSP domains. The generality of our approach has been confirmed by our experiences in successfully implementing, on top of the DSS framework, individual scheduling systems for solving dynamic RCSPs in three substantially different domains.

1.7 Navigational Aids

Chapter 2 provides the background for our work, beginning with a formal description of the class of RCSPs that are solved by DSS, and proceeding with a discussion of classical OR approaches to solving RCSPs. The latter half of Chapter 2 describes the field of knowledge-based scheduling, presenting the state-of-the-art in the field, and presenting three important systems with which DSS can be compared. A number of additional AI approaches to solving this class of problems are also discussed.

Chapter 3 is the first of two chapters devoted to the full description of DSS. We begin by presenting our general assumptions about the class of dynamic RCSPs, followed by a description of the basic entities for their representation in DSS. Finally, we discuss the various types of constraints that are handled by DSS. Issues related to our extension of the class of RCSPs to include inter-order tasks and mobile resources with significant travel requirements are discussed here as well. Chapter 3 illustrates the generic capabilities exhibited by DSS in terms of the broad range of problems that it is capable of handling.

In Chapter 4, we provide a complete description of the implementation of our reactive, opportunistic, operation-based scheduling approach in DSS. We begin with the blackboard hierarchy upon which the approach is built and then explain in detail the complete scheduling process, including the mechanisms for preserving and exploiting available slack time, our multiple-perspective consultation techniques, and our least-commitment decision-making process.

The evaluation of DSS and its scheduling approach appears in Chapter 5. One set of experiments involving a common benchmark demonstrates the improved results achieved by DSS in comparison to some existing scheduling systems. The second set of experiments involves the operation of DSS within a number of dynamic environments to test its reactive capabilities. This chapter demonstrates the success of our scheduling approach in reacting to a range of unexpected environmental events and producing quality schedules under such conditions. It also demonstrates the generic capabilities of DSS in representing a number of significantly different RCSPs.

Finally, in Chapter 6, we review the contributions of our work with DSS, and discuss a number of possible directions for future work.

We also provide two appendices that offer a more detailed look at DSS. Appendix A presents the complete definitions of two application systems built on top of DSS to implement the various RCSPs referred to in this document, and Appendix B presents an annotated DSS execution trace.

CHAPTER 2

RELATED WORK

We begin our discussion of the important related work in the field of resource-constrained scheduling with a focus on classical operations research (OR) approaches. We continue with a description of some relevant rule-based expert system approaches. In both of these sections we include specific examples in which these solution techniques are applied to various kinds of specific resource-assignment problems that occur within the airport ground service scheduling (AGSS) domain. Finally, we survey some other relevant artificial intelligence (AI) approaches, focusing on three important knowledge-based scheduling systems that have helped influence the design and development of DSS.

General surveys of the issues involved in the solution of RCSPs can be found in [Davis, 1973, Graves, 1981, Willis, 1985]. Discussions of a number of different AI approaches to solving the problem can be found in [Noronha and Sarma, 1991, Smith, 1991].

2.1 Operations Research Approaches

As mentioned in Chapter 1, the class of RCSPs has been extensively studied within the field of operations research. It is, therefore, a relatively well-understood class of problems. In this section we will discuss the progression of standard OR approaches for solving RCSPs. We follow the presentation of our ILP formulation with a description of the standard optimal and non-optimal methods that have been developed for the solution of RCSPs. For each of these methods, we discuss the range of established solution approaches.

The progression of OR approaches to solving the RCSP moves along two dimensions, both having to do with the nature of the desired solution. Early OR work focused on the production of optimal solutions, according to singular objective functions. But the difficulties involved with finding optimal solutions for large-scale problems led to a shift of effort towards developing methods for producing near-optimal solutions, which would approximate optimality while incurring significantly less computational expense. Again, however, when such approaches were applied to real-world problems with often complex solution criteria, another change in approach was warranted. At this point, the shift was toward the development of approaches that could produce solutions that satisfied objective functions comprising multiple scheduling objectives, which brings us to the state of the art in terms of methods for solving RCSPs in the OR field. Throughout the rest of this section, we will discuss each of the phases mentioned above, concluding with a discussion of some other recent OR solution approaches and a summary of the differences between the state of the art in OR and the approach taken in the design of DSS.

2.1.1 An Integer Linear Programming Formulation

We begin by presenting an integer linear programming (ILP) formulation to describe explicitly the various kinds of RCSPs that are handled by DSS. Our formulation is based on a similar model presented in [Pritsker *et al.*, 1969]. It also refers back to our discussion of dynamic RCSPs in Section 1.3. Another formulation for a tighter-constrained version of the RCSP can be found in [Papadimitriou and Steiglitz, 1982].

Integer linear programming is a classical technique for finding optimal solutions to sets of multiple-variable, integer-constrained, linear equations, of which the RCSP is a typical example. An ILP formulation comprises a collection of variables, constraints on their possible values, and an objective function to be either minimized or maximized in the process of assigning values to the variables. An objective function may encode either a *singular* scheduling objective, such as the minimization of order tardiness or required setup activity, or it may attempt to satisfy a collection of *multiple* objectives.

In the standard scheduling problem, where a set of jobs must be serviced by a collection of resources, variables can be defined to indicate which particular resource is assigned to which particular job, and to record the specific times during which an operation is to be performed by a resource. Additional constraints, such as those for controlling temporal sequencing, can be formulated using equations that specify that the operation starting times for succeeding tasks must be greater than the operation finishing times of their predecessors. Additional variables store the expected durations of the tasks. The objective function for an RCSP may be an equation whose value is intended to be either minimized (such as schedule makespan, tardiness cost, or total resource setup time) or maximized (such as the degree of resource utilization). Formulations of practical problems can easily require a large number of variables and constraints.[1]

We begin by restating the goal of the basic RCSP, which is to determine for a set of jobs (orders, projects), machine assignments for all tasks (within each job) that require the services of a resource. Each machine assignment consists of a particular machine and the starting and finishing times for the execution of the task. Our objective in this problem is to minimize both the average tardiness cost (lateness penalty) per order, and the percentage of tardy jobs over the set of all jobs.

2.1.1.1 Assumptions

The basic relevant assumptions for our problem formulation are reiterated from Figures 1.3 and 1.4.

- The set of available resources is limited.
- Tasks may have precedence relations within and among jobs.
- Each job has a ready time and due date.
- There may be a choice among resource types to perform a task.
- Each task requires at most one ‘unit’ of any resource type.

[1] Our formulation makes no attempt to minimize the total number of variables or constraint equations required.

- Tasks are atomic entities. They may not be split.
- Tasks cannot be preempted during execution.

Note that because our model assumes the availability of deterministic task processing time durations, we calculate the duration of each task by including the maximum possible amount of time required to perform all necessary setup, reset, or travel activities.

A preliminary analysis of the process plans for a set of jobs can be used to determine relevant time bound information for a particular problem, such as the earliest and latest starting and finishing times for tasks, and the earliest and latest completion times for jobs.

2.1.1.2 Definitions

Figure 2.1 provides some important definitions for our formulation. The earliest possible finishing time (EFT_{ij}) and latest possible finishing time (LFT_{ij}) define a range during which each task T_{ij} must finish processing. The exact time period during which a task finishes is

i	=	job number, $i = 1, 2, \ldots, I$; I = number of products (one job per product).
j	=	task number, $j = 1, 2, \ldots, N_i$; N_i = number of tasks in job i.
t	=	time period, $t = 1, 2, \ldots, LCT$; LCT = absolute due date.
k	=	resource class, $k = 1, 2, \ldots, R$; R = number of different resource classes.
T_{ij}	=	task j of job i.
d_{ij}	=	processing duration (in time periods) for task T_{ij}. If task T_{ij} *starts* executing in period s_{ij}, it *finishes* executing in period $s_{ij} + d_{ij} - 1$.
A_i	=	ready time for job i.
D_i	=	desired due date for job i. All tasks in job i must be *finished* in a time period $t < D_i$. Job i is *late* if it is *completed* in a time period $t \geq D_i$.
P_i	=	lateness penalty for completing job i in period $t > D_i$.
ECT_i	=	earliest possible completion time for job i.
LCT_i	=	latest possible completion time for job i. $LCT > \max(LCT_i)$.
EST_{ij}	=	earliest possible starting time for task T_{ij}.
EFT_{ij}	=	earliest possible finishing time for task T_{ij}.
LFT_{ij}	=	latest possible finishing time for task T_{ij}.
r_{ijk}	=	the amount of resource type k required by task T_{ij}.
R_{kt}	=	amount of resource type k available in period t.
x_{ijt}	=	a binary variable indicating whether or not task T_{ij} *finishes* in period t (1 for yes, 0 for no). $x_{ijt} = 0$ for $t < EST_{ij}$ and $t > LFT_{ij}$.
x_{it}	=	a binary variable indicating whether or not job i is *complete* in period t (1 for yes, 0 for no). $x_{ij} = 0$ for $t < ECT_i$ and $t > LCT$.

Figure 2.1. Definitions for an RCSP Integer Linear Programming Formulation

represented by a value of 1 for the corresponding x_{ijt} variable (where t indicates the finishing time period). Finishing times are dependent on resource availability, and additional sequencing and processing duration constraints. The earliest possible completion time (ECT_i) and latest possible completion time (LCT_i) define a range during which project i must be completed. For each time period within this range that a project has been completed, the corresponding x_{it} variable receives the value of 1. A complete schedule is effectively represented by the collection of non-zero valued x_{ijt} variables.

2.1.1.3 The Model

In Figure 2.2 we present a visual representation of a schedule constructed by our model for a simple RCSP consisting of two projects, each consisting of three operations. The process plan for project 1 contains three tasks: T_{11}, T_{12}, and T_{13}, and a sequencing constraint that requires task T_{11} to precede task T_{13}. The process plan for project 2 contains tasks T_{21}, T_{22}, and T_{23}, and defines task T_{21} as an aggregate task that encompasses the remaining two tasks, T_{22} and T_{23}. In addition, task T_{22} may be performed by one of two different kinds of resources (type 1 or 2). (In Figure 2.2, we define the variables x^1_{22t} and x^2_{22t} to represent the finishing status for task T_{22} using either of the two possible resource classes.)

Figure 2.2 shows, for each task, the finishing time range $[EFT_{ij}, LFT_{ij}]$, and the required processing duration and actual scheduled assignment (indicated by the heavy black line). The shaded completion time range $[ECT_i, LCT_i]$ for each project is also shown. The ready time period, that is, the time period *in* which the processing of a project's tasks may begin, is 1 for the first project, and 6 for the second project. The due date time period, that is, the time period *by* which the processing on all of a project's tasks must be completed, is 16 for both projects (requiring all task processing to *finish* no later than time period 15). Figure 2.2 shows the values of the binary x_{ijt} and x_{it} variables for both scheduled projects (for example, the value of x_{114} is 1, indicating that task T_{11} finishes executing in time period 4). All values not explicitly stated for the x_{ijt} and x_{it} variables default to 0. In the completed schedule represented in Figure 2.2 project 1 is completed by its due date, while project 2 is completed 5 time periods late. Note also that task T_{22} is performed using a resource of class 1. Additional details of this figure are discussed throughout the rest of this section.

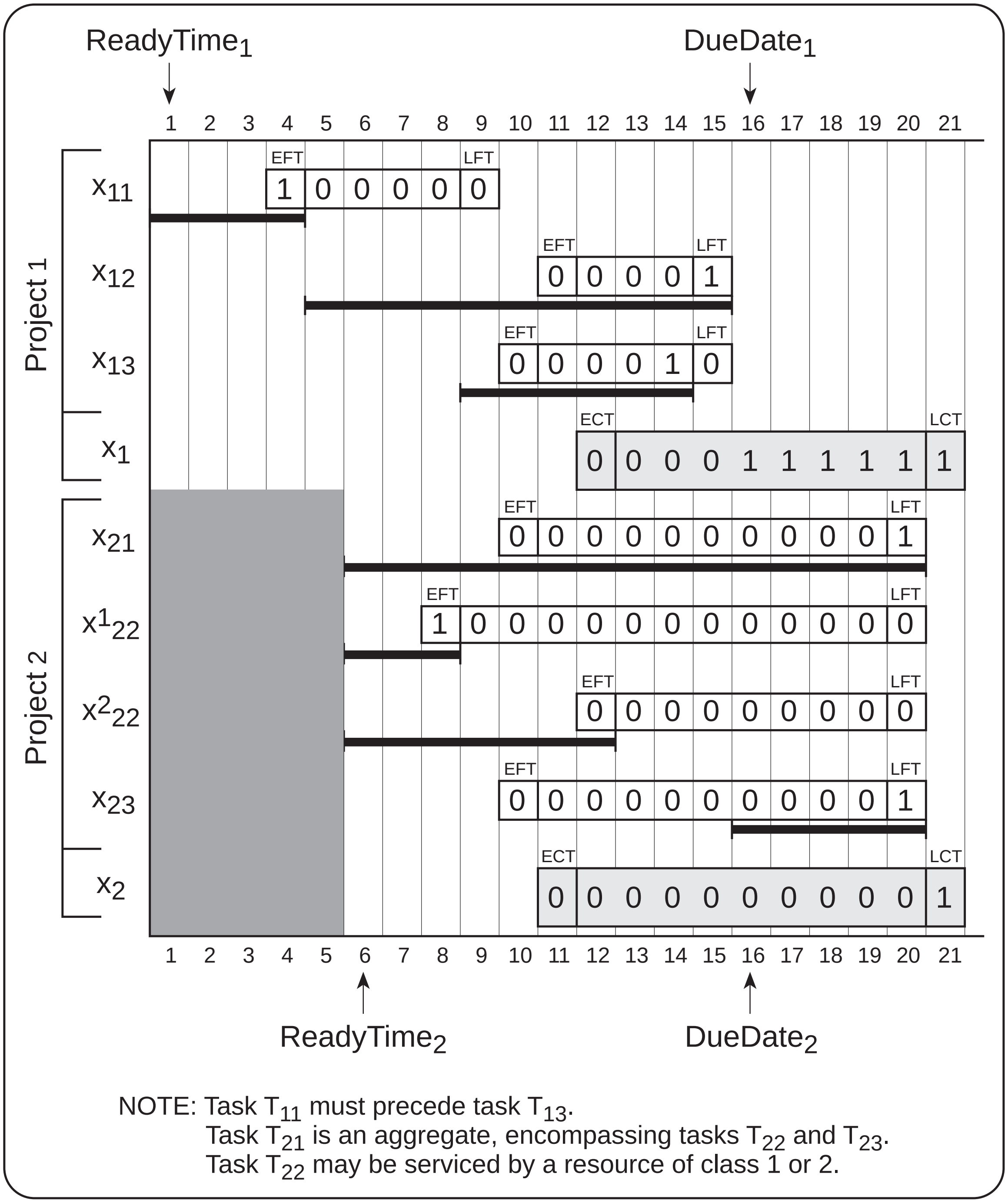
2.1.1.4 The Objective Function

Our primary objective in solving DRCSPs is to minimize the average tardiness cost per order. This is equivalent to minimizing the total tardiness cost.

Minimizing the value of the following equation minimizes the un-*weighted* tardiness sum for a set of jobs. Each non-zero value for x_{it} within the time range of $[D_i, LCT)$ indicates a time period at which point job i has been completed.

$$\sum_{i=1}^{I} \sum_{t=D_i}^{LCT-1} x_{it} \tag{2.1}$$

Referring back to Figure 2.2, note that the larger the result of the above equation, the earlier job i is completed, and hence, the lower its tardiness. For example, the x_{1t} variables for project 1 produce an un-weighted tardiness sum of 5 (across all values of t). The x_{2t} variables likewise



NOTE: Task T_{11} must precede task T_{13}.
Task T_{21} is an aggregate, encompassing tasks T_{22} and T_{23}.
Task T_{22} may be serviced by a resource of class 1 or 2.

This figure is based on Figure 1 (page 95) from "Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach," ***Management Science****, v.15, no.1, pp.93-108, September 1969, by Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe.*

Figure 2.2. Visual Representation of an RCSP Solution Produced from an Integer Linear Programming Formulation.

produce a sum of 0. If Equation 2.1 evaluates to 0, the job is late (as is project 2 in this case). Note that we can therefore minimize total tardiness by *maximizing* the value of the equation.

By factoring in the lateness penalty P_i for each time period during which job i is complete, our final objective function (to be maximized) becomes:

$$\sum_{i=1}^{I} \sum_{t=D_i}^{LCT-1} P_i \, x_{it} \tag{2.2}$$

2.1.1.5 Constraint Equations

In this section we present the constraint equations that define the bulk of our RCSP model.

2.1.1.5.1 Task Finishing Constraints

Each task finishes exactly once. This constraint is achieved by the following equation which ensures that for each task T_{ij}, only one x_{ijt} variable may be assigned the value 1 during the time range of $[EFT_{ij}, LFT_{ij}]$.

$$\sum_{t=EFT_{ij}}^{LFT_{ij}} x_{ijt} = 1 \quad i = 1, 2, \ldots, I; \; j = 1, 2, \ldots, N_i \tag{2.3}$$

2.1.1.5.2 Job Completion Constraints

A job may not be completed until all of its tasks have finished executing. The following equation enforces this constraint by requiring the binary x_{it} variable for job i at any time t to be no greater than the ratio of the number of finished tasks in job i in the immediately preceding time period to the total number of tasks in job i (N_i).

$$x_{it} \leq \left[\frac{1}{N_i}\right] \sum_{j=1}^{N_i} \sum_{q=EFT_{ij}}^{t-1} x_{ijq} \quad i = 1, 2, \ldots, I; t = ECT_i, ECT_{ij} + 1, \ldots, LCT \tag{2.4}$$

2.1.1.5.3 Temporal Sequencing Constraints

Our model allows temporal sequencing relations to exist between any two tasks. Tasks related by sequencing constraints may or may not reside within the same job. Temporal sequencing constraints require that all preceding tasks finish executing prior to the start of execution for any succeeding tasks. The finishing time f_{ij} for each task T_{ij} may be defined by the following equation which simply converts the single non-zero value for the binary x_{ijt} variable into a time period.

$$f_{ij} = \sum_{t=EFT_{ij}}^{LFT_{ij}} t \, x_{ijt}$$

For any two sequential tasks m and n in jobs i and j respectively (possibly $i = j$), the temporal sequencing constraint forces the succeeding task n to start its execution at a time later than the finishing time for the preceding task m, by requiring the finishing time for task n to be offset by at least d_{ij} (the processing duration for task n) time periods past the finishing time for task m.

$$\sum_{t=EFT_{im}}^{LFT_{im}} t \, x_{imt} + d_{in} \leq \sum_{t=EFT_{jn}}^{LFT_{jn}} t \, x_{jnt} \tag{2.5}$$

2.1.1.5.4 Task Aggregation Constraints

Aggregate tasks describe operations that provide work sites or areas where other tasks may be executed (and are described further in Section 3.4.2.2). Because aggregate tasks encompass other tasks, their processing durations depend on the individual processing durations of their various child subtasks. Execution of an aggregate task begins as soon as its earliest child subtask begins executing, and finishes when its latest child subtask finishes.

Aggregate task processing durations can be determined as part of the preliminary analysis of the job set that determines values for the EST_{ij}, EFT_{ij}, LFT_{ij}, ECT_i, and LCT_i variables.

2.1.1.5.5 Resource Usage Constraints

Unlike some other formulations of this problem, we limit (to one) the number (or quantity) of resources that may service a task (though there may be a choice among alternative resource types).[2]

A task T_{ij} is executing at time t if it finishes at a time $q : t \leq q \leq t + d_{ij} - 1$. During its execution, a task uses r_{ijk} 'units' of resource class k. The resource-limiting constraint presented below ensures that the amount of each resource class k available at every time period t in the feasible scheduling period is no less than the total amount of that resource in use by all of the tasks executing during time period t.

$$\sum_{i=1}^{I} \sum_{j=1}^{N_i} \sum_{q=f_{ij}}^{f_{ij}+d_{ij}-1} r_{ijk}\, x_{ijq} \;\leq\; R_{kt} \quad \forall\;\; t = \min(A_{ij}), \ldots, LCT;\;\; k = 1, 2, \ldots, R \qquad (2.6)$$

2.1.1.5.6 Resource Substitutability Constraints

The ability to allow a task to be performed by any one of a number of resource classes requires some modification of the existing formulation. Each task T'_{ij} that permits a choice among alternate resource classes is replaced by a set of exclusive tasks $T^k_{ij}\ \forall k \in \{k : r_{ijk} > 0\}$. The processing duration for each of these new tasks depends on its corresponding resource class. The necessary changes to the current formulation affect the task finishing and job completion constraints. The alteration of the Task Finishing Constraint (2.3) forces each T'_{ij} to execute only one of its T^k_{ij} tasks, by requiring a single non-zero value among the appropriate binary x^k_{ijq} variables during the time range of $[\min(EFT^k_{ij}), LFT_{ij}]$ and across the set of alternate resource classes for T'_{ij}. The new task finishing constraint for each T'_{ij} thus becomes:

$$\sum_{k \in \{k : r_{ijk} > 0\}} \;\; \sum_{q=\min(EFT^k_{ij})}^{LFT_{ij}} x^k_{ijq} = 1 \qquad (2.7)$$

[2] Note, however, that multiple resource requirements for a single task could be modeled using a hierarchy of resource-requiring aggregate parent tasks.

The second alteration involves a similar modification of the Job Completion Constraint (2.4) to handle the new task finishing requirements for T'_{ij} tasks. The new job completion constraint (for all jobs) thus becomes:

$$x_{it} \leq \left[\frac{1}{N_i}\right] \sum_{j=1}^{N_i} \begin{cases} \sum_{q=EFT_{ij}}^{t-1} x_{ijq} & \text{if } |\{k : r_{ijk} > 0\}| \leq 1 \\ \sum_{k \in \{k:r_{ijk}>0\}} \sum_{q=\min(EFT_{ij}^{k})}^{t-1} x_{ijq}^{k} & \text{otherwise} \end{cases} \tag{2.8}$$

$$i = 1, 2, \ldots, I; t = ECT_i, ECT_i + 1, \ldots, LCT$$

The formulation presented in [Pritsker *et al.*, 1969] is efficient in terms of limiting the number of variables and constraints it requires for a given RCSP. It reportedly further benefits from increased sequencing constraints, longer task durations, and the closeness of the scheduling horizon (LCT) to the optimal schedule completion time. But while this formulation demonstrates success in finding optimal solutions for certain (limited) instances of the RCSP, its suitability to large-scale problems having many jobs and operations, extended scheduling horizons, and abundant slack time, is questionable.

2.1.2 Optimal Solution Approaches

Much of the early work on solving ILP formulations of the RCSP focused on the development of optimal solutions according to singular objective functions. A number of different optimal solution methods have been developed for producing optimal solutions within reasonable time frames, although the success in applying these approaches to practical-sized problems has not been widespread. In this section, we describe a number of these optimizing solution methods.

2.1.2.1 Branch and Bound Approaches

The *branch-and-bound* method for solving constrained optimization problems, such as RCSPs, is based on the idea of intelligently traversing a developing search space. These methods work by starting with a solution of some kind (often a satisficing solution or an optimal solution to a simplified version of the same problem), and then progressively developing, through the tightening of constraints, the best existing solution, until a solution to the desired problem is achieved. This progression towards the optimal solution involves the generation of a tree structure rooted with the original solution, with links representing the introduction of constraints, or some other kind of decision on each parent problem, to connect it to all of its feasible, tighter, or more complete, partial solutions. The search is bounded using the process of *pruning*, where all partial solutions that are provably unable to lead to better quality solutions than the best existing solution are removed from further consideration. The pruning process guarantees that the final solution to the original constrained optimization problem will be optimal.

The branch-and-bound tree can be extended in two different ways, based on the style of search undertaken. A *backtracking* approach performs a *depth-first* traversal of the tree, extending the deepest nodes in the tree by applying a candidate-selection rule that favors more complete partial solutions. Whenever a complete solution is obtained, or a dead end is reached, the procedure backtracks to the best remaining candidate and continues. Once all nodes have either been visited or pruned, the best existing solution is therefore guaranteed to be optimal. The other method of traversal is called a *skiptracking* approach, wherein a *breadth-first* traversal of the tree is performed by extending those candidates that are the farthest from completion.

The main differences between these approaches are the number of search states (nodes in the tree) that must be stored during the process, the ability to generate effective lower bounds, and the effectiveness of the pruning heuristics. A backtracking approach requires less storage of candidates and its quick progression towards complete solutions generates frequent new lower bounds, but its pruning capabilities are not as strong because so many of the other existing candidates have not been developed enough to be proven non-optimal, so that it tends to execute more slowly. On the other hand, the skiptracking approach requires significantly more storage, and while the generation of powerful new lower bounds is less frequent, because the bulk of its nodes are at roughly the same level of development, the pruning heuristics are often more effective, and the process therefore executes more quickly.

Apart from the storage and efficiency concerns, the effectiveness of a branch-and-bound approach depends on a number of factors, namely the quality of the original root solution, the method used to determine a new lower bound in the process of pruning all non-optimal solutions, and the method used to determine which solution subproblem to expand at any point. A general survey of branch-and-bound approaches is provided in [Lawler and Wood, 1966].

The branch-and-bound approach described in [Stinson *et al.*, 1978] generates the tree by progressively scheduling activities forward from the start of the schedule. Each node is expanded by creating a new node for each possible combination of activities that could be scheduled according to both the precedence and resource constraints. At each point, the start time is incremented by the duration of the shortest currently executing activity. When no more activities remain to be scheduled, a complete schedule is obtained. The process of determining the lower bound uses three methods, namely a bound based on completion time that ignores resource constraints, a bound based on the completion time that ignores precedence constraints (by dividing the total remaining resource requirements by the number of available resources), and a bound based on consideration of both resource and precedence constraints. The lower bound and a dominance relation involving the relations between partial schedule nodes is used to prune the tree. The candidate-expansion heuristic uses a vector of six measures that essentially implements a skiptracking approach.

An important conclusion of this work is that the consultation of a variety of measures in the process of selecting the next node to expand resulted in a more efficient exploration of the search space. In addition, the common problem associated with branch-and-bounds approaches, that is, of large, unpredictable variances in the processing time for similar problems was observed. While extensive computational results were obtained, the problem sets were still limited in comparison with practical-sized real-world problems.

Another branch-and-bound approach is presented in [Christofides *et al.*, 1987]. Each node in the tree represents a partial precedence-feasible and resource-feasible schedule that includes the initial activities, and their successors, up to a certain point in time. The candidate node expanded at any point is the one with the longest path to the end of the project (the shortest partial feasible schedule). A candidate is expanded by scheduling the next possible activity that satisfies both the precedence and resource constraints. If unscheduled activities remain, but none can be added, then the candidate is expanded by instead delaying the execution of an already-scheduled activity (or the one that led to the conflict). Backtracking is used to avoid the need for generating multiple branches, and four different lower bounds were considered for pruning the developing tree. Computational results indicated success in solving small problems, especially those with loose resource constraints.

2.1.2.2 Enumeration Approaches

The *bounded enumeration* approach described in [Davis and Heidorn, 1971] shares some similarities with standard branch-and-bound methods. In this case, an RCSP (permitting activities to have multiple resource requirements) is transformed into a directed graph with each activity divided into individual unit-duration subtasks (the number of subtasks equaling the expected processing duration of the activity). Precedence constraints control the linking of the activities, and *immediate* precedence constraints force each sequence of subtasks to be performed together to prevent task splitting. The next step is to produce a table of feasible subset schedules representing the set of unit-duration subtasks that can execute at each time period in the schedule. Resource constraints are ignored at this stage. This table is used to construct another directed graph (called an *A*-network) representing the progression of precedence-feasible and resource-feasible schedules across time periods. The *A*-network can then be used as the basis for solving a *shortest route* problem because the minimal completion time for the schedule is represented by the shortest path through the *A*-network.

Note that the construction of the feasible subset schedules will be expensive for large problems. As a result, two subset elimination procedures are used to minimize the number of subset schedules that are generated by the procedure. The first considers the resource constraints and eliminates any feasible subset schedule whose combined resource requirements exceed the total amount of available resources. The second technique makes use of the latest allowable finishing time for each task, and eliminates any feasible subset schedule that has any task finishing later than that time.

Using this approach, optimal results were achieved for a good portion of a set of small problems (permitting as many as 220 tasks and 5 required resource types per job). The procedure can still result in the creation of a very large number of feasible subset schedules. Again, the applicability of this approach to large scale, practical-sized problems remains questionable. A comparison of three specific enumeration-based optimizing approaches (including the work of Davis and Heidorn) can be found in [Patterson, 1984].

2.1.2.3 Dynamic Programming Approaches

Dynamic programming approaches to solving RCSPs make use of the fact that an optimal solution can be incrementally developed by first constructing an optimal schedule for any two jobs, and then optimally extending that schedule to include another job, until all of the jobs have been scheduled. Scheduling for a set of K jobs is thus achieved by first developing an optimal schedule for a set of $K - 1$ jobs, and then optimally scheduling the remaining job.

An early dynamic programming algorithm for solving simple sequencing problems is described in [Held and Karp, 1962]. While such dynamic programming approaches can significantly reduce computational effort, the chief concern is with the large amount of space required to store the intermediate results calculated by these algorithms.

2.1.2.4 AGSS Domain Applications

The application of ILP techniques to the problem of optimally scheduling various airport ground-servicing activities is explored in a number of projects. Systems of this kind tend to focus on more constrained versions of the overall RCSP, such as the problem of assigning resources of a particular class to a set of tasks. Two such systems are described below.

Mangoubi and Mathaisel's Gate Assignment Optimization Formulation

Mangoubi and Mathaisel have developed an ILP formulation for assigning aircraft to gates that is optimal in terms of minimizing the required passenger travel distance to and from gates [Mangoubi and Mathaisel, 1985].[3] In studies using the actual flight schedule for an average day at Canada's Toronto International Airport, the existing assignment policy was shown to produce travel distances that were 32% higher than the minimum distances produced by the integer program formulation.

It is worth noting that an optimal solution produces a schedule of gate assignments where half of the aircraft are serviced by less than a third of the gates, meaning that the balancing of gate assignments suffers from being ignored by the objective function. In order to better balance the gate usage, additional constraints would have to be added to the formulation.

Babić's Fuel Truck Optimization Formulation

The focus in Babić's work is to determine both the minimal number of fuel trucks required to perform the required refueling operations for a set of aircraft, and the minimal travel distance for those fuel trucks such that no flight is delayed by the refueling process [Babić, 1987]. An ILP formulation with a bi-criterial objective function is defined, and a branch-and-bound technique is used to produce an optimal solution. This technique first finds the minimum number of refueling trucks required by the set of flights and then determines the assignments that produce the minimum amount of travel time for those trucks.

2.1.3 Near-Optimal Heuristic Approaches

While ILP represents a well-understood and established problem-solving method, many of the formulations and algorithms for solving such programs optimally can be extremely opaque, and the formulations themselves can be difficult to specify, modify, and understand. In addition, the amount of computational effort required to solve an RCSP can fluctuate widely across a set of similar RCSPs, meaning that the same approach may incur reasonable computational effort for one problem and exponential effort for another. Finally, the introduction of real-world dynamic complications into large-scale RCSPs greatly increases the difficulty involved in their solution. It appears that a more appropriate use for optimal schedules (assuming that they are available) is for gauging the quality of non-optimal schedules and scheduling approaches, instead of attempting their execution in the real world.

Recognition of the problems of trying to achieve optimal solutions led to a shift in focus towards other methods for finding *near-optimal*, or *approximate* solutions to RCSPs at less computational expense, in terms of both time and space. An heuristic approach represents just such a method. A *heuristic* is a rule that specifies how to make a decision in a particular situation. It can be thought of as a *rule of thumb*. Within the context of the RCSP, a typical

[3] Mangoubi and Mathaisel also developed a heuristic solution approach that is described in Section 2.1.3.1.

heuristic is to schedule those tasks having the earliest possible starting times, or the least available amount of slack time. An heuristic approach to solving an RCSP operates by applying a heuristic (or collection of heuristics) to the set of unsolved subproblems comprising the RCSP to determine the relative priority of each individual subproblem. In this sense, the heuristic serves as a rating function for determining the order in which activities are to be scheduled. The standard heuristic approach first orders the set of activities and then proceeds by assigning resources to each activity in sequence. In some cases, near optimal results can be achieved, and the approach incurs less computational expense. It is therefore more applicable to practical real-world problems having extremely large search spaces.

A *single-attribute* heuristic is distinguishable from a *multiple-attribute* heuristic in terms of the degree of analysis undertaken. Examples of single-attribute scheduling heuristics are the MINEST and MINLFT rules, which assign priority to those activities having the earliest starting times, and latest finishing times, respectively. Another common single-attribute rule is SOF, which assigns priority to the activity having the shortest expected processing duration. The main drawback to single heuristics is the lack of analysis performed. A narrow evaluation of the state of problem solving can allow important developments to go unnoticed and permit serious problems to develop in the future. A multiple-attribute heuristic performs a more extensive analysis of the current state of problem solving, and can therefore act to prevent such problems from developing. A survey of scheduling rules, ranging from simple priority rules to more complex heuristics, is presented in [Panwalkar and Iskander, 1977]. Upper bounds (assuming a minimizing objective) on the quality of a number of approximate heuristic solutions to RCSPs are presented in [Garey *et al.*, 1978].

Returning to the AGSS domain as an example, an heuristic approach to scheduling gate assignments for flights that attempts to minimize required passenger walking distance might work by ordering the flights according to the size of their aircraft, and then assigning gates that incur the shortest walking distance to the largest flights. Gates that are farther away are thus saved for future smaller aircraft. While such a heuristic does not guarantee an optimal assignment of gates (in terms of minimizing walking distance), it does make a reasonable and efficient attempt to do so. This heuristic serves as a simple means of encoding a bit of scheduling knowledge about how to minimize walking distance with an individual gate assignment. It can be used in any situation where the minimization of walking distances is one of the overall scheduling objectives.

The terminology regarding heuristics can get quite specific, depending on whether priority is determined according to one simple aspect of an activity, or whether it is based on the combination of a number of perspectives. Throughout our discussion of heuristics, we make no distinction between the various definitions. We use the terms *scheduling rule*, *dispatch rule*, and *heuristic* to refer to the same thing. A distinction is made, however, between *fixed* variable ordering heuristics (also called a *serial* application) and *dynamic* variable ordering heuristics (likewise, a *parallel* application). A fixed approach performs the ordering at the beginning of the problem-solving process only, while a dynamic approach performs a reordering of all unsolved subproblems following every decision. (The majority of the classical heuristics described and tested in Chapter 5 make use of a *dynamic* variable ordering approach, as does DSS itself.)

A great deal of work has been directed towards the development of heuristics that produce near-optimal solutions, and the determination of the best heuristics to use in certain circumstances. For the most part, however, with the exception of a few specific cases, it has not proven possible to establish a definitive classification for matching a particular class of RCSP with a particular scheduling heuristic.

2.1.3.1 Single Heuristic Approaches

A comparison of some standard heuristics for solving RCSPs was undertaken in [Davis and Patterson, 1975]. In this work, eight single heuristics were applied to a set of single-project, multiple-resource problems, and the results (based on minimized project duration) were compared to optimal schedules produced using the bounded enumeration approach presented in [Davis and Heidorn, 1971]. The results of this comparison indicated that three heuristics, MINSLK (which assigns priority to those tasks having the least amount of slack time), MINLFT, and another, performed the best in terms of achieving the optimal schedule, or coming the closest in percentage to the optimal. The results also showed, however, that the performance of all heuristics suffered when resource contention was high. While this study was applied to small problems, it suggests that heuristics that consider various kinds of time, and the degree of resource usage, generally produce better (near-optimal) results.

An attempt to classify individual RCSPs for the purpose of identifying appropriate scheduling heuristics for their solution is described in [Kurtulus and Davis, 1982]. Two metrics were defined for characterizing RCSPs according to *average resource load* (ARLF) and *average resource utilization* (AUF). A collection of heuristics was tested on a set of sample problems, and the performance of each heuristic was plotted according to a range of values for the two metrics. The results suggested that two heuristics were generally the best performers. The first heuristic, SASP, favored the shortest activity in the shortest project (job), while the second, MAXTWK, favored the activity with the highest combined required resource load (obtained by multiplying the activity resource requirement by the activity processing duration) and cumulative project resource load (obtained by summing the required resource loads for all activities already scheduled within the same project). Interestingly enough, the MINSLK rule produced generally better results in situations where average resource utilization levels were lower, that is, the ratio of requested resource load to supply was in the range [0.6, 0.8].

The results of Kurtulus and Davis were later tested in a heuristic scheduling system designed to solve RCSPs in the Japanese housing construction industry [Tsubakitani and Deckro, 1990]. The analysis described in [Kurtulus and Davis, 1982] suggested the use of the SASP heuristic for solving the housing construction RCSPs. The SASP heuristic and the scheduler within which it was implemented (MPM) were tested against two other heuristic scheduling systems. MPM produced schedules containing shorter makespans in both comparisons. MPM also features a re-scheduling mechanism that can modify a schedule in response to the notification of events resulting from its real-world execution. It is designed to handle as many as fifty simultaneous projects, each containing as many as 100 activities over a 400-day planning horizon. Each activity can require up to fifty resources. Experimental results suggest that MPM “can produce ‘good’ solutions” for multi-project scheduling problems.

Recent work presented in [Lawrence and Morton, 1993] describes a single-heuristic approach to solving RCSPs that attempts to minimize weighted tardiness through the use of a combination of project-related, activity-related, and resource-related metrics. A general priority heuristic is defined that weighs the tradeoff between the cost of delaying an activity on a resource, and the benefit obtained by using that period of delay to assign the resource to some other activity. The benefit of freeing up time on a resource is based on a *resource price* that provides an indication of the urgency of each particular type of resource. Finally, the cost of delaying an activity can be determined by using either the resource demand exhibited by the activity (partially based on the price of required resources), or by the collective resource demand exhibited by all unscheduled activities within the same project. Finally, an iterative schedule-refinement mechanism helps to periodically improve the developing schedule.

The results of this approach were compared against the results from twenty standard scheduling heuristics on a set of some 14,400 individual RCSPs that ranged in size from 125 to 250 activities distributed among five projects, and varied in tardiness penalty, activity duration, and resource requirements. The heuristic described above produced schedules with "significantly" lower average tardiness costs (and weighted delay) than did the standard heuristics, further supporting the notion that consideration of multiple perspectives leads to a more informed decision-making capability.

2.1.3.2 Multiple Heuristic Approaches

The application of multiple heuristics in solving the RCSP represents an extension to the standard single heuristic approach. The goal is to exploit the strengths of a number of different scheduling heuristics in an attempt to increase the chances of producing near-optimal (and occasionally optimal) schedules according to the particular scheduling objective. Additionally, a multiple heuristic approach can be seen as a formal application of the idea of consulting multiple scheduling perspectives. One heuristic might evaluate urgency based on a simple analysis of the time bound constraints on an activity (MINEST, MINLFT). Another might consider the expected activity duration (SOF), while yet another might consider some combination of the two (MINSLK). A scheduler equipped with a variety of such heuristics is better able to react to the developing multi-dimensional topology of the search space.

Multiple heuristic approaches run into the same problems as simple heuristic approaches when it comes to trying to characterize the best heuristic combination to use for solving a given RCSP. The experiments in [Boctor, 1990] do show, however, that the inclusion of certain heuristics can result in the more frequent development of near-optimal, and occasionally optimal, schedules. Notable in these results is the fact that the MINSLK heuristic (which placed at the top in single heuristic comparisons) proved to be the most important of the single heuristics included in any combination. In addition, the MINLFT heuristic (runner-up in the single heuristic comparison) proved to be a valuable companion to MINSLK. Another important result from these experiments is that larger combinations of heuristics demonstrated increased ability to produce higher quality schedules, suggesting that there is a clear benefit to applying a wide range of scheduling perspectives in the process of determining the urgency of each individual activity.

2.1.3.3 AGSS Domain Applications

Schröder's Heuristic Approach to Gate Assignment

An early heuristic approach for producing gate assignments for aircraft is presented in [Schröder, 1972]. Schröder describes a gate-assignment program developed by Lufthansa in 1972 for use at the (then newly constructed) Frankfurt Airport. The objective is to produce near-optimal gate assignments for a set of flights by attempting to minimize the flow of passengers through *hardstands* (gates that require the use of buses to transfer passengers to and from the aircraft), and attempting to maximize the use of *pier* gates (those gates directly reachable by passengers).

The system makes use of a set of scheduling heuristics that contribute to the development of a near-optimal schedule by first assigning priority to each flight based on various characteristics, such as aircraft type and size and the nature of the servicing required. The list of flights to be assigned to gates is then sorted by arrival time, with individual flight priorities used to order flights arriving at identical times. Finally, a near-optimal schedule is produced by assigning gates to the ordered set of flights in such a way as to satisfy all relevant constraints while also maximizing the use of pier gates, and minimizing the use of hardstands.

The focus of this work is the production of near-optimal gate assignment schedules for an existing flight set and airport layout. As a result, a feasible schedule is guaranteed to exist. The original impetus for this work was the need to evaluate the impact of a newly proposed airport terminal layout on an airline's ability to process all of its flights. The system was not intended to assign gates in real time, nor to update an existing schedule in response to an unexpectedly changing environment. It does, however, demonstrate how a set of scheduling heuristics can be used to produce near-optimal resource assignments while avoiding the need for an exhaustive search.

Mangoubi and Mathaisel's Heuristic for Optimizing Gate Assignments

Mangoubi and Mathaisel's heuristic approach for constructing near-optimal gate assignments involves the application of a single heuristic to a set of flights [Mangoubi and Mathaisel, 1985]. The objective is to minimize the total walking distance required by the gate assignments for a set of flights. A single gate-assignment heuristic assigns whichever available gate introduces the shortest amount of walking distance for the passengers on a flight. The heuristic is applied to the list of flights, sorted in order of decreasing passenger volume. Mangoubi and Mathaisel were able to closely approximate the behavior of their optimal ILP formulation (described earlier in Section 2.1.2.4). In the context of this earlier work, the heuristic method is presented as a means of quickly producing near-optimal solutions for larger-scale problems.

Martin-Martin and Mary's CARPPA *Model for Gate Assignment*

The CARPPA Model implements an heuristic approach for producing gate assignments for Air Inter at France's Orly-West Airport [Martin-Martin and Mary, 1986]. The CARPPA Model operates by assigning gates to a set of flights prioritized by arrival time and aircraft size. The primary objective is to produce feasible gate assignments for all flights. Secondary objectives include the achievement of a balanced load across all gates, satisfaction of any existing gate preferences, and the minimization of the required passenger travel distance between gates and airport entry or exit points.

When a gate cannot be found during the time period required by a flight, a series of bounded permutations on the existing schedule is performed, in an attempt to find a relaxed gate assignment for the problem flight that will incur the least amount of passenger travel time. Further permutations can be performed on the completed schedule to achieve the desired load balancing and satisfy gate preferences.

Hamzawi's Gate Assignment Program

Hamzawi describes a computer program for assigning gates to flights that has been successfully tested at a number of Canadian airports [Hamzawi, 1986]. Flights are processed in order of their arrival time, with preference among identical arrivals given to those flights using larger aircraft. The model handles standard airport constraints such as technological compatibility between gate and aircraft, access to government inspection services provided by individual gates (for servicing international flights), and the assignment strategy for each gate adopted by the airport (exclusivity or preferences granted to certain airlines). The main scheduling objective is to maximize the use of specific gates, and minimize both passenger and aircraft delays. The program has been designed to produce schedules for feasible sets of flights and constraints.

An interactive feature is included for allowing specific aspects of the problem to be altered or overridden by the user. The program can be re-run in response to such modification to produce a new schedule. While the system can be used as an operational scheduler, its ability to handle over-constrained problems produced by unexpected events is limited by the fact that the heuristics are tailored to feasible situations.

2.1.4 Multiple Objective Approaches

Many approaches to solving RCSPs have focused on the optimization of one specific scheduling objective. It is more often the case, however, that a range of objectives exists, and that the quality of a schedule must be evaluated according to how well it satisfies a set of such objectives. For example, a *single-objective* scheduling system might attempt to minimize *work-in-process* (WIP) times while ignoring the makespan and the level of resource utilization in the resulting schedule. A *multi-objective* approach would instead engineer its decision-making process to construct a schedule while simultaneously attempting to minimize WIP times and schedule makespan, and maximize resource utilization.

A multiple-objective approach relates to the notion of multiple scheduling perspectives, in that each objective generally corresponds to a particular perspective on the scheduling problem, which must be considered in the process of producing a quality schedule. If certain scheduling perspectives are ignored by the decision-making process, any potential solution will fail to address those concerns, and as a result, the objectives corresponding to those perspectives will not be met.

A goal programming approach can be used in cases where the set of multiple objectives can be arranged according to a strict priority scheme. In this case, each objective is represented by a separate equation. The solution process begins by finding a solution that optimizes the first (and highest priority) objective. After this initial solution is found, any constraints imposed by it are incorporated into the problem formulation, whereupon the solution process continues by focusing on the optimization of the second objective. This approach continues until all of the objectives have been met. Goal programming is useful for problems where each objective has a distinct priority, and cannot be combined effectively into a single objective function. When the objectives are closer-related and less stratified, a goal programming approach will tend to sacrifice the lower-ranked objectives, and thereby lower the degree to which all objectives are met.

Much of the recent work on solving RCSPs using standard OR techniques centers on multi-objective approaches. The work described in [Norbis and Smith, 1988], for example, involves an heuristic method for developing near-optimal schedules that satisfy a set of specific scheduling objectives. A series of levels is defined for representing consecutive orderings of the set of all task-level subproblems for a particular RCSP. As the collection of subproblems moves up through the various levels, it is progressively subdivided according to the objectives defined for each level. Priority rules at each level are invoked to order the subproblems within each group. An interactive feature allows for input from the user to be factored into the problem-solving process through a rearrangement of the subproblem sequence. At the highest level, the subproblems can then be solved in sequence to construct the schedule.

The approach of Norbis and Smith also accommodates dynamic events such as surprise orders, order time bound changes, and changes in resource availability. These developments force specific alterations to the existing schedule and constraints, whereupon the multi-level problem-solving mechanism is re-invoked to repair the schedule. Schedule repair time is kept to a reasonable fraction of the original schedule generation time.

A more recent multiple-objective hierarchical approach is described in [Speranza and Vercellis, 1993]. Here the goal is to model the multiple objectives of the planning and scheduling portions of real-world RCSPs. A *tactical* (planning) level concerning *net present value* costs (inventory, tardiness), and an *operational* (service) level concerning the assignment of resources and execution times to activities are established. This two-stage hierarchy is implemented using three ILP models. The *multi-project model*, partially responsible for implementing the tactical level concerns, is responsible for organizing the timing of the various projects and the distribution of the available resources among the projects. The *shrinking model* provides information to the multi-project model for use in distributing the resources, by highlighting the collective resource requirements of each project. These two models together solve the tactical level planning decisions. The final *detailed project model* takes responsibility for the operational level by assigning resources and execution times to the activities according to the constraints imposed at the tactical level.

A branch-and-bound algorithm is used to solve the ILP formulations corresponding to the three models. The approach has been applied to real-world data from a construction project, consisting of problem sets containing from five to ten projects, with ten to twenty activities per project. While specific experimental results were not included, the running time for the approach applied to the individual problem sets was reported to be “on the order of minutes.” Further work involving the introduction of another level into the hierarchy to take responsibility for the dependability and reliability of schedules during execution is planned.

2.1.5 Other Operations Research Approaches

An interesting alternative approach for solving RCSPs is presented in [Adams *et al.*, 1988], where the RCSP is divided into individual *resource* subproblems, instead of job or activity subproblems. A schedule is produced by sequentially solving a single-machine scheduling subproblem for each resource, and interleaving a re-optimization process after each machine is scheduled. The re-optimization process allows for any conflicts that develop between previously scheduled resources and newly scheduled resources to be resolved before continuing. After the final resource is scheduled, the RCSP has been solved. The priority of the resources (for the purpose of determining scheduling order) is calculated using a bottleneck status check that solves a relaxed single-machine scheduling problem to determine which of the resources is expected to incur the most delay given its demand.

The results of this approach are quite good for a set of problems containing up to ten resources and as many as 500 operations. In fact, this work suggests that the number of operations has only a moderate impact on the computational expense of the approach. But the key factor, of course, is the number of resources. The procedure was also tested against a set of ten standard heuristics, and it produced the best results in 38 of the 40 test problems, while incurring the similar computational expense. The resource-centered perspective taken by this approach shows a great deal of promise as a means of improving other methods for solving RCSPs.

2.1.6 A Summary

With standard OR approaches, an RCSP is generally treated statically. A problem is encoded into a particular representation, an algorithm is applied to it, and a solution of some kind, be it optimal or approximate, is returned. Little attention is given to the dynamic nature of many real-world problems. Aside from discounting the value of optimal solutions, real-world problems also present difficulties for static approaches, in that certain aspects of the problem are subject to change during the process of problem solving. The objective itself becomes a moving target, to which a problem solver must adapt. Only recently has the dynamic nature of the problem-solving process been addressed within the field of OR [Norbis and Smith, 1988].

The approach implemented in DSS accounts for the dynamic nature of the problem by allowing the environment to force changes in the state of problem solving throughout the entire decision-making process. Environmental conditions which affect the problem can therefore be considered at the appropriate times to influence properly the behavior of the system. As the target of the problem-solving process changes with the environment, the problem solver adapts quickly to these changes and alters its existing decision-making strategy in response.

Computational expense plays an important role in the process of solving RCSPs. In situations involving large-scale problems, the performance of many OR algorithms can become unpredictable, and sometimes prohibitively expensive. Given a sufficiently large RCSP, it is not always clear that a standard OR solution approach will be completed in reasonable time. Finally, the opaqueness of many problem representations can make it hard to understand the solution process and alter a problem formulation in response to unexpected developments. By contrast, the amount of required computational effort for DSS is dependent on the number of individual tasks to be scheduled, the number of available resources, and the nature of the order sets (notably the amount of resulting contention for resources), so that its behavior is more predictable.

In terms of knowledge representation, there is much similarity between the ILP formulations used by classical OR approaches and the various resource and task models employed by DSS. Furthermore, the basic DSS problem-solving method is essentially a multiple-attribute, dynamic heuristic approach that focuses on the most urgent unsolved subproblem at any point in time. Moreover, DSS endeavors to solve dynamic RCSPs completely on its own, without relying on input from an outside user.

2.2 Artificial Intelligence Approaches

In this section, we discuss two kinds of AI approaches for solving RCSPs. We first describe the class of expert systems, in which an heuristic approach produces a near-optimal result by systematically applying both generic and domain-specific scheduling knowledge to the RCSP. Knowledge-based scheduling, a more recent AI approach, combines domain-independent scheduling knowledge with meta-level control knowledge to produce a more flexible and sophisticated control scheme for solving potentially dynamic RCSPs.

2.2.1 Expert System Approaches

Expert systems implement a formal problem-solving approach characterized by multiple-heuristic decision making. An expert system is comprised of three important entities. The *rule* or *knowledge base* contains a set of *situation–action*, or *if-then* rules codifying the heuristics for solving a particular class of problems. The *working* or *short-term memory* stores all of the initial information about a particular problem as well as the developing solution. Finally, an *inference engine* controls the process of incrementally selecting and applying the most promising applicable rules from the rule base. The cyclical actions of the inference engine serve to augment the working memory throughout the decision-making process. Updates to the knowledge base reflect the particular actions specified by the rule. The process finishes successfully when the working memory contains a solution to the overall problem. Considerable work has been devoted to the development of expert systems [Weiss and Kulikowski, 1984], but for the purposes of this discussion, we classify them simply as structured heuristic problem solvers.

In the past few years there has been a significant amount of work done on the development of expert systems for solving specific RCSPs in the airport ground-service scheduling domain. Pan American World Airways uses an heuristic-based system to assist in the scheduling of gate assignments at New York's John F. Kennedy Airport [Riccio and Ron, 1985]. Texas Instruments has developed the *Gate Assignment Display System* (GADS), an expert system which has been used by United Airlines to assign gates to flights at O'Hare Airport in Chicago and Stapleton Airport in Denver [Shifrin, 1988]. Texas Air produces gate assignments for all Eastern and Continental Airlines flights at Houston and Miami International Airports using a system called GATEKEEPER [Fisher, 1988].

We discuss below two additional expert systems that have been developed for solving the gate-assignment problem.

GAP*: Gosling's Gate Assignment Program*

Gosling's Gate Assignment Program (GAP) is an expert system designed to assign an airport's arriving and departing flight aircraft to the set of available gates [Gosling, 1990, Gosling, 1982]. The process of assigning a flight to a gate is constrained by the kinds of aircraft the gate can accommodate, the existing assignments of aircraft to neighboring gates, and considerations involving the time required for transferring both passengers and baggage between gates.

GAP is an event-driven expert system equipped with a rule base that contains rules for initially selecting candidate gates for flights, and also for reassigning flights to other gates in response to late arrivals, delayed departures, or equipment changes. In response to unexpected events of the kind listed above, GAP propagates the necessary changes throughout the developing schedule, by preempting previous gate assignments and introducing delays to selected flights.

The focus of GAP is on the actual gate-selection process. Given a flight, and a collection of variously constrained gates and existing gate assignments, the goal is to modify the current schedule in such a way as to provide a gate assignment for the new flight (possibly accompanied by some modifications to the schedule) that maintains its feasibility without overly deteriorating the quality of the existing schedule.

As opposed to other systems designed to produce gate assignments, GAP is capable of handling the contention that can arise in over-constrained problems. Its rule base encodes extensive scheduling knowledge that can be applied to resolve conflict situations through modification of the existing schedule. Other systems that merely order the flights and then assigns under the assumption that the problem will not be over-constrained, are more susceptible to encountering difficulties that result in lower-quality solutions.

GATES*: Brazile and Swigger's Gate Assignment Program*

The GATES program is an expert system designed to perform the airport gate assignment task [Brazile and Swigger, 1988]. It has been used successfully to produce gate assignments for Trans World Airlines (TWA) at New York's John F. Kennedy Airport.

GATES is able to generate initial gate-assignment schedules from aircraft flight timetables, and modify those schedules as changes within the environment are encountered, such as flight delays, weather changes, and equipment failures. It handles a variety of constraints, including physical constraints involving equipment compatibility, policy constraints involving matching government inspection services to particular types of flights, and timing constraints involving service durations. It also attempts to minimize airport congestion through some degree of load balancing across the set of gates.

Priority is given to flights that are the most heavily constrained. For example, gates are first assigned to flights using large aircraft such as Boeing 747's, because fewer gates are equipped to handle larger aircraft. The rule base is designed to favor the choice of gates that do not violate any of the hard constraints or preferences, such as providing a sufficient amount of time between successive assignments at the same gate for aircraft to get in and out of the apron area. When no gate can be found for a flight, GATES can either delete optional constraints, reduce non-critical preferences in an attempt to find another possible gate, or go into a backtracking mode that can preempt previous gate assignments. As a last resort, a complete re-scheduling process can also be initiated.

Reactive scheduling in response to unexpected environmental changes is handled by first determining the conflicts that result from such changes, and then initiating the backtracking process. The extent of any backtracking can be regulated by the user to limit the amount of time spent modifying an existing schedule.

The GATES program implements an advanced heuristic approach to handling the airport gate assignment problem in two important ways. First, an attempt is made to focus the attention of the system towards those subproblems that are most highly constrained, and thus expected to be more difficult to solve. The hard choices that potentially require a great deal of modification to the schedule are therefore made earlier, minimizing the impact of any required backtracking. The less constrained subproblems are only solved when there is nothing more important left to handle. Second, the rule base is designed with the intention of being able to address over-constrained situations that may result from unexpected events such as resource failures and flight delays.

2.2.2 Knowledge-Based Approaches

Despite the widespread importance of the RCSP for so many different fields, few generic scheduling systems have been designed and implemented for solving such problems, be they dynamic or not. In this section, we describe the state of the art in knowledge-based scheduling, and present a history of the important developments within the field of AI for solving RCSPs that have led to the design of DSS.

The major developments in the field of knowledge-based scheduling can be organized into three stages. Initial work by Mark Fox on the ISIS (*Intelligent Scheduling and Information System*) treated the scheduling process as a constraint-directed search, and resulted in the first significant knowledge-based program designed to solve job-shop RCSPs [Fox *et al.*, 1982, Fox, 1983, Fox and Smith, 1984]. Later work by Stephen Smith on the OPIS (*Opportunistic Intelligent Scheduler*) introduced the idea of considering multiple perspectives in the scheduling process, to apply the appropriate information to particular decision-making points [Smith and Ow, 1985, Ow and Smith, 1986, Smith *et al.*, 1986b].[4] Finally, Norman Sadeh's MICRO-BOSS (*Micro-Bottleneck Scheduling System*) exhibited an opportunistic, activity-based scheduling approach that helped achieve a finer degree of opportunism, thereby allowing for the rapid shifting of attention from one location to another within a developing schedule, thereby allowing potential conflicts to be resolved as early as possible [Sadeh, 1991].

In the following sections, we describe the development of these three approaches in greater detail, and discuss their major contributions to the field of knowledge-based scheduling.

2.2.2.1 Scheduling as Constraint-Directed Search

ISIS was the first generic, knowledge-based system for solving RCSPs. By viewing the scheduling process as a constraint-directed search of the space of possible schedules, ISIS emphasized the importance of representing a wide range of constraints and exploiting them throughout the scheduling process. The focus of ISIS is therefore directed to the representation and manipulation of constraints that narrow the search space of potential schedules for solving RCSPs.

ISIS implements a four-phase hierarchical scheduling approach. In the first phase, it selects the order with the highest priority from among the set of all orders, and in the second phase, analyzes the resource requirements for that order to constrain later searching during subsequent phases. Using a beam search, the third phase constructs a sequence of both operations and resources for the order's developing process routing. Existing constraints on the order and its operations are used during this phase to generate legal extensions to the best existing process routing, and help the beam search rate and prune the set of all candidate routings. The result of the first three phases is a process *and* resource routing for an order that lacks only the specific machine reservation times for each operation. Individual machine reservation times are determined in the fourth (and final) phase.

ISIS instantiates a process plan for an order (in the third phase) by sequentially allocating resources to operations by extending either forward from the first operation, or backward from the last. As a process routing is constructed, no amount of slack time is maintained within the developing schedule, unless an unavoidable wait for a resource must be accommodated. Service times for operations are compacted to minimize work-in-process time for the order. When conflicts are detected, the relevant constraints are identified and relaxed, and the process routing involved in the conflict is corrected.

ISIS operates under a single, order-centered scheduling perspective. Orders are selected one by one to have their operations scheduled according to their priority. Decisions regarding the selection of resources and reservations simply attempt to satisfy the constraints relating to the order (to properly sequence the operations and meet the due date). Potential constraints on the resources (minimizing set and travel times), and among operations (the availability of required

[4] See [Smith *et al.*, 1986a] for a discussion of issues concerning both the ISIS and OPIS systems.

resources), are (at best) only partially addressed by the heuristics used to select resources and reservations.

The major advantages of this work are its contributions in the area of constraint formulation, and the use of constraints to narrow an enormous search space. The disadvantages center on the static nature of the control of the system and its lack of flexibility, namely in its use of a beam search that cannot anticipate future scheduling problems. Furthermore, its sole focus on an order-based scheduling perspective limits its ability to deal with important resource-based conflicts or inter-order concerns. It is therefore susceptible to the problem of not recognizing potential problem situations until after a substantial amount of problem-solving work has already been done, and producing schedules with a heavy backup at bottlenecked resources.

2.2.2.2 Exploitation of Multiple Scheduling Perspectives

The OPIS system represented a major step forward in the development of knowledge-based scheduling systems by identifying and demonstrating the utility of viewing the RCSP from multiple perspectives. Smith, *et al*, have defined and experimented with what they call *order-based* and *resource-based* perspectives [Smith and Ow, 1985]. Within the process plan template for a job, sequencing constraints provide an order-based perspective used by the scheduler to ensure that the sequencing of tasks within a process plan of that type is legal. Such a perspective, however, provides only a limited view of the overall scheduling problem. A resource-based perspective presents a view of the scheduling problem that highlights the contention that exists among tasks requiring the same resources during similar times. The consultation of a resource-based perspective, *in addition to* an order-based perspective, has proven to be useful in providing an informed view of the overall problem that allows the scheduler to consider the current contention for resources *and* the proper sequencing of tasks in the process of making important scheduling decisions.

The opportunistic nature of the scheduling approach used by OPIS derives from its ability to shift its focus of attention according to conditions within the changing environment, which are highlighted by its consideration of resource-based and order-based scheduling perspectives. Upon receiving a new order to schedule, OPIS performs a capacity analysis based on the updated demand for resources. If a sufficient level of contention for any particular resource class results from the introduction of the resource requests for the new order, a resource-based scheduling strategy is invoked to satisfy all outstanding requests for the resources now exhibiting bottlenecked status. A *Resource Scheduler* performs the task of extending completely the schedule for any resource class that reaches the sufficient level of bottleneck status at any time during the scheduling process. When the resource-based strategy is not required by the changes in resource demand introduced by a new order, an order-based scheduling strategy (the ISIS scheduler) is invoked to secure all of the required resources for the order. Any adjustment to the current decision-making strategy can therefore only occur between the complete scheduling of either an order or a newly bottlenecked resource. The *Order Scheduler* handles the scheduling of all operations requesting non-bottlenecked resource classes, by either scheduling them in their entirety, or extending existing order schedules forward or backward from previous resource-based scheduling decisions. When a conflict is encountered, the affected order and resource schedules are repaired by the appropriate schedulers.

While OPIS is capable of tailoring its problem-solving approach depending on certain conditions existing within the current state of problem solving, it over-applies its response to these situations. Once it selects a particular response, it employs a specific decision-making strategy to make a series of important scheduling decisions before it considers its next step. The *Order Scheduler* either finishes or completely generates the scheduler for an order. The *Resource Scheduler* either extends or completely generates the schedule for a resource class. It is therefore unable to react immediately to the implications of each of the many decisions made by both the order and resource schedulers, until they are finished executing, and the entire state of problem solving has been reevaluated. Much can happen as the result of the many individual actions taken by the order and resource schedulers, or as the result of environmental developments that occur in the meantime. These developments, potentially leading to more difficult problems in the future, cannot be addressed by OPIS as soon as they should be, especially when working within a dynamic environment.

Furthermore, while OPIS makes use of two different scheduling perspectives, it uses only one of these perspectives at any time, depending solely on the current level of contention for the resource classes required by the orders to be scheduled. The individual scheduling decisions in OPIS thus rely on a single perspective on the problem, either resource-based or order-based, and the decision as to which scheduling perspective to apply in a given situation is determined solely from a resource-centered perspective based on resource contention. Other concerns about the state of problem solving are ignored. When the resource-based approach is applied to a bottlenecked resource class, reservations are selected to optimize a particular resource schedule, thereby ignoring the concerns of the order involved (such as the minimization of flow time). Similarly, when the order-based approach is applied to an order, the optimization of resource schedules is ignored (such as the minimization of fragmentation and setup time), in favor of increasing the quality of the order schedule.

Finally, because the OPIS *Order Scheduler* is just the basic ISIS scheduler, the only potential for slack to be introduced is when an unsecured resource of a low enough contention level is not immediately available, or when a bottlenecked resource has been secured by the *Resource Scheduler*, with no consideration of any adverse effect on the order's schedule. Otherwise, there is little room made intentionally available for maneuvering when the current problem-solving situation unexpectedly changes.

OPIS succeeded in demonstrating the utility of exploiting multiple perspectives on the RCSP, but it falls short of making fully effective use of them. The level of opportunism remains at too low a level, making its applicability to dynamic environments questionable.

2.2.2.3 Micro-Opportunistic Activity-Based Scheduling

The activity-based approach employed by MICRO-BOSS is based on the idea that the granularity of opportunism exhibited by a system, that is, the lowest level at which problem-solving decisions are made, has a major impact on the ability of a system to respond to the effects of its own internal decision-making process, as well as any unexpected external events. A distinction is made between *macro* and *micro* opportunism. Problem solvers that periodically pause at certain points throughout the decision-making process to analyze the current state of problem solving and determine a particular strategy to impose until the next analysis point, exhibit a higher degree, or *macro* level, of opportunism. If a problem solver is able to shift its focus of attention immediately after every decision, then it exhibits a finer degree, or *micro* level, of opportunism.

MICRO-BOSS employs a micro-opportunistic decision-making approach, which allows it to follow more closely the developing state of problem solving, by viewing the entire scheduling problem as a collection of individual subproblems, each representing a single resource request. Each pending subproblem is rated according to the demand on the particular class of resource it requires. The heuristic for selecting which subproblem to address at any point first determines the resource class experiencing the highest degree of contention at any time, and then chooses the subproblem whose task depends the most on a resource of that class during the period of contention. The queue of pending subproblems is adjusted after each resource assignment so that the implications of each new reservation can be incorporated into the scheduler's view of the current state of problem solving. This activity-based approach achieves a fine level of opportunistic granularity, and prevents the possibility that the system will inappropriately boost the urgency of any related, but less important subproblems, merely because they also require the same bottlenecked resource class. MICRO-BOSS is thus able to direct its attention within the developing schedule to focus quickly on the most urgent remaining subproblem at any time.

A micro-opportunistic approach allows the scheduler to react immediately to developments within a changing environment. This tends to limit the amount of processing that results from each decision, because each atomic problem-solving activity produces only a quick, minor adjustment to the current strategy. It also avoids the need for major periodic analyses of the state of problem solving coupled with possibly drastic redirections of effort. The macro-opportunistic approach, as exhibited by OPIS, spreads out the strategic decision-making points, thereby minimizing the frequency with which it pauses to reevaluate its current decision-making strategy. The problem with the macro-opportunistic approach, especially within the context of a dynamic environment, is the potential for the problem solver's focus of attention to diverge more easily from the current state of the environment, thereby leading to situations where significant amounts of processing may be required to get the problem solver back on track within the changed environment.

As it turns out, MICRO-BOSS actually employs a rather narrow view of the state of problem solving within its set of variable-ordering and value-ordering heuristics. Despite the operation-centered (activity-based) granularity of its subproblems, MICRO-BOSS relies solely on an order-based perspective when selecting resource reservations for individual operations, and solely on a resource-based perspective when determining the urgency of the outstanding operations. The lack of consideration for the concerns of the resources when selecting reservations means that the quality of the resource schedules is sacrificed for the quality of the order schedules. While the degree of resource contention is useful in estimating subproblem urgency, there is no consideration of the orders involved, meaning that such concerns as order schedule flexibility and temporal urgency are completely disregarded. The drawback to this approach is that while the scheduler can quickly respond to changes in the state of problem solving brought about by its own decision-making process, these decisions are actually based on a very limited scope of information.

Like ISIS and OPIS, MICRO-BOSS also attempts to eliminate any existing slack time from within its developing order schedules to help minimize flow time (and inventory costs). The higher cost of backtracking can be accepted in return for a minimization of inventory costs and flow time. But the flexibility that is lost cannot then be used to alleviate the severity of any necessary future backtracking.

Work on MICRO-BOSS demonstrated that improvements in the performance of a scheduler could be obtained by using a micro-opportunistic, activity-based problem-solving approach.

By not adhering to a rigid, prearranged control strategy, and instead shifting attention from one serious localized bottleneck situation to another, MICRO-BOSS is able to produce schedules more efficiently than the macro-opportunistic OPIS system. It is not, however, equipped with the mechanisms necessary for understanding the full implications of unexpected real-world events that violate its initial assumptions about the problem.

2.2.3 A Summary

The constraint-directed search implemented within ISIS provides a valuable starting point for any knowledge-based scheduling system. It highlights the importance of considering the impact of the various constraints on the problem-solving process, and demonstrates that an enormous search space can be managed by using the constraints to bound the search process. Despite its reliance on a single scheduling perspective throughout the scheduling process, ISIS represents an important first step in the development of efficient knowledge-based scheduling systems.

The development of the multiple-perspective decision-making capabilities in the OPIS system identified the value of viewing the RCSP from both order-centered and resource-centered perspectives. These perspectives highlight the various conflicts between orders, operations and resources, which are often ignored at key decision-making points. OPIS also demonstrated the benefits of applying opportunistic problem-solving techniques to the scheduling problem, to enable a scheduler to react to the dynamic nature of the RCSP, where the state of problem solving can change with every scheduling decision.

MICRO-BOSS represents the move towards a highly opportunistic, activity-based problem-solving approach that enables a scheduler to respond immediately to changes in the state of problem solving. What is missing from this approach, however, is the full consideration of a broad range of relevant scheduling perspectives at all key decision-making points. The system responds quickly and directly to the changing state of problem solving, but it does so based on a limited view of the environment.

In terms of their ability to handle dynamic, real-world environments that generate a variety of unexpected events, each of the three systems described above fails to treat the conflicts that are created by such events in a truly reactive fashion. Conflicts caused by resource failures and order receptions are processed by special conflict-resolution mechanisms that immediately perform some form of backtracking or re-scheduling without first placing the conflict within the context of all currently outstanding problem-solving activities. To perform successfully in a dynamic environment, a scheduler must be able to determine accurately the relative urgency of every scheduling task, whether it is predictive or reactive in nature.

Another feature of ISIS, OPIS, and MICRO-BOSS, is their consistent attempt to minimize order flow times by compacting their developing schedules by removing all accrued slack time. The flexibility that is removed by this action is then unavailable to help resolve future conflicts, causing such activities to require greater computational effort in backtracking and re-scheduling.

With the design of DSS, we have incorporated the consideration of a broad range of perspectives throughout the scheduling process, to facilitate a broad understanding of the state of problem solving at all times. In addition, we maintain flexibility within developing schedules by including available slack time, to provide flexibility for dealing with unforeseeable events.

DSS thus represents a continuing step forward in the development of reactive knowledge-based scheduling systems.

In the following two chapters, we describe the various generic structures provided by DSS for representing a broad range of real-world RCSPs, and then present in detail the entire DSS scheduling approach.

CHAPTER 3

THE DSS REPRESENTATION SCHEME

In this chapter, we describe the generic entities provided by DSS for representing resource-constrained scheduling problems. We first discuss some additional assumptions about DRCSPs as implemented in DSS, followed by an explanation of the resource and product class specifications and the various aspects of the DSS task structure. Finally, we describe the range of hard and soft constraints supported by DSS. The purpose of this chapter is to emphasize the generic qualities of these representational mechanisms.

3.1 Some General Assumptions

In addition to the assumptions about dynamic resource-constrained scheduling problems provided in Figures 1.3 and 1.4, we first present some further assumptions about the DRCSP model implemented within DSS.

- Servicing operations can require at most one kind of resource class.

- All resources are non-consumable. DSS supports resources that represent machines or equipment used to perform specific tasks on a product. Consumable raw-material resources are not supported.

- The performance attributes of a resource do not change over time. The same tasks will always take the same amount of time to execute on the same resource.[1]

3.2 Resources and Shop Locations

DSS provides for the definition of both securable non-consumable *resources* and non-securable *shop locations*. While resources are of limited supply and must be reserved for the operations that use them, shop locations are available in effectively unlimited supply. There are no constraints on the use of shop locations, and they need not be reserved by the operations that use them. Shop locations serve as place holders for representing the behavior of resources as they interact with the factory environment.

We now discuss both of these entities in further detail.

[1] Note that a logical and interesting enhancement to our current domain model would be to allow for various resource performance attributes to change over time, thereby allowing for the representation of more real-world situations, such as the impact of weather and machine operator variances on resource behavior.

3.2.1 Stationary and Mobile Resources

Many real-world RCSPs involve both stationary and mobile resources, distinguishable by their ability to move about the factory environment. The servicing of a product may be performed by a mixture of both stationary and mobile resources. When a stationary resource is used, the product is moved from its current location within the factory environment to the location of the assigned resource, at which point the product is serviced by the resource. With a mobile resource, on the other hand, the product remains at a particular location, while the mobile resource moves from its current location to the location where the product currently resides. The process of determining the required locations is controlled by the various contextual constraints discussed in Section 3.5.1.6.

3.2.2 Shop Locations

In addition to stationary and mobile resources, a layout may include instances of a class of stationary, infinite capacity non-schedulable resources that we call *shop locations*. Shop locations play an important role in the behavior of mobile resources. They provide special locations within the factory environment for resources to be setup or reset, or for products to be serviced. The important distinction between a stationary resource and a shop location is that there is no queue for the shop location. A shop location may be used by any number of resources at any time, and need not be reserved by the resources for that use. Furthermore, a shop location may provide an infinite supply of materials for use by the resources and products.

Consider the following two examples from the airport ground service scheduling (AGSS) domain (introduced in Section 1.3.1 and described fully in Appendix A.1.1). Baggage trucks are used in the process of loading and unloading baggage from an aircraft. The resource plans for these tasks include the filling or emptying (respectively) of the baggage truck at a particular location within the airport, specifically an outlet point for the airport (or possibly an airline) baggage claim area. These outlet points are shop locations. A second example (from the same domain) involves the fuel trucks that must be refilled periodically (prior to each refueling operation) to be able to refuel their assigned aircraft. A fuel tank used to refill the fuel trucks is also a shop location.

3.2.3 Resource Behavior and Performance

Dynamic RCSP environments allow a wide variety of behavior to occur in the process of executing a schedule. Certain behaviors may have a substantial impact on the decision-making strategy employed by a scheduler. An important class of behavior involves the actions of the resources. Their behavior strongly affects the ability of a scheduler to satisfy its resource-reservation requests. We identify a number of important kinds of behavior that may be exhibited by the resources in a dynamic factory environment. These behaviors are classified into specific states that are maintained during explicit and discrete time periods. We now present these behavior states and describe their various implications to the scheduler.

- `UNUSED`: indicates that a resource is available to perform a product task.

- `SERVICING`: indicates that a resource is performing a generic `SERVICING` activity that corresponds to a particular product task.

- `SETUP`: indicates that a resource is performing a generic `SETUP` activity associated with a particular product task. The duration of such activity may depend on the previous use of the resource.

- `RESET`: indicates that a resource is performing a generic `RESET` activity associated with a particular product task.

- `EN ROUTE`: indicates that a resource is performing a generic `EN ROUTE` activity associated with a particular product task. The duration of such activity may be modified in the process of refining a reservation.

- `OFF LINE`: indicates that a resource is unavailable due to a work shift or periodic maintenance constraint.

- `DOWN`: indicates that a resource is permanently unavailable until the end of time.

- `PRE BREAKDOWN`: indicates the time spent in the process of performing the tasks associated with a reservation by a resource that fails in the course of performing its servicing duties. It is conceptually identical to the `DOWN` classification.

As was mentioned in Figure 1.3, we assume that all tasks are completed satisfactorily, barring resource failure. Our model does not allow for the rejection of an order due to quality concerns.

In the event of a resource failure, a special knowledge source is immediately triggered (a byproduct of the official notification of the scheduler), with the purpose of canceling all outstanding reservations for the failed resource and updating the various system metrics that are based on current levels of resource availability. Any task currently being performed at the time of a resource failure notification is treated as incomplete, and must be rescheduled on some other resource. Therefore, a resource failure in the middle of a task execution is the only reason a task may not be satisfactorily performed by a resource.

For the purpose of checking the feasibility of a schedule, DSS is able to analyze the state transitions of its resources upon the completion of their servicing activities to ensure that all relevant hard constraints were met in the process.

3.3 Products

The *product* is basically a place holder and indirect address for the storage of domain information used by DSS throughout the scheduling process. Products constrain the selection of resources that can be used for their servicing or production; they also inform the methods responsible for determining task-processing durations.

3.4 Tasks

DSS provides two kinds of tasks for representing the operations performed in the course of servicing or producing a product. The first, called a *product task*, describes a specific operation performed directly on a product. It is important to note that some product tasks do not require the services of a resource. These non-resource-requiring product tasks represent scheduled activities that are part of the overall servicing or production of a product that are achieved without using any resources. In the AGSS domain, the deplaning and enplaning of the passengers on an aircraft may not require any equipment apart from the jetway that is provided by a gate. In this case, the deplaning and enplaning tasks are indicated by product tasks that occur as part of the overall servicing of a flight, but do not require the reservation of any specific resources. Similarly, in factory-scheduling environments, certain products may require periods of cooling or settling prior to moving on to the next stage of production. These tasks also do not require any specific resources to be reserved. For the most part, our discussion of product tasks will always refer to both resource-requiring and non-resource-requiring tasks. Where there is a specific need to differentiate between the two kinds of tasks, we will do so.

The second kind of task, called a *resource task*, describes the specific ancillary operations associated with the execution of a product task. Any setup or reset activity required by a machine to prepare for a particular type of service is represented as a resource task, as is the process of moving a mobile resource through the factory environment. Each resource-requiring product task has a corresponding resource task representing the actual servicing activity to be carried out by a resource. Product tasks provides am order-based view of operational activity, while resource tasks provide a more detailed resource-based view.

In the rest of this section, we discuss how these kinds of tasks are used to help represent real-world resource-constrained scheduling problems.

3.4.1 Resource Tasks

In DSS, every resource task is an instance of one of four generic resource activity classes. These classes provide the basis upon which a set of six higher level domain-independent resource tasks have been defined. These generic resource tasks in turn provide the basis upon which domain-specific resource tasks may be defined for a particular application. We present below the four generic resource activity classes and the six domain-independent resource classes built upon them.

- `EN ROUTE`: used by mobile resources for traveling through the factory environment
- `SETUP`: represents all of the work required to prepare a resource to perform its main servicing task
- `SERVICE`: indicates the main servicing task performed on a product
- `RESET`: represents all of the work required to reset a resource following the performance of its main servicing task

The six generic resource tasks provided by DSS are described as follows:

- `GOTO-SETUP-LOCATION`: an instance of the `EN ROUTE` activity class used for moving a resource to its designated setup location
- `SETUP-RESOURCE`: the single generic instance of the `SETUP` activity class
- `GOTO-SERVICE-LOCATION`: an instance of the `EN ROUTE` activity class used for moving a resource or product to its designated servicing location
- `SERVICE-PRODUCT`: the single generic instance of the `SERVICE` activity class

- GOTO-RESET-LOCATION: an instance of the EN ROUTE activity class used for moving a resource to its designated reset location
- RESET-RESOURCE: the single generic instance of the RESET activity class

Each resource task defined for a specific domain must be an instance of one of the preceding generic resource tasks. Figure 3.1 presents the entire activity class inheritance hierarchy for DSS.

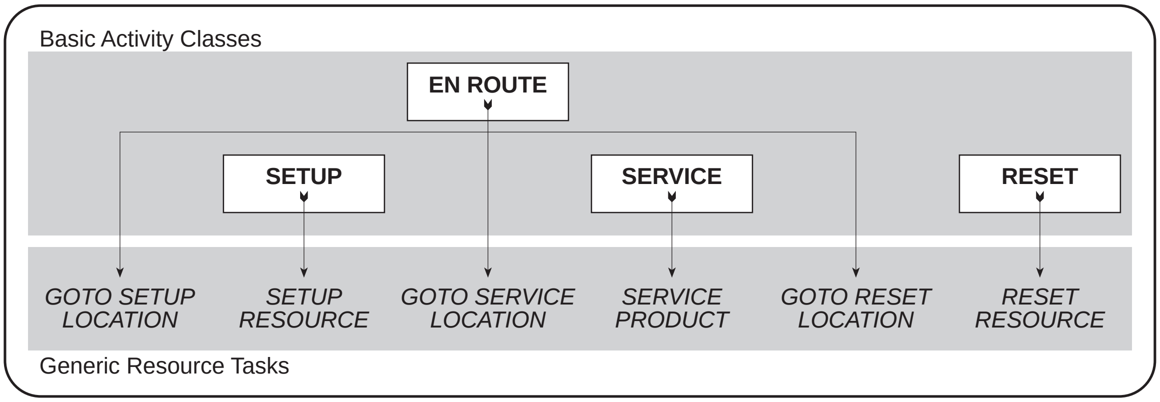


Figure 3.1. DSS Activity Class and Generic Resource Task Hierarchy.

Resource tasks are performed in purely sequential order, with no idle time in between. Figure 3.2 demonstrates the possible ways in which the generic resource tasks may be combined to define legal resource plans. When a particular resource plan instantiated as part of a reservation, starting and finishing times are assigned to each resource task in the plan. If the performance of some resource task within the plan is not required, its starting and finishing times are left empty within the resource plan instantiated for that reservation. In domains involving the use of mobile resources, it is possible that travel time will not be necessary in some situations, for example when a resource is already positioned at its required location. Similarly, a setup operation may not be required if the current state of a resource is such that it is able to immediately service another product without having to alter its current settings.

Domain-specific instances of the generic resource tasks are grouped into resource plan sequences and associated with product tasks by their resource class, so that different resource plans may be used to perform the same product tasks, depending on the kind of resource that is secured. For example, in the AGSS domain, it is possible to refuel an aircraft by using either a pump truck or a fuel truck, depending on the kind of equipment provided by the gate where the refueling is to be performed. The resource activities performed by the pump truck and the fuel truck are not the same. The pump truck simply reports to its designated gate and then refuels the aircraft by pumping fuel from an underground tank directly into the aircraft. A fuel truck, on the other hand, must initially report to a fuel tank and take on enough fuel to refuel the aircraft before reporting to its designated gate and to refuel the aircraft.[2] Each product task

[2] Note that this approach to refueling using fuel trucks is a simplification of the actual AGSS domain. In the real world, the trip to the fuel tank is a periodic task that is scheduled as needed, depending on the number of sequential refueling activities undertaken by a particular fuel truck.

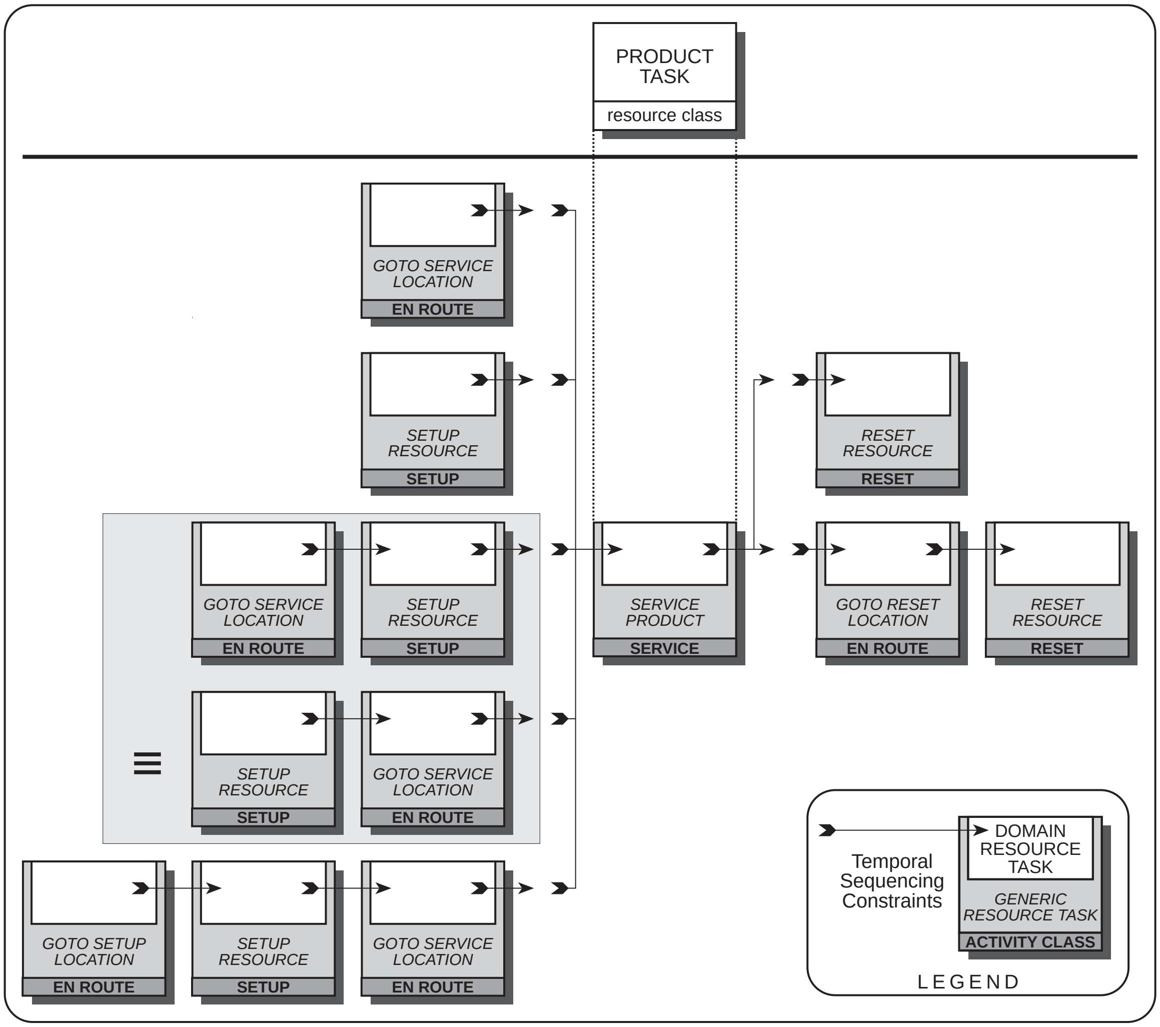


Figure 3.2. DSS Resource Plan Grammar.

that requires a resource must provide a resource plan for each type of resource that may be used to perform the task.

3.4.2 Product Tasks

A product task corresponds directly to a generic `SERVICE-PRODUCT` resource task, and provides the means for viewing an operation from the viewpoint of an order or product. From that perspective, the performance of a task by a resource is achieved in a single step, while from the resource perspective, and more importantly, to the scheduler, it may actually consist of a sequence of steps. Product tasks that do not require resources place constraints on the starting and finishing times for neighboring tasks, and the release and completion times for related task networks. These product tasks have no associated resource tasks.

In this section we describe certain product task attributes that have an important impact on the scheduling process.

3.4.2.1 Shift Preferences

An important requirement for many scheduling domains is the need to represent shift preferences for product tasks. These preferences indicate the desired point at which to schedule a task within its current allowance. For some kinds of tasks, it is better, from the perspective of the order to which they belong, for them to be scheduled as early as possible. For other tasks, later scheduling is preferred. DSS is equipped with the means of representing both early and late shift preferences, to broaden the range of scheduling problems it can handle. The shift preference for a product task is used by the reservation-securing knowledge sources responsible for securing the resource required for performing the task. These knowledge sources attempt to find reservations that are as early as possible for early-shifted tasks, and as late as possible for late-shifted tasks.

Considering again the AGSS domain, the product task responsible for unloading baggage from an aircraft should be scheduled to occur as soon after the arrival of a flight as possible, thereby allowing the arriving passengers to quickly claim their checked baggage after deplaning. The baggage-loading product task, however, should be scheduled as close to the departure of a flight as possible, to give departing passengers as much time as possible to get their checked baggage loaded onto the flight.[3]

3.4.2.2 Aggregate Tasks

An important issue in factory-scheduling environments is the relationship between a work area and the activities that must be performed there. In the AGSS domain, the gates at which aircraft are docked serve as work areas for the collection of mobile resources that assemble to help service the aircraft. The gate must therefore be secured for a block of time that encompasses the earliest time that any of the mobile resources may be scheduled to begin their tasks, and the latest time that any of them may be scheduled to finish. In this way, it is guaranteed that all of

[3] Figures A.6 and A.7 present the resource plan and task specifications (respectively) for the baggage-unloading task defined in our implementation of the AGSS domain. Figures A.4 and A.5 provide the same information for the late-shifted baggage-loading task.

the various tasks to be performed by the mobile resources will be executed during the block of time for which the gate is secured.

A similar relationship exists when a single resource must be used to perform a series of different product tasks on the same product. Again, each of the tasks must be performed during the block of time for which the resource has been secured. If these tasks are represented as individual product tasks each requiring the same kind of resource, then the scheduler may secure different resources to perform each product task, thus violating the constraint that the same resource be used for all tasks.

To handle the situations described above, we have developed a special kind of product task that we call an *aggregate*. An aggregate task is a product task that encompasses a collection of 'child' product tasks, each of which may or may not require their own resources, and may be aggregate tasks themselves. An aggregate task need not require its own resource. The aggregate task structure provides a hierarchical organization of resource availability that may be used to represent a wide variety of interesting and complex scheduling problems.

For our implementation of the AGSS domain, three product tasks are defined as aggregate tasks that provide work areas, in the form of airport gates, where the execution of a variety of required ground servicing product tasks takes place. Similarly, the product task responsible for transferring baggage between connected flights is represented as an aggregate task that requires the use of a single baggage truck to perform the sequence of product tasks involved in carrying out the transfer process.[4]

The process of representing the various temporal-sequencing constraints involved with aggregate tasks is discussed further in Section 3.5.1.3.

3.4.2.3 Inter-Order Tasks

The mention of aggregate AGSS baggage-transfer task leads directly to the issue of representing inter-order tasks within RCSP domains. As we have mentioned earlier, the ability to represent inter-connection among orders is an important capability for a scheduling system. We define an inter-order task as a single product task that begins within one order and ends within another. It may or may not be an aggregate task, and it may or may not require a resource. From the viewpoint of the scheduler, all inter-order tasks are associated with the orders from which they originate. When inter-order tasks are joined into the task networks of their inter-connected orders, they add to the timing constraints within that order that are already implied by the temporal sequencing of the tasks within the order, and its scheduled ready time and due date. Further constraints may be imposed between inter-connected orders as the result of particular scheduling decisions.

In the AGSS domain, the servicing required by connecting flights includes the separate transfer of both passengers and their baggage between the connected flights. Each of these flights is represented by a separate order, with its own timing constraints. In our implementation of

[4] Figures A.21 and A.22 present the process plan and various specifications (respectively) for the *Arrival* flight service type and its corresponding aggregate task defined in our implementation of the AGSS domain. Figures A.23 and A.24, and A.25 and A.26, provide the same information (respectively) for *Departure* and *Turnaround* flights. Figures A.8 and A.9 present the resource plan and task specifications (respectively) for the aggregate baggage-transfer task.

the AGSS domain, the baggage-transfer task is an aggregate task that requires a single baggage truck. The passenger-transfer task does not require any resources.[5]

A full discussion of these kinds of contextual constraints is provided in Section 3.5.1.6.

3.4.3 Task-Processing Durations

Each individual resource task must be equipped with the means to inform the scheduler of the expected amount of time required for performing the task, given the particular resource being considered, the order and product involved, and the time at which the task is expected to be initiated or completed. In addition, each task must be able to provide the scheduler with an estimate of how much time will be required to perform the task on any resource, given only the product and order involved. These estimates are used by the scheduler in the process of constructing and maintaining the task network for an order. They assist the scheduler when making decisions under conditions of uncertainty, and are intended to provide upper bounds on a task's duration so that the overall duration for a reservation will only decrease as more complete information is obtained. Tasks that do not require the services of a resource are expected to provide stable processing durations to the scheduler at all times.

Setup operations introduce an interesting twist to the class of RCSPs. In many domains, setup operations are not always required for resources, for example, when the previous use of a machine is identical to its succeeding assignment. In such situations, the processing duration of the setup operation, being dependent on the context within which the resource is used, is subject to change as the schedule develops.[6] This situation is a constant concern with mobile resources, where the amount of required travel time required by a resource can change frequently during the course of scheduling.

3.5 Constraints

In this section, we provide a description of the kinds of constraints that may be specified on the resources and tasks involved in the production or servicing of a product.

3.5.1 Hard Constraints

We begin our discussion with descriptions of the various *rigid*, or *non-relaxable* constraints that must be fully and consistently satisfied throughout the scheduling process. These constraints provide initial information that is used by the scheduler to assess the nature of a particular scheduling problem.

[5] Figure A.19 presents the task specification for the passenger-transfer task defined in our implementation of the AGSS domain. Figures A.8 and A.9 present the resource plan and task specifications (respectively) for the baggage-transfer task.

[6] Variable setup operations are indicated by a non-nil value for the `VARIABLE-SETUP-DURATION` flag in the definition of a `SETUP` activity class resource task.

3.5.1.1 Technological Constraints

Technological constraints govern the matching of resources with product tasks, given the particular resource, product and order involved. They are the first in a series of constraints considered in the process of securing a resource reservation for a particular product task.

Technological constraints may consider a variety of factors, such as the physical compatibility between a product and a resource. In the AGSS domain, some airport gates are limited in the types of aircraft they may handle. Abnormally large or small aircraft may therefore find the pool of potential gate resources able to process their flights somewhat limited, thus placing them at a greater disadvantage in finding gates than other flights with normal sized aircraft.

The compatibility between a product and the order to which a task belongs may also constrain the selection of a particular resource reservation for a product task. In the AGSS domain, international flights require specific governmental inspection services to be performed as part of the processing of the flight at a gate. Gates that are not equipped to provide the appropriate inspection services thus may not be used to service the aircraft used for international flights. DSS uses these technological constraints to filter out any incompatible resources from consideration to satisfy a particular service goal for a product task. Resources that are determined to be incompatible with a service goal (upon its creation), are forever ignored in the process of both determining the urgency of the subproblem representing the goal, and attempting to secure a satisfying resource reservation for the goal. Technological constraints are specified in the form of domain-specific methods that specialize on a resource class, product and order.

3.5.1.2 Temporal-Sequencing Constraints

Temporal-sequencing constraints control the order in which the various operations that comprise a job may be executed. These constraints are specified as part of the definition of the process plan template for producing or servicing a product. They are used in the process of initially determining, and periodically updating, the size of the allowance of each product task, by providing the bounds on its earliest starting time and latest finishing time.

The temporal-sequencing constraints themselves do not change during the course of the scheduling process. Instead, the values they control (the earliest starting times and latest finishing times for each subproblem), are changed through the propagation of the impact of timing changes that are imposed by the introduction of resource reservations for neighboring tasks, and by changes in the ready times and due dates for orders.

A serial relationship between two tasks indicates that the first task must be completed before the second task may begin.[7] A parallel relationship between separate groups of tasks indicates that the execution of each group of tasks may be undertaken during the same time period.

Figure 3.3 shows a sequence of serially grouped tasks, and illustrates the way in which the earliest starting times and latest finishing times for these tasks are determined. Figure 3.4 provides the same information for parallel task groupings.

In Figure 3.3, moving from left to right at the top, a critical path analysis takes the earliest starting time for TASK 1 and increments it by TASK 1's expected processing duration to produce

[7] Note that this constraint applies only to the execution of the actual product tasks. The execution of the resource tasks required to perform consecutive product tasks may legally overlap (except the `SERVICE-PRODUCT` resource tasks, and assuming that each product task uses a different resource).

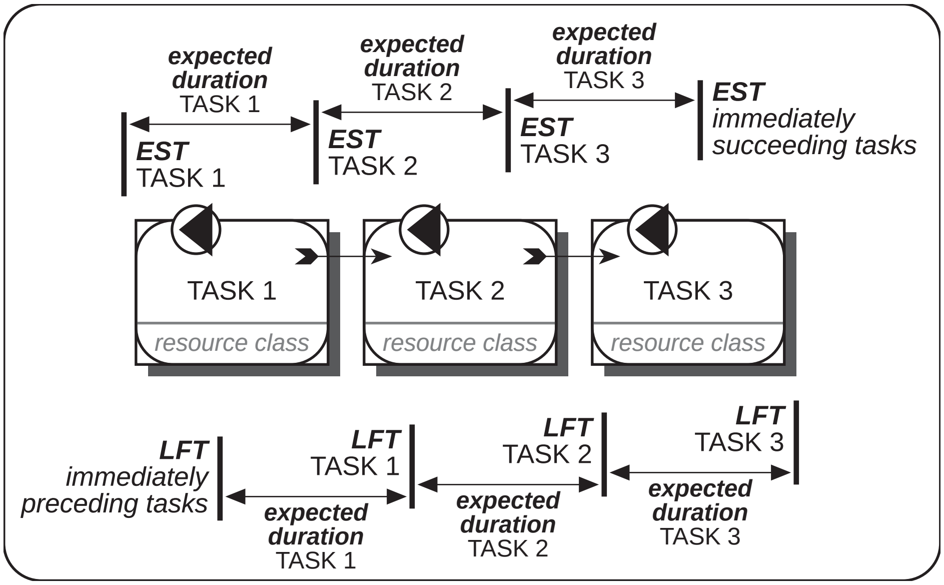


Figure 3.3. Serial Temporal Sequencing in DSS.

the EST for TASK 2, and so on. A corresponding movement from right to left at the bottom produces the LFTs for each task. Note that the allowances created for each task by their ESTs and LFTs overlap with each other, thereby providing maximum flexibility for the scheduler to use in securing resource reservations. Remember also that if a resource is not required by a task, the processing duration for that task can be used in these calculations, and the size of the allowance for such a task will be equal to that duration.[8]

In Figure 3.4, the EST for TASK 1 and TASK 3 are identical, and the EST passed on to any task(s) immediately following the split is determined by the maximum of the ESTs returned at the end of each parallel branch (from TASK 2 in this case). The same goes for the LFT calculation from the other direction. The LFT passed on to any task(s) immediately preceding the split is determined by the minimum of the LFTs returned at the beginning of each parallel branch (from TASK 1 in this case). The effect in both cases is to allocate extra slack time to TASK 3 owing to its requirement of simultaneous operation with TASK 1 and TASK 2.[9]

3.5.1.3 Task-Aggregation Constraints

A task-aggregation constraint defines the hierarchical grouping relationship (introduced in Section 3.4.2.2) that may exist between a parent task and a collection of child tasks, to ensure that the execution of the parent (aggregate) task must completely encompass the execution times for the child tasks. In these relationships, the child subtasks may not begin executing

[8] Examples of serial temporal-sequencing constraint specifications can be found in Figures A.31, A.33, and A.35, which present the process plan definitions for the three different service types used in the TCP application system described in Section A.2.

[9] Examples of parallel temporal-sequencing constraint specifications can be found in Figures A.22, A.24, and A.26, which present the aggregate task definitions corresponding to the three flight types used in our implementation of the AGSS domain.

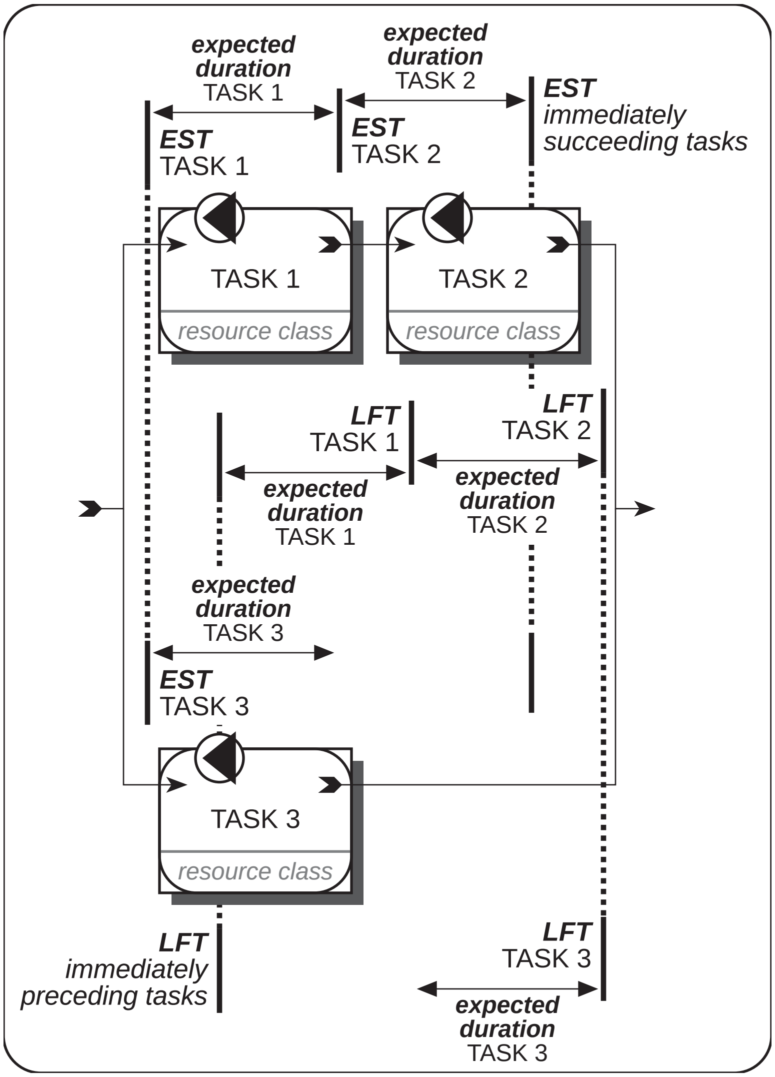


Figure 3.4. Parallel Temporal Sequencing in DSS.

until the parent itself has begun executing, and the subtasks must have finished executing by the time the parent finishes.

Figure 3.5 shows how aggregate tasks and their child tasks are grouped, and how their earliest starting times and latest finishing times are determined. The process of determining ESTs and LFTs begins at the level of the child subtasks. After determining the EST for the initial child subtask(s) and the LFT for the final child subtask(s), these times are passed up to the parent task and used to set its EST and LFT. It is important to note here that the expected processing duration for an aggregate task is based on the EST and LFT it inherits from its child subtask(s). Aggregate tasks therefore do not need the domain-supplied methods that are ordinarily required to provide processing duration information for product tasks.[10]

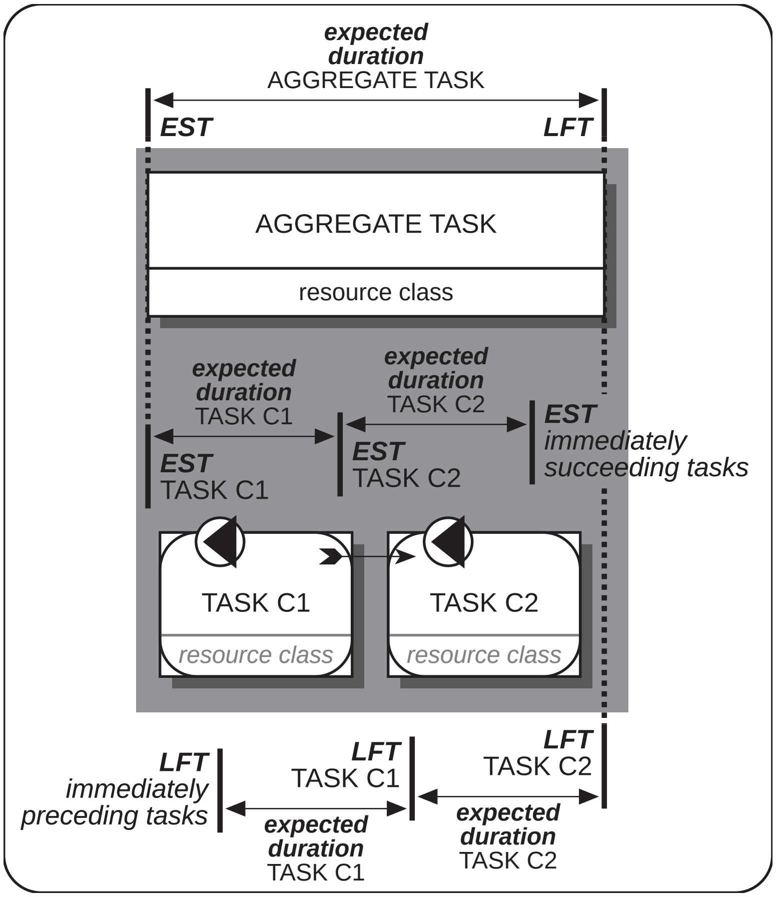

expected duration AGGREGATE TASK
EST
LFT
AGGREGATE TASK
resource class
expected duration TASK C1
expected duration TASK C2
EST TASK C1
EST TASK C2
EST immediately succeeding tasks
TASK C1
resource class
TASK C2
resource class
LFT immediately preceding tasks
LFT TASK C1
LFT TASK C2
expected duration TASK C1
expected duration TASK C2

Figure 3.5. Task Aggregation in DSS.

3.5.1.4 Time-Bounds Constraints

Temporal-sequencing constraints provide the means for the scheduler to determine the earliest starting times and latest finishing times for the tasks that comprise a job, based on the

[10] The reader is again referred to Figures A.22, A.24, and A.26, for examples of the task-aggregation constraint specifications used in our implementation of the AGSS domain.

current ready time and due date for the order, and the expected processing duration for each task. There may be, however, additional constraints that serve to further refine these time bounds. Earliest starting time, latest starting time (LST), earliest finishing time (EFT), and latest finishing time constraints are defined in terms of time intervals from either the ready time or due date of a task's own order, or one of its inter-connected orders.

In the AGSS domain, for example, the task representing the process of enplaning the departing passengers may not begin any earlier than 15 minutes prior to the aircraft's departure time (due date). The EST constraint that enforces this requirement ensures that the enplaning task will always begin within 15 minutes of the aircraft's departure time, and no earlier.

An interesting example from the AGSS domain involving inter-order aggregate tasks serves to demonstrate the importance of handling inter-order time-bound constraints. The baggage-transfer process is represented as an inter-order aggregate task, with child subtasks responsible for unloading all of the baggage to be transferred to a particular departing flight, transferring it, and finally loading it onto the departing flight aircraft. It is therefore necessary that the unloading of all transfer baggage be completed while the originating flight aircraft is still docked at its assigned gate, and that the loading of all transfer baggage onto a departing flight aircraft begin no earlier than a fixed time after that aircraft has been docked at its own assigned gate. These constraints are specified using LFT and EST constraints.

The LFT for each baggage-transfer unloading task (one for each connected flight) are constrained by the due date for the originating flight aircraft, while the earliest starting times for each of the baggage-transfer loading tasks are constrained by the ready time for the (transfer) destination flight aircraft.

These constraints help to refine the current allowance for a task whenever the ready time or due date for its order, or any relevant inter-connected orders, is modified. Note, however, that the time bounds for a task may not be altered to the point of violating the limits already established by the temporal-sequencing constraints. That is, these new constraints may allow a task to start later than previously allowed, but not any earlier. Similarly, they may force a task to be finished earlier than previously allowed, but not any later. LST and LFT constraints lower the upper limit on the allowance for a task, while EFT and EST constraints raise the lower limit.

3.5.1.5 Task-Processing-Duration Constraints

The total amount of time required to perform a particular resource task is determined by a number of constraints, including the actual task to be performed, the resource or type of resource being considered to perform the task, the product or type of product on which the task will be performed, and the order or type of order to which the task belongs. Task-processing-duration constraints are implemented by methods defined for each task that specialize on the resource and the product involved, and determine the expected amount of time required to perform any individual task.

The major factor in the calculation of task durations for `SERVICE` activity class resource tasks is the nature of the task itself, although it is rarely the only factor involved. Most often, the processing duration of a particular task is further affected by the type of product being serviced or produced. For example, the size of the product may have a significant impact on the processing duration. Servicing tasks performed on larger products generally take longer to perform. The type of resource performing the task may also affect the processing duration, because different kinds of resources may perform the same task in different ways, and therefore at different speeds. Finally, the type of the order to which a task belongs may also affect the processing duration. Some orders require more from a particular task than do other orders, thereby increasing the task's duration in that particular situation.

For SETUP and RESET activity class resource tasks, the durations are also affected by the resource, product and order involved. In the case of SETUP tasks, however, the duration may be further constrained by the previous type of usage for the resource, in that some sequence of actions may have to be performed to shift the resource from its previous type of usage and prepare for its next assignment. Such setup operations must be specifically declared as part of a task's definition.

The EN ROUTE activity class resource tasks for mobile resources are generally constrained by the type of resource doing the traveling, and the amount of distance to be traveled. The amount of time required for traveling depends on the exact current location of the resource, its exact required destination, and the speed with which it is expected to be able to get from the former to the latter.

To help determine the expected travel time durations for EN ROUTE activity class resource tasks, DSS provides two kinds of constraints that specify the initial and final locations to be inhabited by a mobile resource in the process of performing a particular servicing task. These constraints are called BEGINNING-LOCATION-CONSTRAINTs and ENDING-LOCATION-CONSTRAINTs (BLCs and ELCs, respectively). They record (respectively), the initial destination required for a mobile resource performing a particular task sequence, and its final location upon the completion of its duties. BLCs and ELCs may vary in the degree to which they specify exact locations within the factory environment. A location may be specified explicitly, such as a particular stationary resource or shop location that provides a work area for other resources, or indirectly, by means of a function of a particular object, such as the closest location to a particular stationary resource or shop location.

These constraints are imposed by the first and last resource tasks in a sequence of tasks performed by a mobile resource. They are considered in the process of securing a mobile resource to perform the task sequence, and also in the case where the previous ending location of a mobile resource is changed, in which case DSS must ensure that there is still enough time for the resource to get from its new ending location to the initial destination of its succeeding reservation. Similarly, any potential change in the beginning location of a mobile resource, often brought about by a change in work areas, must also ensure that there is still enough time for the resource to get from the final destination of its preceding reservation to the new beginning location. BLCs and ELCs are defined as part of the task-processing-duration calculation methods.

The SETUP, SERVICE and RESET activity class resource task-processing durations vary over the range of products, resources and types of service, but remain constant over time. Similarly, the traveling time duration for EN ROUTE tasks varies over the range of potential resources. Finally, mobile resource traveling speed varies over resource class, but remains constant over time.

The modification of previously determined processing durations for EN ROUTE activity class resource tasks is achieved through the resource reservation refinement mechanism discussed in Section 4.2.6.

3.5.1.6 Contextual Constraints

Contextual constraints are hard constraints that are imposed on product task subproblems as the result of previous scheduling decisions. They primarily serve to further limit the options available for satisfying future subproblems, by forcing future decisions to conform to the restrictions introduced by decisions made earlier in the scheduling process. DSS identifies two classes of contextual constraints, one for ensuring compatibility among the various types of equipment used to service a job, and the other for ensuring the proper allocation of travel time to mobile resources moving through the factory environment in the course of reporting to their designated locations and performing their assigned servicing tasks. These constraints are only in effect as long as the previous scheduling decisions that introduced them are maintained. We now discuss these two classes of contextual constraints in further detail.

3.5.1.6.1 Equipment-Compatibility Constraints

Equipment-compatibility constraints are technological constraints that are introduced whenever the selection of a particular resource for performing a particular servicing task serves to constrain the choice of potential resources for performing a related servicing task. The choice of a specific resource or class of resource to perform a particular task often requires that a certain other resource or class of resource be used to perform a related task.

For example, in the AGSS domain, the refueling operation may be performed by either a fuel truck or a pump truck, depending on the kind of refueling equipment provided by the particular gate at which an aircraft is to be serviced. Fuel trucks have their own mobile fuel tanks that are refilled at a central fuel tank at the start of each refueling operation. Pump trucks, on the other hand, work in conjunction with an underground fuel tank that may be provided by a gate. The selection of a pump truck to perform the refueling operation thus introduces a contextual equipment-compatibility constraint on the subproblem responsible for securing the gate, thereby forcing it to consider only those gates that are equipped with underground fuel storage tanks. If a fuel truck is used instead, then there is no constraint imposed on the gate-selection process. Any gate may be used, assuming it satisfies all other constraints. If a gate is secured before the refueling subproblem has been solved, then the refueling subproblem is only constrained if the gate is not equipped with its own underground fuel storage system, at which point a pump truck may no longer be used to perform the refueling operation.

The introduction of a contextual equipment-compatibility constraint as the result of solving a particular scheduling subproblem requires DSS to update the urgency of any affected and still unsatisfied subproblems whose ability to secure a resource reservation has just been further limited by the new constraint. Such a restriction may further decrease the number of resources or kinds of resources that may be used to satisfy the constrained subproblem, thus increasing its urgency.

3.5.1.6.2 Movement Constraints

As we have alluded to earlier, the inclusion of mobile resources into the scheduling problem requires that an important kind of contextual constraint regarding the movement of resources throughout the factory environment be incorporated into the scheduling process. In this case, the notion of equipment compatibility is extended to cover the problem of limiting the required travel distances among the pool of candidate resources considered for servicing a product.

The speed at which a mobile resource is able to travel acts as a contextual constraint on any related subproblems providing work areas for that mobile resource. This type of constraint acts to limit the choice of the possible stationary resources that may be selected to provide work areas, based on how much time has been allocated to the mobile resources to report to their designated work areas. The amount of travel time allocated to a mobile resource to get from one location to another constrains both of the subproblems providing (or determining) those locations. If the reservation of a stationary resource responsible for one of those locations has not been made before any of the mobile resources that depend on that stationary resource have been secured, then the set of stationary resources that may be used to satisfy the stationary resource subproblem is limited to those resources that are within the travel distances allowed by the travel times already allocated to the previously secured mobile resources. Such a situation happens frequently when a strict ordering is not imposed on the process of satisfying the scheduling subproblems that forces all necessary stationary resources to be secured before any of the mobile resource subproblems are solved. To allow complete flexibility in the scheduling decision-making process, DSS is capable of satisfying these contextual-movement constraints so that the order in which the scheduling subproblems are solved is not dependent on whether a required resource is mobile or stationary.

The issue of flexibility with regard to contextual-movement constraints also arises in the process of determining how much time to allocate to a mobile resource reservation to allow for travel between as yet undetermined locations. The amount of time allocated for this movement depends on the present state of problem solving, specifically how much information the scheduler has about a mobile resource's present location, and the location to which it must report. Given this information, the scheduler has to decide how much time should be allocated for the required movement. Upon making this decision, the scheduler introduces a contextual-movement constraint on the subproblems responsible for the resource's previous and required locations. The decisions about each affected subproblem must now be made in such a way as to satisfy the new contextual-movement constraint imposed by the specific allocation of time for traveling between the two undetermined locations.

If anything less than an appropriate maximum amount of time is allocated for travel between undetermined locations, then the contextual-movement constraint becomes restrictive, making the process of securing later movement-constrained subproblems more difficult. To maintain flexibility within the developing schedule for use in handling future problems and avoiding difficulties related to over-constrained subproblems, DSS operates under a policy of allocating the maximal amount of time necessary to get from one location to another when either of those locations has yet to be determined. In this way, flexibility is maintained so that the process of securing the resources that provide those undetermined locations is not so tightly constrained as to severely limit the set of possibly satisfying resources, and create the need for substantially more computational effort on the part of the scheduler.

The domain provides either exact or maximal estimates of the amount of time required for any particular mobile resource movement. A maximal estimate provides enough time for the resource to get from any location where it may reside, to any location to which it may have to report. The travel time duration is determined using the values specified by the relevant BLCs and ELCs for each potential mobile resource. The scheduler may therefore make travel time allocations that do not introduce further constraints on the overall scheduling problem, and may be refined whenever more information about the context within which a mobile resource will be used becomes known.

DSS provides three types of contextual-movement constraints for constraining the resource selection process. Each of these forms is described below:

- **Anchored-Distance Constraint**

  An `ANCHORED-DISTANCE-CONSTRAINT` is used to constrain the selection of a stationary resource to those resources within range of a particular location specified by the reservation for a previously secured mobile resource. For example, suppose that a mobile resource must first report to a designated loading area within the factory before reporting to an assigned work area to perform its required product servicing task. This travel activity becomes part of the resource plan for the task, and thereby limits the set of possible work areas to those that may be reached by the mobile resource when traveling from the loading area, within the amount of time previously allocated for that travel activity.

  An extension of this constraint allows for indirect specification of the anchored location, in the case where the exact location from which the mobile resource will be required to travel is dependent on the location of the selected stationary resource. Taking our previous example further, consider the case where a factory is equipped with several loading areas, and the mobile resources are required to perform their initial loading activities at the loading area associated with the selected work area. In this case, the actual loading area to be used will not be determined until the stationary resource providing the work area is selected, so the constraint must refer to a location that is specifically determined for each candidate stationary resource being considered. The travel time previously allocated to the travel activity for the mobile resource thus specifies how much time is available for getting the mobile resource between the work area and its appropriate loading area.

  It is important to note that these contextual-movement constraints need not be limited to situations involving the use of mobile resources. Distance constraints may be imposed by non-resource-requiring tasks, with the necessary ranges being determined according to specific flow rates instead of resource class traveling speeds.

- **Inter-Order-Distance Constraints**

  The `INTER-ORDER-DISTANCE-CONSTRAINT` is used to constrain the selection of stationary resources for inter-connected orders. Inter-connected orders frequently require that various materials be transported between them using mobile resources. As a result, the selection of the stationary resources involved with the servicing of the inter-connected orders may be constrained by the amounts of time allocated to the travel activities involved in the various transportation tasks.

  In these situations, the selection of a stationary resource for either order constrains the selection of other stationary resources within the immediate order and any inter-connected orders.

  It is again important to note that the tasks through which these inter-order-distance constraints are imposed need not involve the use of a mobile resource. In our implementation of the AGSS domain, any two connecting flights are linked by both a baggage-transfer task, as well as a passenger-transfer task. The former uses the traveling speed of a baggage truck to determine the maximum allowable distance between the gates used to service the two connecting flight aircraft. The latter uses an estimate of the standard walking speed of the transferring passengers.

- **Initial-Travel-Distance Constraints**

  The `INITIAL-TRAVEL-DISTANCE-CONSTRAINT` controls the choice of stationary resources to ensure that enough time is provided for any involved mobile resources to get from their previous locations, as specified by the ending locations of any immediately preceding reservations, to wherever they will be needed.

  For example, consider a supply truck that must report to a series of work areas (one work area per reservation). The selection of each work area must allow the truck to get from the immediately preceding work area to the work area being considered, and from the work area being considered to the immediately succeeding work area.

  The process of selecting a stationary work area resource is thus constrained by all of the reservations for the mobile resources that must use the work area. These mobile resources must be able to get to the work area from where they were previously used, and they must also be able to get to their next assignments.

3.5.2 Soft Constraints

The previously described constraints are all hard constraints that may not be relaxed during the scheduling decision-making process, unless, in the case of contextual constraints, the reservations responsible for their introduction are themselves canceled. This leaves a lot of constraints for the scheduler to satisfy, with little apparent room for flexibility. In even slightly constrained situations, the scheduler must be able to relax some constraints in order to construct a feasible schedule. The constraints that DSS is able to relax are the ready times and, primarily, the due dates of the orders it receives.

Note that ready times may not be moved any earlier—they may only be moved later. The assumption is that orders may be released only when they are ready, and no earlier, and the scheduler has no control on when orders are ready. Due dates, on the other hand, may be moved either forward or backward, depending on the process plan for the order. In fact, in many factory-scheduling domains, it is always desirable to move due dates earlier if the schedule allows. DSS uses such an approach if the process plan for an order requests that its schedule be compacted as much as possible. Otherwise, the due date will only be pushed later as conditions (generally relating to high levels of resource contention) warrant.

Due dates are important constraints that must be met to satisfy the demands of the orders. For lower priority orders, missing the due date is a minor infraction that may result in a slight increase in overall penalties. For more important orders, however, a delay may have a major impact on the quality of the final schedule, and result in significantly higher penalties. The penalties for such delays vary among order types, order priorities, and scheduling domains.

In the long run, the meeting of due dates for all orders is a primary goal for any scheduling system. Regardless, real world situations frequently present scheduling systems with no other options but to sacrifice on-time production for the sake of schedule feasibility. DSS is no different. If a reservation may not be secured, by any means, within the allowance provided to a product task, then the due date for the job containing the task will be sacrificed to find a reservation at the next best time. In situations involving inter-connected orders, the shifting of an inter-order task may force the ready time for an inter-connected order to be moved downstream, thus forcing the order to wait past its desired ready time before it may be released to the factory.

Whenever a decision involving the introduction of lateness or tardiness into an order's developing schedule is made, the penalty to be incurred, weighted according to the specific priority of the order, is always considered, in an attempt to minimize the cost of all such relaxation.

CHAPTER 4

THE DSS SCHEDULING PROCESS

The Dynamic Scheduling System is an event-driven, operation-based scheduling system. At the most basic level, DSS works by solving a series of localized and independent scheduling subproblems, each requiring either the securing of a resource to perform some task, or the refinement of an existing reservation. The intermittent occurrence of certain real-world events triggers other scheduling activities, such as the instantiation of new task networks consisting of interrelated subproblems in response to order receptions, and the cancellation of existing resource reservations in response to the notification of resource failures.

The main control mechanism of DSS comprises an agenda-based process that selects and executes the top-rated subproblem on the agenda at the beginning of each problem-solving cycle. During each cycle, new subproblems may be instantiated and added to the agenda, and existing subproblems may be modified, and their positions on the agenda updated, as the result of internal actions taken, or external events having occurred during the cycle. DSS is finished with the overall scheduling process when the agenda is finally emptied. At this point it will either have produced a feasible schedule for the set of orders it received, or indicated that no feasible schedule could be produced under the current environmental conditions.

As discussed earlier, a domain description supplies DSS with information about the types of orders, products, resources, tasks, constraints, and process plans that comprise a particular RCSP. Prior to the initiation of the scheduling process, DSS is provided with the description of a particular layout indicating all of the available resources and their initial locations within the environment. The complete set of orders to be scheduled, also provided by the domain, may be presented to DSS either periodically or in a single batch, or using a combination of the two, as specified by the user.

Before discussing the specifics of our opportunistic, least-commitment scheduling approach as implemented within DSS, we first highlight the important aspects of the implementation that help achieve the contributions of this work.

- **Handling Inter-Connections Among Orders**

  Inter-order tasks serve to connect otherwise independent orders. This introduction of dependency impacts on a number of decisions that involve the affected orders. The mechanisms responsible for securing resource reservations and determining the urgency of the scheduling subproblems are both designed to consider the effects of these inter-order connections on the decision-making process.

- **Explicitly Represented Slack Time**

  The explicit representation of the slack time that is available within the developing order schedules helps to inform DSS in the process of selecting appropriate methods for securing resource reservations. This order slack provides localized areas within which selected reservations for operations may be introduced with the guarantee that other areas of the schedule will not be affected, thus limiting the ramifications of such decisions and reducing the amount of constraint propagation required.

- **Consultation of Multiple Perspectives**

  The job of determining the urgency of the subproblems requiring the attention of the scheduler is handled by a rating mechanism that considers a wide variety of factors in the process of evaluating each specific subproblem. When combined with the highly opportunistic control mechanism, this allows for quick and effective reaction to the changing environment, by providing timely and accurate assessments of subproblem urgency as the environment evolves.

- **Maintaining Flexibility Through Least-Commitment Decision-Making**

  Flexibility within the developing schedule is maintained through the use of a least-commitment decision-making process that attempts to limit the number of constraints introduced by any particular scheduling decision by preserving slack time. The maintenance of flexibility helps to preserve the range of options available in the future for dealing with outstanding, and often highly constrained, subproblems.

- **Fine-Grained Opportunistic Processing**

  The operation-level granularity of the overall scheduling problem combined with an agenda-based control scheme provides the means for implementing a fully opportunistic scheduling approach. The scheduler is not forced to adhere to any previously established strategic scheduling policy, and is instead able to shift its focus of attention as necessary, according to the current state of problem-solving as indicated by the urgency of the unsolved subproblems remaining to be satisfied.

We begin our discussion of the DSS scheduling approach with a description of the blackboard hierarchy used to represent all of the necessary decision-making structures. Our description of the actual scheduling process explains the sequence of activities triggered by each order reception, including the methods for determining subproblem urgency and selecting the potential methods for subproblem solution. We then describe the various resource reservation-securing knowledge sources implemented within DSS. The remaining sections describe the processes of re-securing a reservation (following a cancellation), refining a reservation, and processing a resource failure.

4.1 Implementation Details

DSS is a blackboard system [Erman *et al.*, 1980], implemented in Common Lisp [Steele, 1990] and the Common Lisp Object System (CLOS) [Keene, 1989], and using the Generic Blackboard System (GBB™) [Blackboard Technology Group, 1992].[1]

The blackboard architecture provides an effective means for representing, organizing, and manipulating the various components that constitute a knowledge-based scheduling system. Multi-dimensional structured *blackboard spaces* are used to store the various units manipulated by a problem-solving process. In DSS, these units include the orders that require production or servicing, the resources used to perform the necessary scheduling tasks, and the networks of task nodes that collectively represent the developing schedule for a set of orders. The scheduling

[1] The primary development platform has been the Texas Instruments Explorer II Lisp machine.

knowledge is encoded within *knowledge sources* (KSs) that are triggered either explicitly or through interaction with the units stored on the blackboard spaces. The execution of these KSs contributes to the generation of a schedule by modifying existing units and creating new ones as the result of specific scheduling decisions.

4.1.1 The DSS Blackboard Hierarchy

GBB provides the means for defining blackboard hierarchies made up of nested blackboards and spaces. These hierarchies form trees whose leaves are the spaces on which the various blackboard units are stored. Figure 4.1 provides a view of the entire blackboard hierarchy defined for DSS.

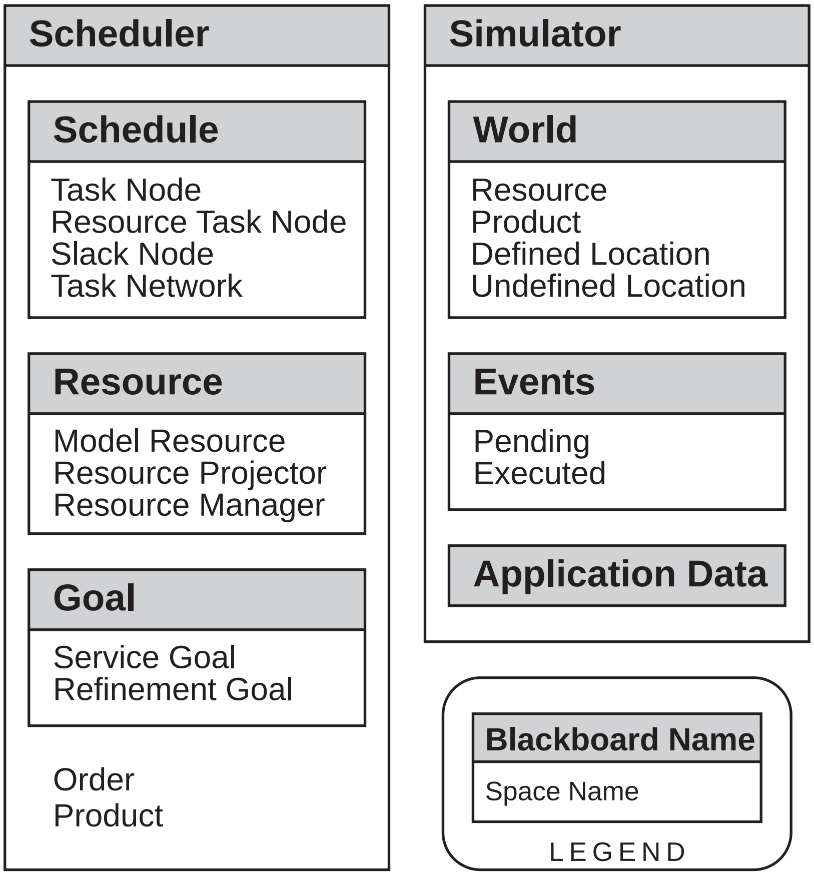


Figure 4.1. DSS Blackboard Hierarchy

The SCHEDULER blackboard is used to store those units directly manipulated by DSS in the course of constructing a schedule for a particular RCSP. It is comprised of three sub-blackboards and two additional spaces. These constituents are described below.

- **Schedule Blackboard**

  The SCHEDULE blackboard is used to maintain the actual developing order and resource schedules for a particular scheduling problem. It contains four spaces for storing task nodes, resource task nodes, slack nodes, and task networks. The TASK NODE space represents the schedule from the perspective of the individual tasks comprising the jobs, while the RESOURCE TASK NODE space represents the schedule from the more detailed perspective of the resources. The TASK NODE and RESOURCE TASK NODE spaces provide dimensions for indexing their respective units by time and type of task. The SLACK NODE space represents the order slack that is incorporated into the developing schedule and used to distinguish existing areas of flexibility residing within the order schedules. The slack nodes stored on this space are indexed by time and the type of the task with which they are associated. The TASK NETWORK space stores the task networks that group together the individual task-nodes for each order and maintain their current starting and finishing times. The task network space provides dimensions for indexing task network units by time, and feasibility (to indicate order schedule completion).

- **Resource Blackboard**

  The RESOURCE blackboard is used to store the various resource representations that are used by DSS to construct a schedule. It contains three spaces. The resource units that correspond to the actual resources provided in a layout are stored on the MODEL RESOURCE space. These *model-resource* units are secured and released by DSS throughout the scheduling process. They are indexed by transitional time and state pairs that represent their expected behavior at any point in time. DSS model-resource units are similar to the *resource models* used in ACS.1 [Pease, 1978], a knowledge-based planning and scheduling system designed to assist in the development and execution of operational plans. Additionally, the time and state pairs used to represent the scheduled activities for model resources in DSS are similar to the *scroll tables* employed by ACS.1.

  Projection is performed by some of the reservation-securing KSs to aid with some of the more complicated scheduling computations. Special copies of the model resources, called *model-resource projector units*, are defined for this purpose. In the case of attempting to preempt or right shift existing resource reservations, projection units for the resources involved are manipulated (instead of the actual model resources) to determine the effects of certain actions without having to specifically execute them. These projector units are stored on the RESOURCE PROJECTOR space, and are indexed identically as the model resources. A projector unit is created for every model-resource unit stored on the MODEL RESOURCE space.

  Finally, the RESOURCE MANAGER space stores the resource manager units that maintain information about the current levels of usage, demand and availability for each resource class, and manage the securing and releasing of the resources selected by the reservation-securing KSs. The resource manager units are indexed by time and status pairs that represent the levels of resource usage, demand and availability at any point in time.

- **Goal Blackboard**

  The GOAL blackboard contains two spaces. The SERVICE GOAL space provides an overview of the current demand for resources by storing the service goals for the subproblems that correspond to the resource-requiring product tasks. The SERVICE GOAL space is heavily structured. Service goals are indexed by time, type of task and type of resource. They are also indexed by satisfaction (indicating whether or not their corresponding subproblems have been solved), and activation (indicating whether DSS may begin attempting their solution). Finally, they may be indexed by the histories of their ratings and allowances over time. The REFINEMENT GOAL space stores the goals that guide the reservation refinement mechanism. Refinement goals are activated whenever new information is obtained that helps to further establish the context for previous decisions made under uncertain conditions. Their activation leads to the triggering of the reservation refinement process described in Section 4.2.6. Refinement goals are indexed by time, type of task, type of resource, satisfaction and activation.

- **Order Space**

  The ORDER space stores all of the order units received by DSS. Placement of an order onto this space triggers the task network instantiation process described in Section 4.2.1. Order units are indexed by ready time, due date time, number and service type.

- **Product Space**

  The PRODUCT space stores the product units that correspond to the actual products that require servicing. These *model-product* units are indexed by order number and service type.

The SIMULATOR blackboard provides the connection between the (simulated) real world and the internal representation of that world manipulated by DSS in the process of producing a schedule. DSS uses the simulator for two purposes. First, as an event-driven scheduling system, DSS operates by processing a queue of world events. For example, it is notified of incoming orders by the reception and processing of `RECEIVED-ORDER` events that are taken off of a pending queue of world events. All communication between the "real world" and DSS occurs via this queue.

The more intensive use of the SIMULATOR blackboard involves running DSS in a full simulation mode, where the execution of its scheduling decisions may be fully simulated. This mechanism provides the means for further ensuring the feasibility of DSS's scheduling decisions, and introducing unexpected real-world developments, such as resource failures and task execution delays into the scheduling process.[2]

The SIMULATOR component blackboards and their spaces are described below.

- **World Blackboard**

  Whether running in full simulation mode or not, the WORLD blackboard is always used to store the units representing the actual physical resources that are provided by a layout loaded into the DSS environment to be used for producing schedules. These physical resource units are initially placed in two different areas, namely the RESOURCE space and the DEFINED LOCATION space. The RESOURCE space stores all of the physical resource units. The DEFINED LOCATION space is used for keeping track of the exact current location of the resources within the factory environment during the execution of a schedule. The UNDEFINED LOCATION space is used as a temporary storage area for mobile resources that

[2] The ability to introduce and react to task execution delays has not yet been implemented in DSS.

are currently moving within the factory environment such that their exact location may not be determined. The PRODUCT space is used to store the product units representing the actual physical products that require the servicing or production requested by the orders. These physical product units are also moved between the DEFINED LOCATION and UNDEFINED LOCATION spaces as their locations change during the execution of a schedule.

- **Events Blackboard**

  The EVENTS blackboard is used to implement the simulated world event queue. The PENDING space maintains the queue of pending world events, all of which must be processed by DSS in order for it to complete its overall scheduling task. Once events have been taken off of the pending queue and executed, they are placed onto the queue maintained on the EXECUTED space.

- **Application Data**

  The final blackboard hierarchy provided by DSS is rooted by the APPLICATION DATA blackboard, within which a particular scheduling domain may define additional blackboards and spaces for use in implementing a specific dynamic RCSP.

4.1.2 Agenda-Based Control

GBB provides an agenda-based control shell for managing the execution of DSS. Each problem-solving cycle is defined by the execution of a single KS, in the form of a *knowledge source activation* (KSA). An agenda containing all pending KSAs is maintained by the GBB-provided control shell. At the beginning of each problem-solving cycle, the highest rated KSA is removed from the agenda and executed. During the execution of a KSA, all internally triggered events are collected, for processing at the end of the cycle. The processing of these events generally results in the activation of additional KSs, which are then rated and placed onto the agenda. Execution completes when the agenda is emptied.

4.1.3 Event-Based Processing

As mentioned earlier, the DSS scheduling process works by reacting to both internal and external events. Internal events are triggered as the result of blackboard space updates and unit modifications caused by the various scheduling KSs. Their processing is handled by GBB, as part of the agenda-based control mechanism described above. DSS also manages to process some internal events of its own, specifically the re-rating of any service goals, and their stimulated reservation-securing KSAs, that are affected by the actions taken during the execution of another KSA.

External events are generated by the simulator as the result of the scheduling decisions made by DSS. DSS is notified of resource failures through user-inserted resource failure events.[3] At the end of each problem-solving cycle, the queue of pending external events is processed by DSS. Each processed event is moved to the executed event queue following its execution.

[3] The periodic introduction of resource-servicing-completion delays has not yet been implemented in DSS.

4.1.4 Generic Problem-Solving Knowledge Sources

The collection of generic problem-solving KSs defined to perform the required scheduling activities in DSS is displayed in Table 4.1. Included are six reservation-securing KSs (comprised of standard and relaxed versions of three different reservation methods), a meta-level KS for selecting from among these KSs, and a KS for refining existing reservations. The collection is rounded out by a problem-instantiation KS, and a resource-failure-processing KS. Each of the ten KSs is either triggered directly by another KS, or as the result of the modification of some blackboard space or unit instance. Detailed discussions of each of these KSs appear later in this chapter.

4.2 The Scheduling Process

We begin our discussion of the DSS scheduling process with a description of the sequence of scheduling activities that is set into motion following the reception of an order, and continuing through the process of assigning specific resources to tasks. As part of this discussion, we include some portions of DSS execution trace output. Figure 4.2 provides a general view of the scheduling process implemented within DSS, illustrating the standard flows of control employed for scheduling orders and reacting to the events triggered by certain DSS scheduling actions.

Following the reception of an order, DSS instantiates a network of task-level subproblems according to the process plan template for the particular type of service required. This step is described in Section 4.2.1. The subproblem-instantiation process leads to the allocation of slack time and the creation of service goals that correspond to each resource-requiring task. The urgency of a service goal is determined by the multiple-perspective goal-rating mechanism presented in Section 4.2.2. Finally, the reservation-selection process takes responsibility for finding resource reservations to satisfy each service goal. The KS that manages this process for each service goal is described in Section 4.2.3. The six reservation-securing KSs provided in DSS are presented in Section 4.2.4. Many of these basic steps trigger additional activity for the scheduler in maintaining an up-to-date view of the current state of problem-solving. For example, the allocation of slack time to newly instantiated subproblems requires the urgency ratings of related (existing) subproblems to be recalculated. In addition, the actions taken by the various reservation-securing KSs can have a variety of direct and indirect effects on related subproblems, leading to alteration of their allowances, recalculation of their urgency ratings, and re-securing of their reservations.

In each problem-solving cycle, DSS *invokes* the highest rated KSA from the agenda by executing the appropriate KS on the KSA's stimulus data. The various scheduling decisions made during a problem-solving cycle may cause certain KSAs residing on the agenda to have their ratings, and thus their positions on the agenda, adjusted. Two of the most frequent methods of modifying the agenda arise from the cancellation of existing reservations, and the modification of unsatisfied service goal allowances. The actions forced by these events are discussed in Sections 4.2.5 and 4.2.2, respectively.

In addition to the reservation-securing KSs, there is also a *Reservation Refinement* KS that is responsible for taking an existing reservation that was made under uncertain conditions, and re-assessing its requirements whenever newer, relevant information is obtained. The reservation refinement process is discussed in Section 4.2.6. Finally, the response to unexpected resource failures is described in Section 4.2.7.

Table 4.1. Generic Problem-Solving Knowledge Sources Defined in DSS.

Instantiate Task Network	Responsible for instantiating all necessary problem-solving entities for a particular order (or set of inter-connected orders). Initiates the scheduling process for each order received.
Select Reservation-Securing Method	Controls the sequence in which reservation-securing KSs are triggered to solve a particular subproblem.
Assignment	Attempts to produce a resource reservation within the allowance of a subproblem.
Preemption	Attempts to produce a resource reservation within the allowance of a subproblem, by preempting existing reservations as necessary.
Right Shift	Attempts to produce a resource reservation, either exceeding, or outside of, the allowance of a subproblem, by waiting for the next available resource, delaying any affected downstream reservations, and extending any affected parent reservations, as necessary.
Relaxed Assignment	Attempts to perform a standard *Assignment* for an over-constrained subproblem, relaxing offending reservations as necessary.
Relaxed Preemption	Attempts to perform a standard *Preemption* for an over-constrained subproblem, relaxing offending reservations as necessary.
Relaxed Right Shift	Attempts to perform a standard *Right Shift* for an over-constrained subproblem, either relaxing, or *canceling*, offending reservations as necessary. Will not fail to secure a reservation (if a resource of the proper class exists).
Reservation Refinement	Performs the refinement of previously uncertain resource reservations following the establishment of additional contextual information.
Process Resource Failure	Determines the impact of a resource failure. Cancels all obviated reservations and reactivates their corresponding subproblems.

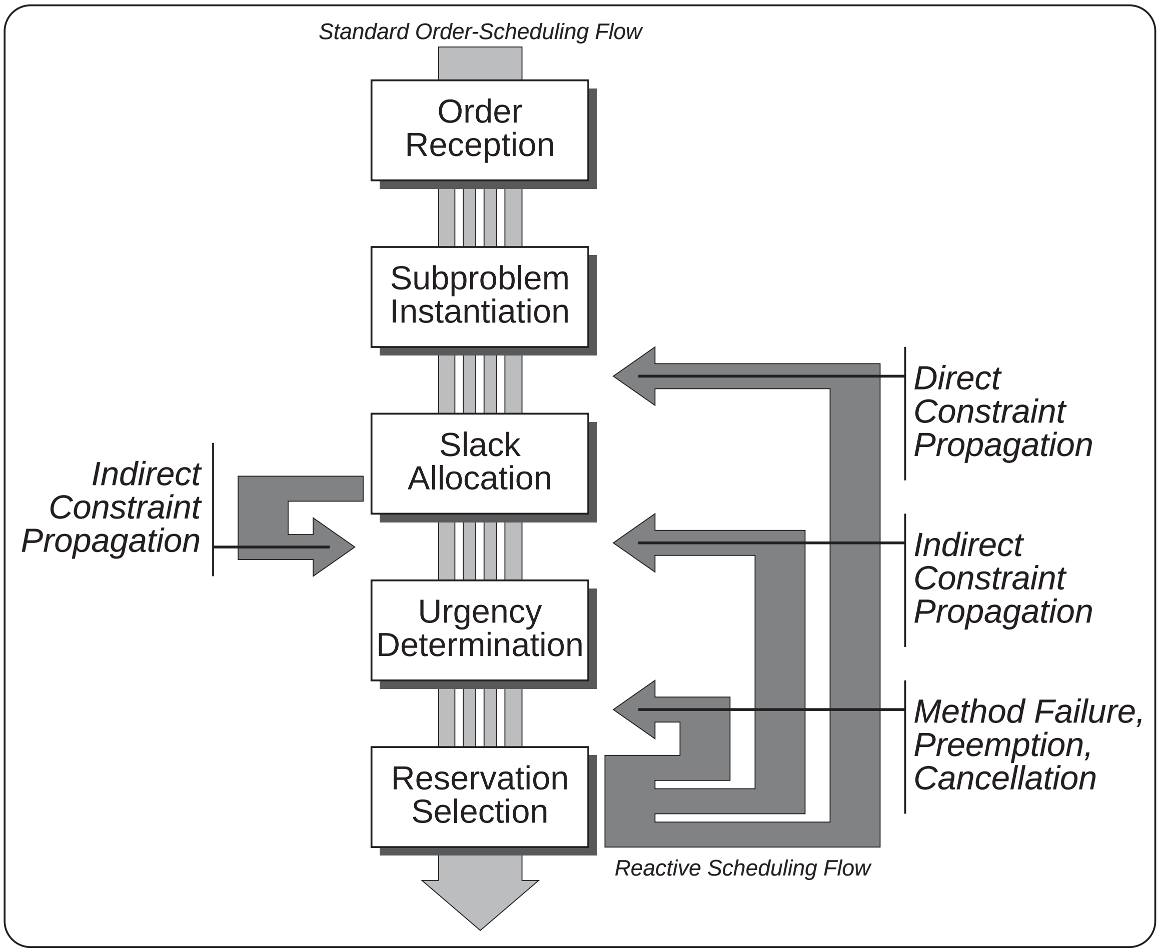

Standard Order-Scheduling Flow
Order Reception
Subproblem Instantiation
Slack Allocation
Urgency Determination
Reservation Selection
Indirect Constraint Propagation
Direct Constraint Propagation
Indirect Constraint Propagation
Method Failure, Preemption, Cancellation
Reactive Scheduling Flow

Figure 4.2. An Overview of the DSS Scheduling Process.

4.2.1 From Order Reception to Subproblem Instantiation

The operation-level granularity of the DSS scheduling approach is achieved through a process of instantiating a collection of individual product task subproblems for each order received by the system. The solution of all of the subproblems for a particular order results in a schedule of resource activity whose execution will complete the production or servicing of the product as requested by the order. The basic sequence of steps involved in the processing of a newly received order and the generation of its corresponding subproblems is engineered by the *Instantiate Task Network* KS. Additional actions are performed as the result of internal events that are automatically triggered by the execution of these steps. We first present an overview of this sequence, and then discuss each step in greater detail.

1. Determine the type of servicing requested by the order and assess the appropriate process plan to implement that service.

2. Instantiate a *task node* for each task specified in the process plan.

3. Determine the earliest starting time and latest finishing time bounds for all of the new task nodes that require resources, and allocate the necessary time for all tasks that do not.

4. Create, initialize, and activate a *service goal* for each resource-requiring task node.

4.2.1.1 Task Network Instantiation

The *Instantiate Task Network* KS is triggered immediately following the reception of a RECEIVED-ORDER event and given the highest possible execution rating, under the assumption that it is preferable for the scheduler to react as quickly as possible to each order that it receives.

To define a particular type of servicing for a product, a *process plan template* is specified, indicating the required product tasks and the sequence in which they must be performed.[4] We use the term *task network* to refer to the instantiated sequence of operations specified by a process plan template. A task network is a directed graph (containing no cycles) of product tasks that are connected by arcs indicating temporal sequencing constraints (see Section 3.5.1.2) and task aggregation constraints (see Section 3.5.1.3). A single task network is instantiated for each order received by the scheduler. It represents the developing process plan for an order, and maintains the expected starting and finishing times for the plan, corresponding to the order's desired ready time and due date.

After determining the type of servicing required by an order, the *Instantiate Task Network* KS instantiates the necessary scheduling units according to the relevant process plan template. It then begins a top down instantiation process that creates all of the necessary *task nodes* to represent the required product tasks. This instantiation process continues down through the required resource tasks for each product task, until all of the necessary task units have been created and linked into the final task network.

Once the task network and all of its task nodes have been fully instantiated, the *Instantiate Task Network* KS must analyze the task network to determine the earliest starting time and latest finishing time for each resource-requiring task node. The allowance for a resource-requiring task defines the total amount of time that is available to perform the task. Any secured resource reservation that is within the bounds of the allowance is guaranteed *not* to introduce any conflicts with the rest of the schedule or incur any delay for its order. The calculation of these time bounds is performed using a Critical Path Analysis [Kelley and Walker, 1959], which involves a forward pass through the network to determine the absolute earliest starting time for each resource-requiring task, and a backward pass to determine the absolute latest finishing time, again, for each resource-requiring task. Figures 4.4 and 4.5 illustrate the results obtained by each of these steps for a typical newly instantiated task network. The task network in presented in these figures is taken from our implementation of the ARM system described in Appendix A.1. Detailed presentations of the ground servicing activities required to service *Turnaround* (and other) flights are provided in Sections A.1.6 and A.1.5.) Figure 4.3 provides a legend for both of these figures.

The forward pass starts with the initial set of non-aggregate task nodes and the ready time for the order, and moves forward through the network, calculating the earliest possible starting time for each task based on the earliest EFT among all immediately preceding tasks, and considering any further constraints on its starting time. The EFT for a task is determined by adding its expected duration to its EST. The second pass starts with the final set of non-aggregate task nodes and the due date, and moves backward through the network, calculating the latest possible

[4] The DSS process plan template is similar to the ACS.1 *process model* [Pease, 1978].

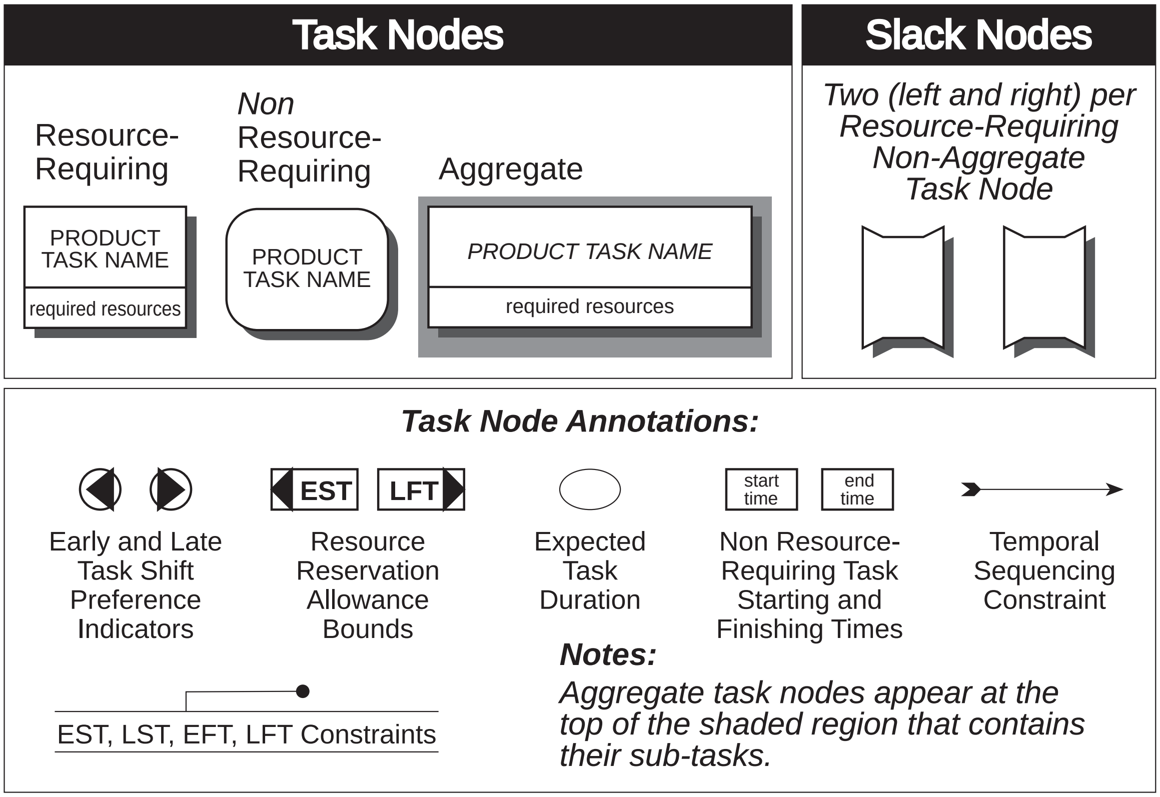

Task Nodes
Resource-Requiring
Non Resource-Requiring
Aggregate
PRODUCT TASK NAME
required resources
PRODUCT TASK NAME
PRODUCT TASK NAME
required resources
Slack Nodes
Two (left and right) per Resource-Requiring Non-Aggregate Task Node
Task Node Annotations:
EST
LFT
start time
end time
Early and Late Task Shift Preference Indicators
Resource Reservation Allowance Bounds
Expected Task Duration
Non Resource-Requiring Task Starting and Finishing Times
Temporal Sequencing Constraint
EST, LST, EFT, LFT Constraints
Notes:
Aggregate task nodes appear at the top of the shaded region that contains their sub-tasks.

Figure 4.3. Task Network Component Legend.

finishing time for each task based on the latest LST among all immediately succeeding tasks, and considering any further constraints on its finishing time. The LST for a task is determined by subtracting its expected duration from its LFT. For non-aggregate resource-requiring tasks, the expected task duration is determined by averaging the minimum and maximum expected durations over all of the resource classes that may possibly be used to perform the task. For resource-requiring aggregate tasks, their expected durations are inherited from the time bounds of their outermost children. For non-resource-requiring tasks, the exact expected duration is readily determined. Non-resource-requiring tasks are not provided with slack during this process. Any slack that they might have accrued is passed on to their neighboring tasks, depending on their shift preference.

In Figure 4.4, the ready time for the order is 1 (specified in generic time units, in this case minutes), and the initial non-aggregate task is SHUTDOWN. This task does not have any constraints on its starting time, so its starting time is assigned the value 1, which is in turn inherited by its parent TURNAROUND ACTIVITY aggregate task. The expected duration of the SHUTDOWN task is 4. Additionally, because SHUTDOWN is a non-resource-requiring task with an early shift preference, its finishing time may be assigned the value of 4 at this time as well.[5] The immediately following tasks, namely RESTOCK, SERVICE, UNLOAD BAGGAGE and

[5] Time ranges in DSS are represented as non-overlapping intervals, so that a range of 10 time units starting at time 1 and finishing at time 11 is expressed as [1, 10]. The time range values appearing in Figures 4.4 and 4.5, and the trace segments in this chapter and Appendix B, reflect this feature of the implementation.

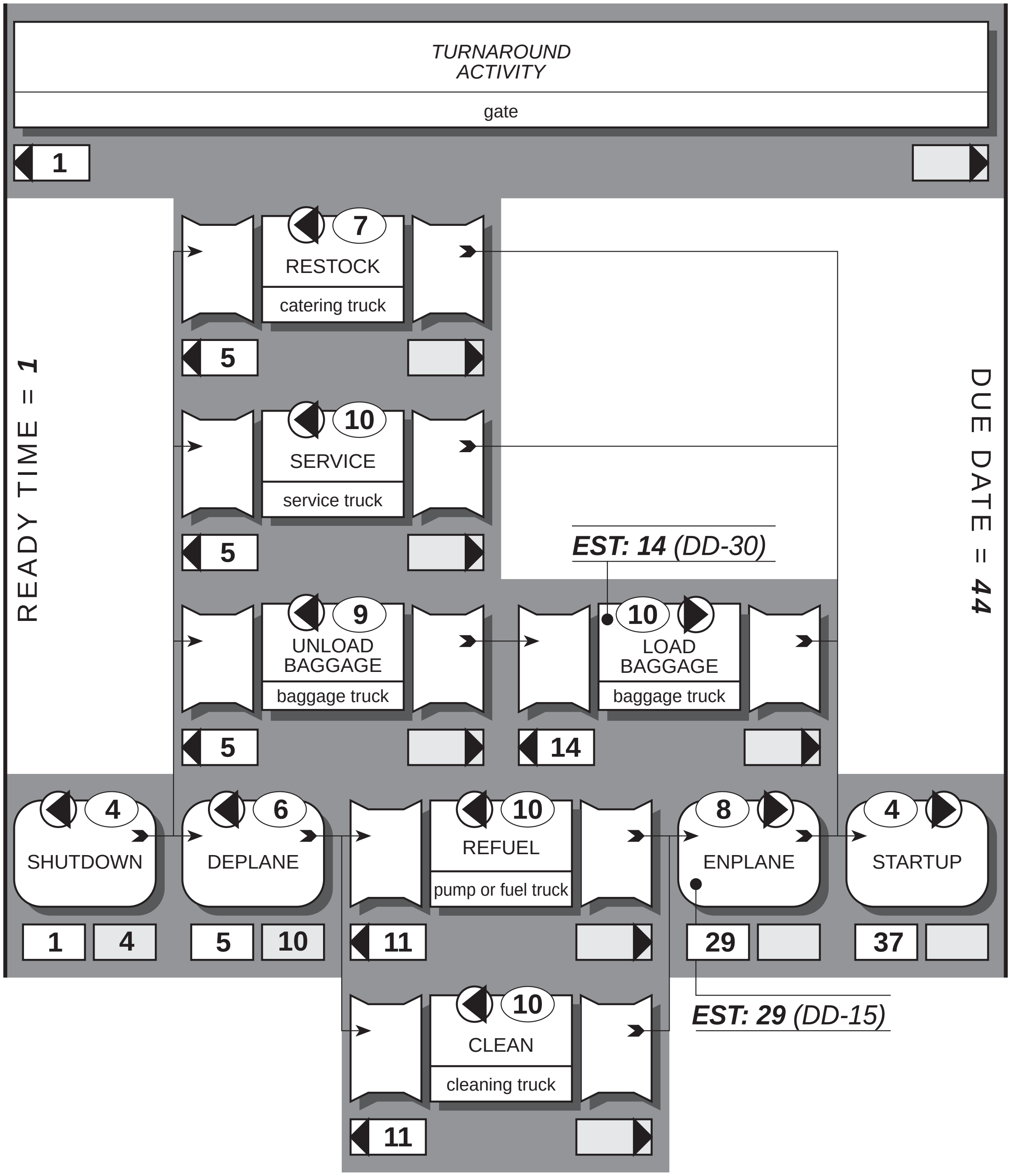


Figure 4.4. Determining ESTs within a Task Network.

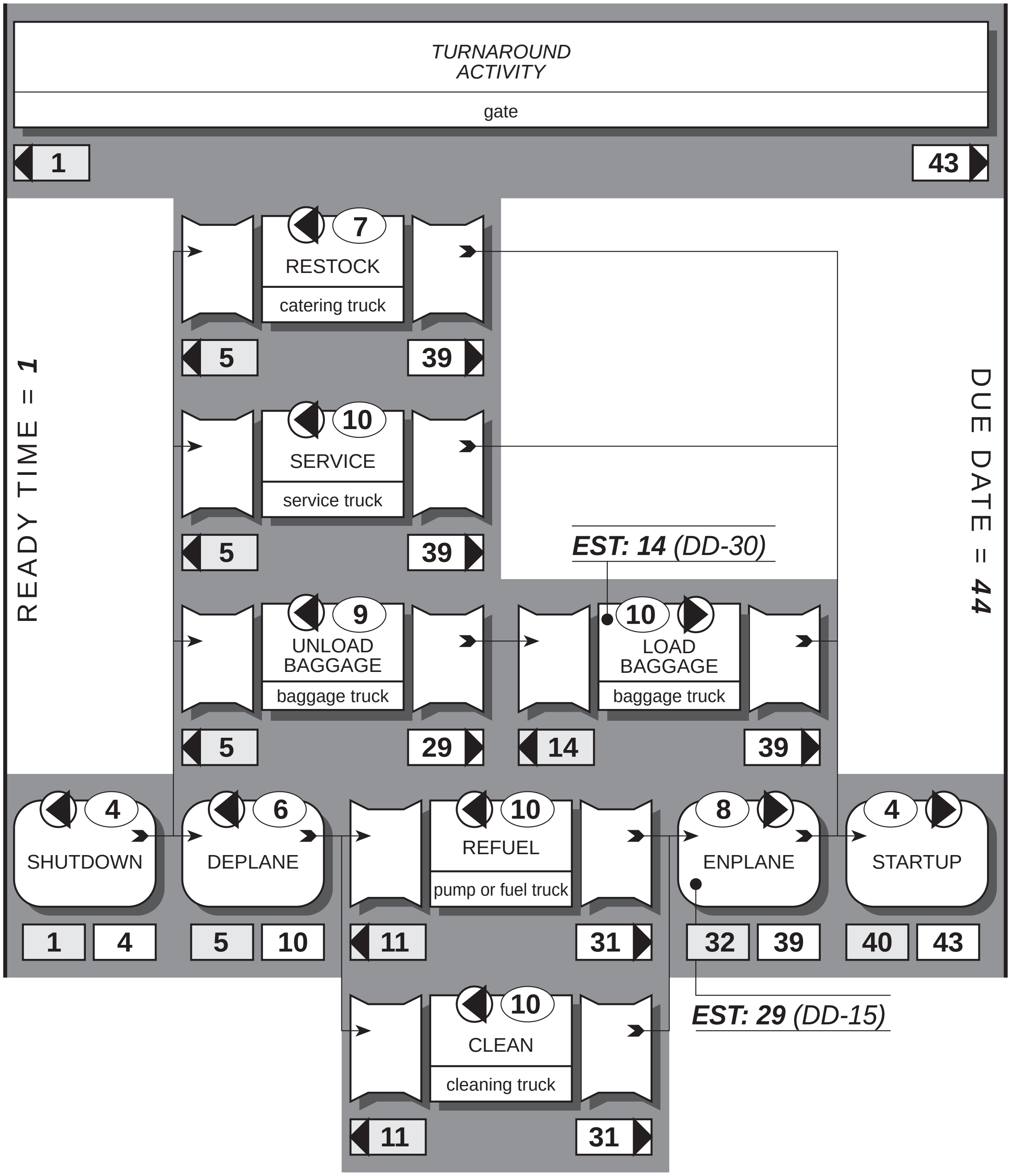


Figure 4.5. Determining LFTs within a Task Network.

DEPLANE, are then all assigned ESTs of 5. Because of the similarity between DEPLANE and SHUTDOWN, the finishing time for DEPLANE is set to 10, which leads to an EST of 11 for the REFUEL and CLEAN tasks that follow it. For the LOAD BAGGAGE task, the EST of 14 that is carried forward from the UNLOAD BAGGAGE task is equal to the EST constraint value of 14 based on the order's due date of 44 (the LOAD BAGGAGE task may not begin any earlier than 30 minutes prior to the departure time). For the ENPLANE task, while the preceding REFUEL and CLEAN tasks suggest an starting time of 21, the EST constraint on ENPLANE (requiring that it not begin any earlier than 15 minutes prior to the departure time), forces a starting time of 29 instead. Finally, the STARTUP task is assigned a starting time of 37. Because the last two non-resource-requiring tasks have a late shift preference, their finishing times are left uninitialized.

In Figure 4.5, the due date for the order is 44, and the final non-aggregate task is STARTUP. This task does not have any constraints on its finishing time, so its finishing time is assigned the value 43, which is in turn inherited by its parent TURNAROUND ACTIVITY aggregate task. The expected duration of the STARTUP task is 4, and because the STARTUP task is a non-resource-requiring task with a late shift preference, its starting time is reset to 40 (from 37), thereby maximizing the amount of slack that will be available to the preceding LOAD BAGGAGE task. The preceding RESTOCK, SERVICE, LOAD BAGGAGE and ENPLANE tasks are then assigned LFTs of 39. The ENPLANE task, also a late shift preference non-resource-requiring task, has its starting time reset to 32 (from 29). The preceding REFUEL and CLEAN tasks are then assigned LFTs of 31. The UNLOAD BAGGAGE task receives a LFT of 29 from LOAD BAGGAGE, and at this point, the allowance time bounds determination process is complete.

4.2.1.2 Service Goal Creation

For non-aggregate resource-requiring tasks, the EST and LFT bounds are used to initialize a pair of *slack nodes* that represent the window of time that is available within an order schedule for securing a resource reservation. The lower boundary time of the left slack node is set to the EST for the task, and the upper boundary time of the right slack node is set to the LFT. For aggregate resource-requiring tasks, slack nodes are not created, because aggregates encompass the slack of their outermost child tasks. In this case, the EST defines the lower bound on the desired resource reservation for the aggregate, while the LFT defines the upper bound.

The operation-level scheduling subproblems addressed by DSS are defined by the pairing of the task node for a resource-requiring product task with a *service goal* that guides the search for a satisfying reservation. The task node provides the description of the subproblem, while the service goal controls the process of triggering the appropriate KSs for finding successful resource reservations for the task node. A resource-requiring task thus becomes a subproblem as the result of the instantiation of both a task node and a service goal. Tasks that do not require resources are not paired with service goals, and thus do not appear as scheduling subproblems, although sufficient time for performing their designated tasks must be included within the final schedule for an order. Throughout the rest of this discussion, the use of the term *subproblem* will refer to a resource-requiring product task seeking a resource reservation within some particular allowance of time.

Each service goal maintains an allowance defined by the EST and LFT bounds determined for its corresponding task or slack nodes, and within which any satisfying resource reservation is desired to be found. The initialization of the boundary times for an aggregate resource-requiring task node or the slack nodes of a non-aggregate resource-requiring task node triggers the creation of a companion service goal. This process is summarized in Figure 4.6. Whenever there is a change in the time bounds for either a resource-requiring aggregate task node (as the result of propagated changes resulting from updates to the timing constraints of its children) or either of a (non-aggregate) resource-requiring task node's slack nodes, the allowance for the corresponding service goal must also be updated to reflect the change.

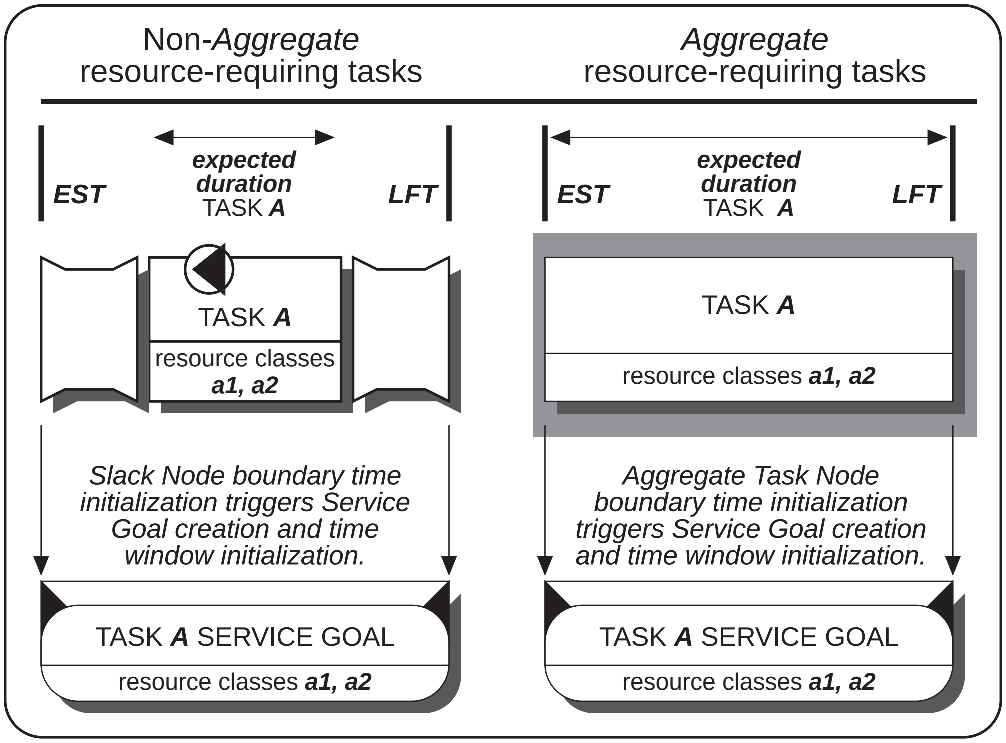

Non-Aggregate resource-requiring tasks
Aggregate resource-requiring tasks
EST
expected duration TASK A
LFT
EST
expected duration TASK A
LFT
TASK A
resource classes a1, a2
TASK A
resource classes a1, a2
Slack Node boundary time initialization triggers Service Goal creation and time window initialization.
Aggregate Task Node boundary time initialization triggers Service Goal creation and time window initialization.
TASK A SERVICE GOAL
resource classes a1, a2
TASK A SERVICE GOAL
resource classes a1, a2

Figure 4.6. Aggregate and Non-Aggregate Service Goal Initialization.

A service goal also records all of the resource classes that may be used to perform the task represented by its corresponding task node, based on the task definitions provided by the domain description. Using its allowance and list of usable resource classes, a service goal engineers the triggering of specific resource reservation-securing KSs to find reservations that satisfy the various requirements of its task node. In the case where more than a single resource class may be used to perform a task, the service goal is responsible for triggering reservation-securing KSs for each potential resource class.

4.2.1.3 Inter-Order Task Instantiation

Inter-connected orders warrant special attention during the instantiation process. Their existence requires an expanded task network structure that links formerly independent task networks through various inter-order tasks. This expanded network structure represents the networks for an entire set of inter-connected orders, and thus contains multiple ready times and due dates, each of which now has a potential impact on the ESTs and LFTs for all of the task nodes. As a result, the *Instantiate Task Network* KS must first instantiate the task network structures for the entire order set before determining the time bounds for any of the task nodes, and these bounds may only be determined after both the forward and backward passes have been performed using the ready times and due dates (respectively) of *each* of the orders in the inter-connected order set. A minor implication of this behavior is that the KS must check when it actually receives the official notification of an order to make sure that it has not already instantiated a task network for the order and begun securing its required resources.

Wherever an inter-order task is to occur within an order, an entry must be included in the process plan template that provides the name of the task, a flag indicating whether the task is originating or completing, and the means for determining the set of orders to which the task may be connected. In accordance with the rules of temporal sequencing specified in a process plan template, originating inter-order tasks may not be started before all of their preceding operations have been completed and their parent (if any) has begun its execution. At the other end, completing inter-order tasks act just like local product tasks, in that any succeeding tasks may not be initiated until the inter-order tasks have been completed.

Figure 4.7 presents a portion of a DSS execution trace showing the reception of an order (an airplane flight, in this case), followed by the triggering, and immediate execution of the *Instantiate Task Network* KS that leads to the creation and activation of the required service goals. Note that this example involves two inter-connected orders, requiring the *Instantiate Task Network* KS to process the two orders during a single invocation. The specific DSS-based application system used to produce these trace segments is described in Section A.1.

In Figure 4.7, DSS is notified of `<[Flight #235/235(Mon)D]-#0>` exactly two hours prior to its expected arrival time.[6] The receipt of this order triggers the precondition function for the *Instantiate Task Network* KS, which checks that a task network for the order does not already exist and then determines a rating for the KSA. An activation of the *Instantiate Task Network* KS is then created and placed on the queue. The requested service type for `<[Flight #235/235(Mon)D]-#0>` is `TURNAROUND`, which entails the deplaning of all arriving passengers and baggage, the execution of a collection of assorted ground servicing tasks, and the final enplaning of all departing passengers and baggage. `<[Flight #235/235(Mon)D]-#0>` and `<[Flight #1447/1447(Mon)D]-#0>` are connected by two shared PASSENGER TRANSFER and BAGGAGE TRANSFER tasks. After the ESTs and LFTs for the required ground servicing tasks of both flights have been determined, all of the necessary service goals are created. The execution trace in Figure 4.7 shows the time at which each service goal is created, its potential resource classes and initial allowance, and the order to which each belongs.

Figure 4.8 presents a diagram showing a newly instantiated task network for an *Arrival* flight, following the execution of the *Instantiate Task Network* KS. `<[Flight #529/(Mon)D]-#0>` (the *Arrival* flight) is connected to `<[Flight #/171(Mon)D]-#0>` (a *Departure* flight), through two inter-order tasks (BAGGAGE TRANSFER and PASSENGER TRANSFER). It requires the use of a stationary GATE resource and three mobile resources. The ARRIVAL ACTIVITY task is the main (aggregate) task, requiring a 30-minute GATE reservation. The BAGGAGE TRANSFER task is an aggregate requiring a 33-minute BAGGAGE TRUCK reservation. Reservations for the remaining CLEAN and UNLOAD BAGGAGE tasks are expected to require an average of 5 and 6 minutes, respectively. The shaded blocks indicate time that

[6] The two hour advance notification period is a user-defined interval.

⋮

```
Order Received --------> [Mon 1:36pm] <[Flight #235/235(Mon)D]-#0>
                         (Release Time: [Mon 3:36pm] - Due Date: [Mon 4:20pm])
                          TURNAROUND <Application Priority: 1>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
------------------------------- KSA 2 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
Created Service Goal --> [Mon 1:36pm] [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#1 (GATE)
                         ([Mon 3:36pm] to [Mon 4:19pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [SERVICE]-SERVICE-GOAL-#2 (SERVICE-TRUCK)
                         ([Mon 3:40pm] to [Mon 4:15pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [RESTOCK]-SERVICE-GOAL-#3 (CATERING-TRUCK)
                         ([Mon 3:40pm] to [Mon 4:15pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#4 (BAGGAGE-TRUCK)
                         ([Mon 3:49pm] to [Mon 4:15pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [UNLOAD-BAGGAGE]-SERVICE-GOAL-#5 (BAGGAGE-TRUCK)
                         ([Mon 3:40pm] to [Mon 4:05pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [CLEAN]-SERVICE-GOAL-#6 (CLEANING-TRUCK)
                         ([Mon 3:46pm] to [Mon 4:07pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [REFUEL]-SERVICE-GOAL-#7 (FUEL-TRUCK PUMP-TRUCK)
                         ([Mon 3:46pm] to [Mon 4:07pm]) <[Flight #235/235(Mon)D]-#0>.
Connected Order -------> [Mon 1:36pm] Scheduling for <[Flight #1447/1447(Mon)D]-#0>
                         (owing to its connection with <[Flight #235/235(Mon)D]-#0>).
Created Service Goal --> [Mon 1:36pm] [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#8 (GATE)
                         ([Mon 4:37pm] to [Mon 5:19pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [SERVICE]-SERVICE-GOAL-#9 (SERVICE-TRUCK)
                         ([Mon 4:40pm] to [Mon 5:16pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [RESTOCK]-SERVICE-GOAL-#10 (CATERING-TRUCK)
                         ([Mon 4:40pm] to [Mon 5:16pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#11 (BAGGAGE-TRUCK)
                         ([Mon 4:49pm] to [Mon 5:16pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [UNLOAD-BAGGAGE]-SERVICE-GOAL-#12 (BAGGAGE-TRUCK)
                         ([Mon 4:40pm] to [Mon 5:10pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [BAGGAGE-TRANSFER]-SERVICE-GOAL-#13 (BAGGAGE-TRUCK)
                         ([Mon 3:40pm] to [Mon 5:16pm]) <[Flight #235/235(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [CLEAN]-SERVICE-GOAL-#14 (CLEANING-TRUCK)
                         ([Mon 4:44pm] to [Mon 5:10pm]) <[Flight #1447/1447(Mon)D]-#0>.
Created Service Goal --> [Mon 1:36pm] [REFUEL]-SERVICE-GOAL-#15 (FUEL-TRUCK PUMP-TRUCK)
                         ([Mon 4:44pm] to [Mon 5:10pm]) <[Flight #1447/1447(Mon)D]-#0>.
```

⋮

Figure 4.7. Execution Trace Showing the Triggering and Partial Execution of the *Instantiate Task Network* KS.

has already been set aside within the order schedule for those tasks not requiring any resources. The dashed lines indicate the initial allowances for the resource-requiring tasks. A completed, feasible schedule for this flight is presented later.

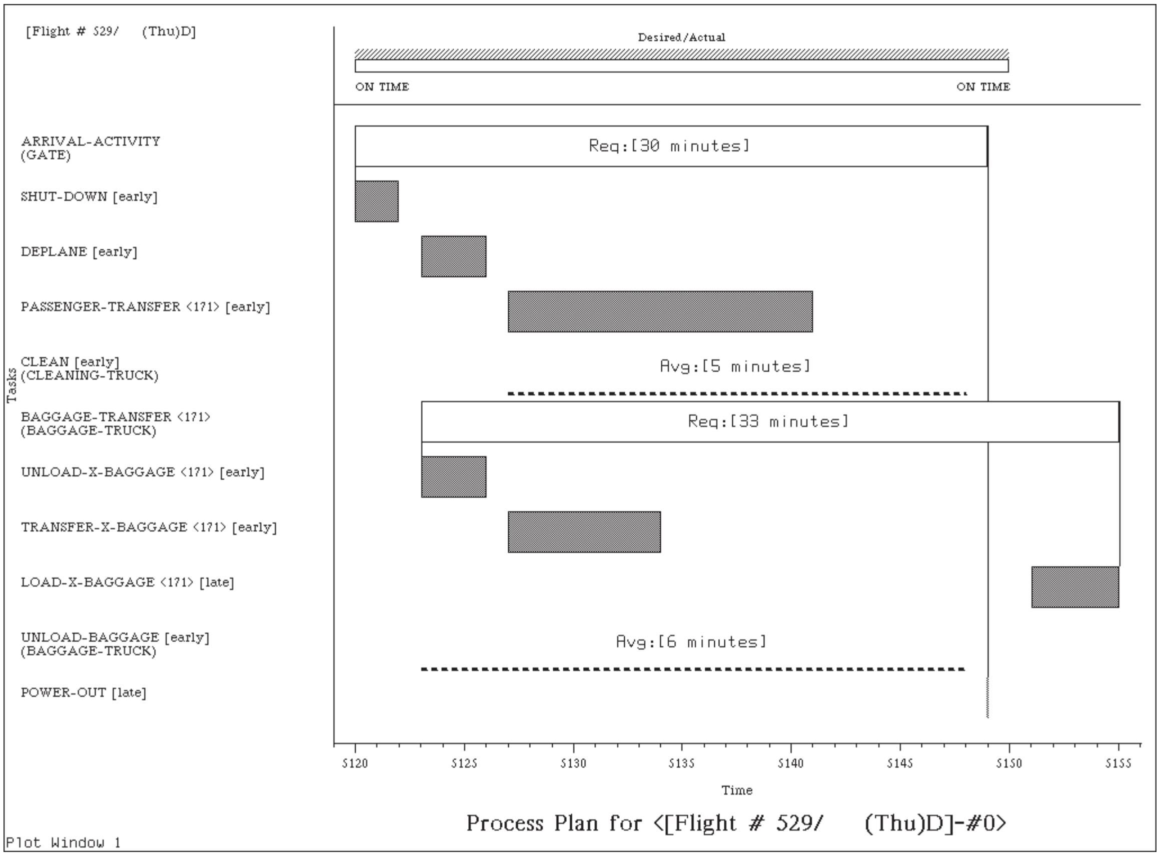


Figure 4.8. A Sample Initial Task Network.

4.2.2 Service Goal Urgency

The heart of the opportunistic scheduling approach implemented in DSS resides in the service goal urgency-rating mechanism, which is responsible for considering a broad range of scheduling perspectives in the process of determining the relative urgency of each service goal produced for an RCSP. The ratings of the service goals and their stimulated KSAs determine the path by which DSS explores its developing search space. The effectiveness of this process affects the quality of the schedules that are produced. Well-focused problem-solving depends on an informed search process that is capable of understanding the topology of the search space and detecting the various textures of the space that indicate areas warranting immediate attention. A misinformed goal-rating process results in a decision-making approach that can extend the search process and lead to lower-quality solutions.

In this section, we provide a formal description of the goal-rating process employed by DSS. Service goals are rated upon creation, and then re-rated at any time the expected amount of difficulty associated with their satisfaction is affected by an internal scheduling decision or external event. The solution of an overall scheduling problem requires the satisfaction of every service goal created. To take advantage of the various perspectives that exist in dynamic RCSPs, the DSS goal-rating mechanism is designed to consider a broad range of information in the process of determining a rating for each service goal. The rating mechanism calculates the urgency of a service goal using a combination of factors derived from the appropriate multiple perspectives. The perspectives consulted in this process are described below.

- **Resource Demand**. The current level of demand for a set of potentially satisfying resources over the period of time specified by a service goal's allowance contributes to an estimate of the anticipated difficulty in finding a satisfying reservation.

- **Resource Availability**. The current level of supply for a particular resource class over time augments the information regarding resource demand. It represents an additional resource-centered perspective that helps to accurately determine resource bottleneck status, the levels of which are rarely stable in real-world dynamic environments.

- **Slack Time**. The present amount of slack time for a particular operation, as determined by its expected processing duration and the size of its allowance, indicates the current degree of flexibility built into a developing order schedule. The amount of existing slack time provides information about the chances of finding a resource reservation for a task that will not incur substantial propagation and schedule-disruption costs upon being introduced. This order-centered perspective affects the urgency of a subproblem in that the more slack that is available, the better the chance the scheduler has to satisfy a reservation request, owing to the larger number of potential reservation times there are to consider.

- **Task Processing Duration**. The expected processing duration for a task provides another important operation-centered perspective on the urgency of a particular subproblem. Large reservations have a serious impact on the developing resource schedules, because they use up large blocks of time, and thereby limit the ability of the scheduler to handle future requests for the same resource class. However, the longer such (large) reservation requests are put off by the scheduler, the more difficult they will be to obtain when finally addressed, and the more costly their introduction to the schedule will become. Smaller reservations may be put off with less danger of their satisfaction causing substantial disruption to an existing schedule.

- **Temporal Urgency**. Temporal urgency helps to determine when a particular subproblem should be dealt with, despite all other concerns. This operation-centered perspective indicates the proximity of the LST for a subproblem to the current real-world time. The information provided by this perspective is extremely important (and only relevant) when scheduling within dynamic environments, where both predictive and reactive scheduling tasks must be integrated into the developing scheduling strategy while real-world events continue to develop. If an operation must begin executing in the near future, it warrants immediate attention from the scheduler.

- **Order Priority**.

  The priority of the order to which a task belongs may have a substantial impact on schedule quality in the form of tardiness penalties. Consideration of order priority allows the scheduler to consult an order-centered perspective in the process of determining subproblem urgency.

- **Task Attributes**.

  Special task attributes may influence the scheduling process, as in the case of inter-order tasks. The scheduling of an inter-order task has a direct impact on two orders, and possibly an extended impact on additional orders. Scheduling decisions that involve inter-order tasks should consider such potentially wide-ranging impact.

By combining the information obtained through consultation with the above scheduling perspectives, the goal-rating mechanism is able to determine a value that indicates the urgency of any particular scheduling subproblem. We now discuss the urgency-rating mechanism in detail.

4.2.2.1 Service Goal Urgency Determination

For the following discussion, we define T_{ij} to be the jth task within an order O_i, and G_{ij} to be its corresponding service goal. The *expected* processing duration for a task T_{ij} is accessible by $\textsf{ExpDuration}(T_{ij})$. For aggregate tasks, the expected processing duration is equal to the allowance of G_{ij}, also accessible by $\textsf{Allowance}(G_{ij})$. For non-aggregate tasks, the expected processing duration is equal to the average of the minimum and maximum expected service activity processing durations over the set of all legal resource classes for the task.

An important concept in the process of determining the current levels of contention for the various resource classes over time is the notion of *service goal overlap.* This relationship provides DSS with a perspective on the search space that highlights areas of resource contention that warrant immediate attention by the scheduler. We preface our discussion of the urgency-calculation formula with a formal definition of service goal overlap.

Throughout the scheduling process, each service goal created by DSS maintains links to all existing resources of the legal classes for its corresponding task that also satisfy *all* relevant technological and contextual constraints on a task. We refer to this list for a service goal G by $\textsf{LegalResources}(G)$. We also define the temporal overlap, $\textsf{Overlap}(I1, I2)$ (for two time intervals $I1$ and $I2$) to be the size of their intersecting portions, that is, the amount of time common to both. Formally, using $\textsf{LowerBound}(I)$ and $\textsf{UpperBound}(I)$ to return the lower and upper bounds (respectively) of a temporal interval I, $\textsf{Overlap}(I1, I2)$ may be described as follows:

$$\begin{aligned}\textsf{Overlap}(I1, I2) \;=\; 1+ &\; \min(\textsf{UpperBound}(I1), \textsf{UpperBound}(I2)) \\ - &\; \max(\textsf{LowerBound}(I1), \textsf{LowerBound}(I2))\end{aligned}$$

Two goals $G1$ and $G2$ are considered to *overlap* under the following circumstances:

$$\begin{aligned}\textsf{Overlap}(\textsf{Allowance}(G1), \textsf{Allowance}(G2)) &> 0 \quad \text{and} \\ \textsf{LegalResources}(G1) \wedge \textsf{LegalResources}(G2) &\neq \emptyset\end{aligned}$$

Service goal overlap is thus determined according to the set of legal resource candidates common to each goal and the intersection of their shared allowances. We will refer to the set of overlapping goals for a goal G by $\textsc{OverlappingGoals}(G)$. This set is continuously updated by DSS throughout the scheduling process.

We are now ready to present the service goal urgency-rating formula. It is a two-phase calculation, involving the determination of the bottleneck status for the legal resource candidates for a goal, and the modification of that value according to a number of additional perspectives.

4.2.2.1.1 Step 1–A: Resource Contention

The first calculation determines the current level of demand for the resource classes being sought by the service goal. This value provides an indication of the current level of contention for the set of legal resources available to a particular service goal among the set of all unsatisfied service goals that overlap with that goal. The resource contention value for a service goal G is calculated using the following equation, where $\mathcal{G} \leftarrow \textsc{OverlappingGoals}(G)$.[7]

$\textsc{ContentionValue}(G) =$

$$\sum_{G_{ij} \in \mathcal{G}} \begin{cases} 0 \text{ if } G_{ij} \text{ is satisfied, otherwise} \\ \left[\begin{array}{c} \min(\textsc{ExpDuration}(T), \textsc{Overlap}(\textsc{Allowance}(G_{ij}), \textsc{Allowance}(G))) \\ * \left(\frac{\textsc{ExpDuration}(T_{ij})}{\textsc{Size}(\textsc{Allowance}(G_{ij}))} \right) \end{array} \right] \end{cases}$$

The minimization clause ensures that no amount of temporal overlap with another goal will exceed the expected processing duration for the task in question. This value is then further modified by factoring in the available slack for an overlapping goal so that the impact of its own flexibility on the degree of overlap is considered. Overlap with highly flexible goals (those goals having significant available slack in their allowances) is thereby discounted. The resource contention value is thus based on the amount of *real* overlap between the original goal and each of its overlapping goals. As such, it provides an accurate assessment of the amount of contention for a common set of resources at a particular time.

Finally, as shown in Equation 4.1, the resource contention value is normalized by both the number of unsatisfied overlapping goals $\mathcal{C} \subseteq \mathcal{G}$ and the expected processing duration for the task involved (T). If $\mathcal{C} = \emptyset$, then the contention value is 1.

$$\frac{\frac{\textsc{ContentionValue}(G)}{|\mathcal{C}|}}{\textsc{ExpDuration}(T)} \tag{4.1}$$

4.2.2.1.2 Step 1–B: Resource Availability

After calculating a measure of the existing level of resource contention among a service goal and its (unsatisfied) overlapping goals, DSS next determines the current level of availability for the set of legal resource candidates for that service goal. The resource availability value provides an accurate assessment of the expected difficulty of satisfying a particular service goal, comprising the levels of both availability and existing demand for its set of legal candidate resources.

To determine the resource availability value, we first describe an important aspect of our resource representation scheme. Each resource R_i maintains an array of state descriptions that

[7] Note that each service goal G_{ij} (or G) is linked directly to its corresponding task T_{ij} (or T).

each indicate a particular activity and the time interval during which that activity is scheduled to occur. The entries in this array are indicated by $R_i[k]$ (each R_i having K_i activity states). The activity scheduled during the kth state of resource R_i is referenced by $\textsf{ACTIVITY}(R_i[k])$, while the time interval itself is accessed by $\textsf{RANGE}(R_i[k])$. At this point we also introduce the $\textsf{SIZE}(I)$ construct, used to return the size (or duration) of a time interval I.

The current level of availability for the set of resources for a goal G is calculated using the following equation, where $\mathcal{L} \leftarrow \textsf{LEGALRESOURCES}(G)$.

$\textsf{RESOURCEAVAILABILITY}(G) =$

$$\sum_{R_i \in \mathcal{L}} \sum_{k=1}^{K_i} \begin{cases} 0 \text{ if } \textsf{ACTIVITY}(R_i[k]) = \texttt{UNUSED}, \text{ otherwise} \\ \textsf{OVERLAP}(\textsf{ALLOWANCE}(G), \textsf{RANGE}(R_i[k])) \end{cases}$$

Finally, as shown in Equation 4.2, the resource availability value for a service goal is normalized by both the number of legal candidate resources for the goal $\mathcal{L}$ and the size of its allowance. If $\mathcal{L} = \emptyset$, then the resource availability value is 1.

$$\frac{\frac{\textsf{RESOURCEAVAILABILITY}(G)}{|\mathcal{L}|}}{\textsf{SIZE}(\textsf{ALLOWANCE}(G))} \tag{4.2}$$

4.2.2.1.3 Step 1–C: Available Slack

The amount of slack available to a goal is based on its allowance and the expected processing duration of its task (T). DSS discounts the urgency of service goals having a great deal of slack. The greater the amount of slack available, the less important the resource contention and resource availability values become, because of the increased flexibility for the goal. Equation 4.3 is used to calculate a slack value that contributes to the overall bottleneck status value.

$$\frac{\textsf{EXPDURATION}(T)}{\textsf{SIZE}(\textsf{ALLOWANCE}(G))} \tag{4.3}$$

The results of Equations 4.1 and 4.2 and 4.3 are then combined with a constant C to produce an overall *bottleneck status value* that quantifies, for a particular service goal, the current scarcity among the set of resource candidates that may be considered to satisfy the goal within its allowance. Note, however, that service goals that do not overlap any other service goals will receive a value of 0 as a result of this first rating phase.

The second phase of the urgency-calculation formula consists of a series of modifications to the bottleneck status value, in consideration of the other important scheduling perspectives on the service goal. DSS ignores the bottleneck status value if it is less than 1, in which case the current level of resource contention is deemed too low to be a serious factor. Three additional factors are combined in this second phase to produce the final urgency rating.

4.2.2.1.4 Step 2–A: Temporal Urgency

This step accounts for the immediacy of a service goal's task. By maintaining a record of the farthest downstream due date among the entire order set, DSS is able to gauge the temporal urgency of each service goal by comparing its allowance with this value. The following form is used for this purpose.

$$\frac{*\textsf{LargestDueDate}* - \begin{cases} \textsf{LowerBound}(\textsf{Allowance}(G)) & \text{if } T \text{ is an aggregate} \\ \left\lceil \begin{array}{c} \textsf{UpperBound}(\textsf{Allowance}(G)) \\ -\textsf{ExpDuration}(T) \end{array} \right\rceil & \text{otherwise} \end{cases}}{*\textsf{LargestDueDate}*}$$

The result of the above equation is combined with the value of the ∗TimeGranularityFactor∗ parameter to account for the different time scales used by various DSS applications.

4.2.2.1.5 Step 2–B: Order Priority

High-priority orders generally incur higher tardiness penalties. The priority of the order involved is factored directly into the result so that orders with high priority receive a boost in accordance with their rank.

4.2.2.1.6 Step 2–C: Inter-Order Task Consideration

Finally, inter-order tasks receive a boost as a result of their extent across multiple orders. Unless a task is an inter-order task, it is discounted by a constant value less than 1.

The urgency-rating for a service goal is thus a product of multiple perspectives on the goal, task, and order involved, as well as the legal resource candidates being considered. DSS thereby attains an enhanced degree of flexibility for detecting developments in the scheduling environment.

4.2.2.2 The Frequency of Service Goal Urgency Determination

A dynamic environment complicates the scheduling process by generating external events that violate the expectations of the scheduler, and require it to continually modify its assumptions about the current state of problem-solving. For example, the reception of a new order increases the expected demand for certain resource classes, possibly leading to a change in expected bottleneck conditions. A resource failure decreases the availability of a particular resource class, and delayed servicing activity by a particular resource may nullify the scheduled starting times for existing downstream reservations. Both developments may alter expected bottleneck conditions.

At specific points during the scheduling process, the impact of both internally and externally generated events is considered by DSS in an attempt to determine how to adapt its current focus of attention. As the search space is modified, the process of exploring it must be adjusted in response. With each received event, modification of the urgency of each related unsolved subproblem may be required. Listed below are some of the key decision points throughout the scheduling process where DSS takes the opportunity to consider the modification of its present decision-making strategy by updating the urgency of some of its pending scheduling tasks.

- *Goal Creation.* All service goals whose allowance and set of legal resources overlap with those of a created service goal require re-rating as a result of the increase in resource demand.

- *Goal Modification.* All service goals sharing a desire for the same resources, and whose allowance either previously overlapped or currently overlaps the allowance of a modified service goal require re-rating as a result of the change in resource demand.

- *Secured Reservation.* All service goals sharing a desire for a newly secured resource, and whose allowance overlaps with the time bounds for the new reservation or the allowance of the newly satisfied service goal, require re-rating as a result of the decrease in resource availability.

- *Canceled Reservation.* All service goals sharing a desire for a newly released resource, and whose allowance overlaps with the time bounds for the canceled reservation or the allowance of the newly unsatisfied service goal, require re-rating as a result of the increase in resource availability.

- *Resource Failure.* All service goals that could have been satisfied by a failed resource require re-rating as a result of the decrease in resource availability.

- *Resource Creation.* All service goals that could be satisfied by a newly available resource require re-rating as a result of the increase in resource availability.

Figure 4.9 includes another segment of a DSS execution trace showing the service goal rating and re-rating activities triggered by the creation of new service goals by the *Instantiate Task Network* KS. Additional examples of goal re-rating activity triggered by the introduction of resource reservations (as described in Section 4.2.4) are shown in Figures 4.15, 4.16, and 4.17. The activity shown in Figure 4.9 occurs following the creation of all service goals necessary for arranging the *Turnaround* servicing of <[Flight #416/416(Mon)I]-#0>.

For example, the creation of [REFUEL]-SERVICE-GOAL-#52 leads to an increase in the rating of the related [REFUEL]-SERVICE-GOAL-#29 (from 602 to 3141), owing to the sudden increase in demand for the fuel truck and pump truck resource classes. [CLEAN]-SERVICE-GOAL-#51 and [SERVICE]-SERVICE-GOAL-#47 have a similar impact on [CLEAN]-SERVICE-GOAL-#28 and [SERVICE]-SERVICE-GOAL-#24, increasing their ratings from 602 to 4580, and from 545 to 1531, respectively. In each of these three cases, the existing related service goals had no overlapping goals prior to the creation of the new goals, which accounts for the substantial increases to their urgency ratings. While the new [RESTOCK]-SERVICE-GOAL-#48 does overlap with an existing goal, the effect on that goal is minimal enough not to require an update to its urgency.

The creation of the UNLOAD BAGGAGE and LOAD BAGGAGE service goals has a slightly different impact on the related baggage activity goals ([LOAD-BAGGAGE]-SERVICE-GOAL-#26, [UNLOAD-BAGGAGE]-SERVICE-GOAL-#27, [BAGGAGE-TRANSFER]-SERVICE-GOAL-#35, and [BAGGAGE-TRANSFER]-SERVICE-GOAL-#43), due in part to the prior existence of overlapping goals for each related goal. The new baggage activity goals are all rated highly as the result of existing levels of contention. But while the new baggage tasks do contribute to an increase in the demand for baggage trucks, the amount of increase, when distributed among the related tasks, is smaller than previous levels, so that the overall contention level for each related service goal actually drops slightly.

```
-------------------------------- KSA 54 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #416/416(Mon)I]-#0>)
```

⋮

Service Goal Creation

⋮

```
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [REFUEL]-SERVICE-GOAL-#52 (FUEL-TRUCK PUMP-TRUCK) {3424}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#29 (FUEL-TRUCK PUMP-TRUCK)
                         from {602} to {3141}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [CLEAN]-SERVICE-GOAL-#51 (CLEANING-TRUCK) {6104}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#28 (CLEANING-TRUCK)
                         from {602} to {4580}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#50 (BAGGAGE-TRUCK) {8894}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#49 (BAGGAGE-TRUCK) {8593}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#43 (BAGGAGE-TRUCK)
                         from {16013} to {14733}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#35 (BAGGAGE-TRUCK)
                         from {17904} to {16797}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#27 (BAGGAGE-TRUCK)
                         from {19077} to {16096}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#26 (BAGGAGE-TRUCK)
                         from {16830} to {14362}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [RESTOCK]-SERVICE-GOAL-#48 (CATERING-TRUCK) {495}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [SERVICE]-SERVICE-GOAL-#47 (SERVICE-TRUCK) {1635}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#24 (SERVICE-TRUCK)
                         from {545} to {1531}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#46 (GATE) {767}.
```

⋮

Figure 4.9. Execution Trace Showing the Service Goal Rating Activity Triggered by the *Instantiate Task Network* KS.

4.2.3 Selecting Reservation-Securing Knowledge Sources

DSS is equipped with a collection of KSs designed to find resource reservations for task nodes. The process of selecting a particular reservation-securing KS to use in any given situation is handled by the *Select Reservation-Securing Method* KS. It is responsible for considering the status of the service goal for a task node, including the results of any previous failed attempts to find a reservation for the subproblem. These records are maintained in a KS ACTIVATION TABLE that is stored with each service goal. Upon the selection of a particular reservation-securing method to attempt, the KS that implements the selected method is triggered, and the service goal's KS ACTIVATION TABLE is updated accordingly.

Every reservation-securing KSA execution records the results of its search (success or failure) into the service goal's KS ACTIVATION TABLE, so that the service goal always contains an up-to-date record of the status of the search for a satisfying reservation. If the executed KSA finds a reservation, then the stimulating service goal is marked as satisfied, and the success of the method is recorded in the table. If, however, the executed KSA fails to find a successful reservation, then the failure is recorded in the table, and the *Select Reservation-Securing Method* KS is triggered again. It will analyze the remaining options for finding a reservation and trigger another reservation-securing KS. If no remaining method is found to attempt to satisfy a service goal, indicating that none of DSS's reservation-securing KSs has been able to find a solution for the subproblem, then the scheduler is unable to produce a schedule for the entire problem.

Some of the reservation-securing KSs in DSS are designed to find reservations within service goal allowances, while others are designed to find any possible reservation in situations where a reservation within the allowance may not be achieved. The size of the allowance, therefore, has a major impact on the process of selecting which particular reservation-securing KS should be used to attempt to satisfy a particular subproblem.

A service goal's allowance may easily be modified through the actions triggered by a reservation-securing KS, most often as the result of the propagation of timing constraints forced by the introduction of new resource reservations. Whenever a resource reservation is introduced into the developing schedule, the allowances for the service goals corresponding to the immediately preceding and succeeding tasks, and possibly some parent tasks as well, must be modified to reflect the impact of the time bounds for the new reservation. Because the urgency of a subproblem is partly based on the amount of difficulty that is expected in the process of trying to find a successful reservation within the allowance of a service goal, any change in the size of the allowance affects the overall urgency of the subproblem. Therefore, in the case of a modified allowance, the re-sized service goal must be rerated, and the efforts to find a successful reservation for the subproblem must be restarted, so that the scheduler may again determine the most appropriate reservation-securing KS for solving the modified subproblem.

At any point when the allowance for a service goal is modified, all pending KSAs previously stimulated by the service goal are obviated, the KS ACTIVATION TABLE is reset to its initial state, and the *Select Reservation-Securing Method* KS is triggered to start the process over again using the new allowance.

The current strategy employed by the *Select Reservation-Securing Method* KS is to simply go down the list of available reservation-securing methods that appear in the service goal's KS ACTIVATION TABLE, and choose the next untried method. The table is initialized at the time a service goal is created, and the entries are arranged so that the less computationally expensive and less propagation-inducing methods are attempted first. While this approach is perhaps more simple than it should be, it provides a workable means for regulating the attempts to secure reservations for resource-requiring tasks.

Some product tasks may be performed by more than a single kind of resource, so the service goal for a product task may be looking at more than one kind of resource to satisfy its requirements. Therefore, the service goal maintains a separate KS ACTIVATION TABLE of reservation-securing methods for each class of resource that may satisfy its requirements. When the *Select Reservation-Securing Method* KS processes the service goal's table, it actually does so for each allowable resource class, thereby possibly triggering multiple reservation-securing KSs to satisfy the service goal. As soon as any one of these KSs succeeds, all other pending reservation-securing KSAs for the service goal are obviated, and the search to satisfy the service goal is ended.

Figure 4.10 presents a portion of a DSS execution trace showing the multiple triggerings and resulting executions of the *Select Reservation-Securing Method* KS following the execution of the *Instantiate Task Network* KS. Note that because `[REFUEL]-SERVICE-GOAL-#22` may be satisfied by either a pump truck or fuel truck resource, the execution of the corresponding *Select Reservation-Securing Method* KSAs serves to activate two reservation-securing KSs, one for each resource class. Remember also that the reservation-securing KSs inherit the urgency ratings of their stimulating service goals.

There are a variety of ways in which the process of selecting particular methods for finding resource reservations could be altered, in order to take into consideration a broader range of information about the state of the search. Because some reservation-securing methods are computationally faster than others, these methods could be encouraged in situations where the subproblem involved is extremely urgent. Subproblems requiring highly constrained resources may warrant the selection of methods that are better at finding reservations in densely packed schedules. In lesser constrained situations, the standard approaches may be the most appropriate. An interesting area for future research (discussed in Section 6.3.2) involves the process of determining what aspects of the current state of problem-solving would be relevant in the process of deciding the exact reservation-securing methods to use in any given situation.

Before discussing the particular methods available for securing resource reservations, we include Figure 4.11, showing a completed feasible order schedule for the previously presented `<[Flight #529/(Mon)D]-#0>` from Figure 4.8. At this point, all required resources have been secured so that the ground servicing activity for the flight will both start and finish on time.

4.2.4 Securing Resource Reservations

Each reservation-securing KS is responsible for finding a single resource of a specific class to satisfy a particular service goal. If a service goal may be satisfied by multiple resource classes, then individual KSs for each resource class will be triggered. Each KS follows a standard procedure of first determining a list of candidate resource and reservation pairs that may satisfy the service goal, and then sorting the candidates to determine the most appropriate resource and reservation pair to select based on the current state of problem-solving.

It is important to note that the reservation-securing KSs will never introduce conflicts into the schedule. Each KS is designed to find only those reservations that satisfy all currently applicable constraints. While the decisions made by these KSs may serve to further constrain

```
--------------------------------- KSA 37 ---------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #516/516(Mon)D]-#0>)

                                        ⋮

                              Service Goal Creation

                                        ⋮

                         Service Goal Rating and Re-Rating

                                        ⋮

Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([SERVICE]-SERVICE-GOAL-#17)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([SERVICE]-SERVICE-GOAL-#17)

                                        ⋮

Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([REFUEL]-SERVICE-GOAL-#22)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([REFUEL]-SERVICE-GOAL-#22)
Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#16)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#16)
--------------------------------- KSA 38 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([SERVICE]-SERVICE-GOAL-#17)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([SERVICE]-SERVICE-GOAL-#17)
KS Activation ---------> STANDARD-ASSIGNMENT ([SERVICE]-SERVICE-GOAL-#17)

                                        ⋮

--------------------------------- KSA 43 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([REFUEL]-SERVICE-GOAL-#22)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([REFUEL]-SERVICE-GOAL-#22)
KS Activation ---------> STANDARD-ASSIGNMENT ([REFUEL]-SERVICE-GOAL-#22)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([REFUEL]-SERVICE-GOAL-#22)
KS Activation ---------> STANDARD-ASSIGNMENT ([REFUEL]-SERVICE-GOAL-#22)
--------------------------------- KSA 44 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#16)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#16)
KS Activation ---------> STANDARD-ASSIGNMENT ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#16)

                                        ⋮
```

Figure 4.10. Execution Trace Showing the Triggering and Execution of the *Select Reservation-Securing Method* KS.

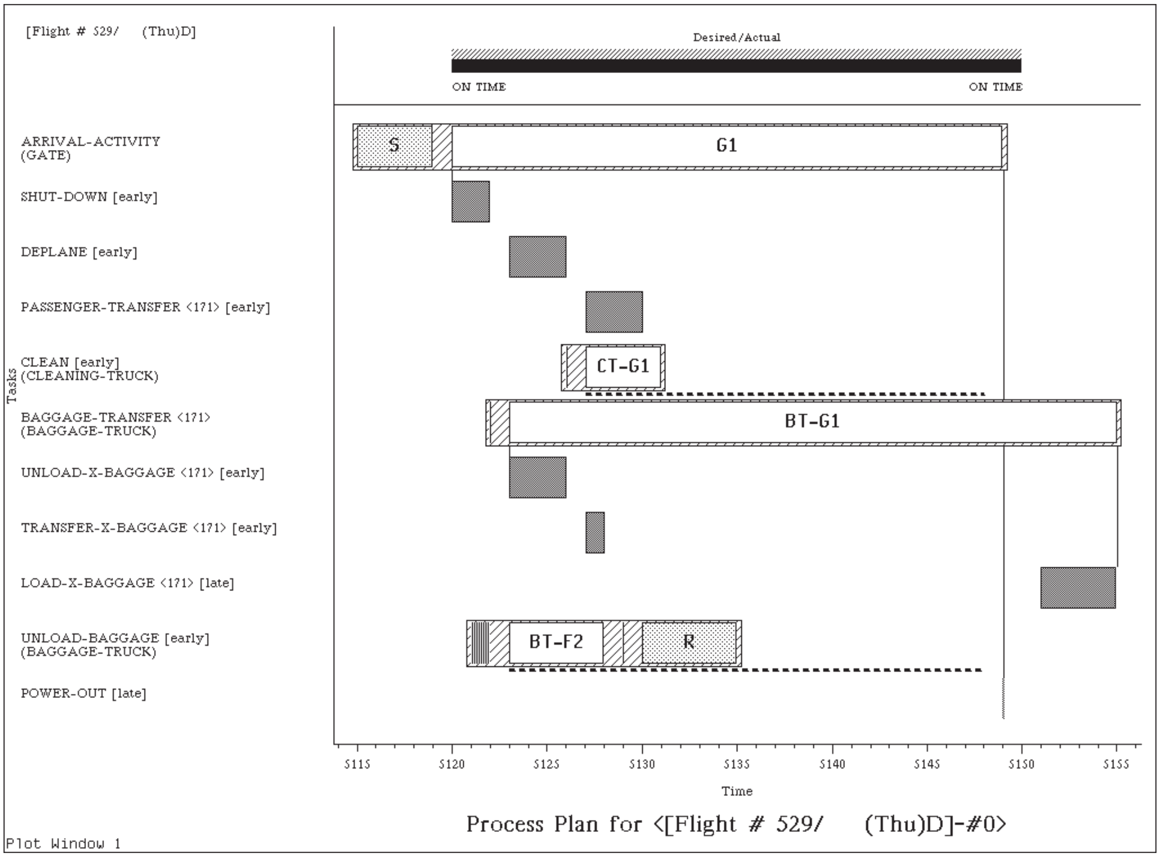


Figure 4.11. A Sample Feasible Completed Task Network.

the state of problem-solving and contribute to the development of future conflicts, no constraint will be violated when a scheduling decision is made.

DSS is equipped with two important sets of reservation-securing KSs. The first, called the *Standard* set, seeks to produce resource reservations without affecting any existing decisions. These KSs will fail in situations where no reservation may be found that satisfies all existing constraints. The second set of KSs is the *Relaxed* set, which is specifically designed to produce reservations in situations that require either existing reservations to be canceled, or various constraints to be relaxed.[8] The two sets of reservation-securing KSs contain different versions of three basic reservation-securing methods, namely *Assignment*, *Preemption* and *Right Shift*. The three standard versions of these methods are described in the following three sections. The relaxed methods of these methods are then addressed collectively in Section 4.2.4.5.

4.2.4.1 Special Considerations for Securing Mobile Resources

In domains equipped with mobile resources, the allocation of travel time as part of mobile resource reservations constrains the selection of the shop locations to and from which they must

[8] Note that while the *(Standard) Preemption* KS will cancel existing reservations in order to produce a reservation for a particular subproblem, it will not cancel any other existing reservations or relax any constraints. The *Relaxed Preemption* KS, on the other hand, is designed to do just that.

report. For example, returning to our simplified AGSS problem from Section 1.4, a baggage truck for a particular flight may have been secured before the flight has been assigned to a gate. In this case, the amount of time allocated in the reservation to allow the baggage truck to get from its previous location, to the location of any candidate gate, defines a radius within which any candidate gate must be located for the baggage truck to be able to report within the amount of time allocated.

To avoid the unnecessary introduction of constraints into the developing schedule, DSS uses a worst-case policy for reserving mobile resources. Figure 4.12 illustrates how this approach works. If the context within which a mobile resource is to be used is incomplete, DSS preserves the set of feasible candidates for the relevant unsolved stationary resource subproblem by constructing an *unrefined* reservation for the mobile resource that allows it to get to *any* possible destination location. The set of feasible destination location candidates is preserved by constructing a reservation that allows enough travel time for the mobile resource to reach all candidates.

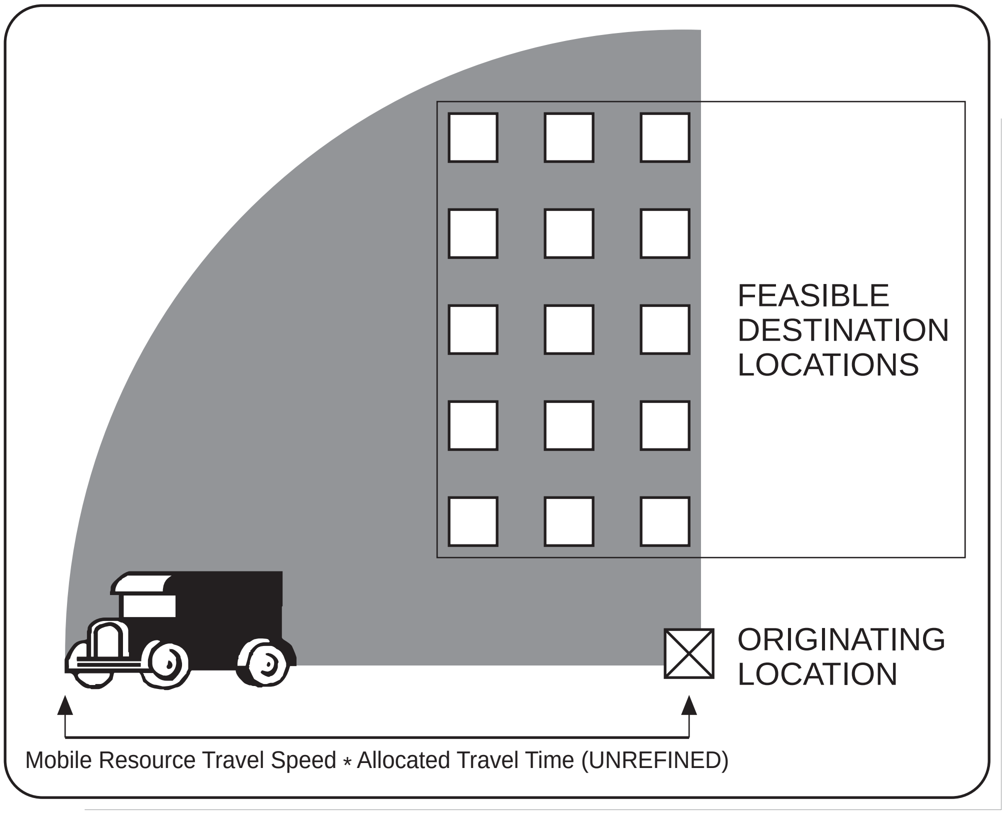

FEASIBLE DESTINATION LOCATIONS
ORIGINATING LOCATION
Mobile Resource Travel Speed * Allocated Travel Time (UNREFINED)

Figure 4.12. Inclusion of *Unrefined* Travel Time Allocation in a Mobile Resource Reservation.

In situations where the context within which a mobile resource is to be used is completely established, DSS will construct *refined* mobile resource reservations that allow just enough travel time for a mobile resource to get to and from its assigned locations. Figure 4.13 illustrates this process. If a destination location has already been selected, then the reservation for the truck need only allow enough time to get from its previous location to the selected destination location. Note that such a decision does preclude a number of possible destination locations from future consideration, but such a location has already been secured. As long as the currently selected destination location remains viable, there has been no serious increase in the degree of constraints on the related subproblems.

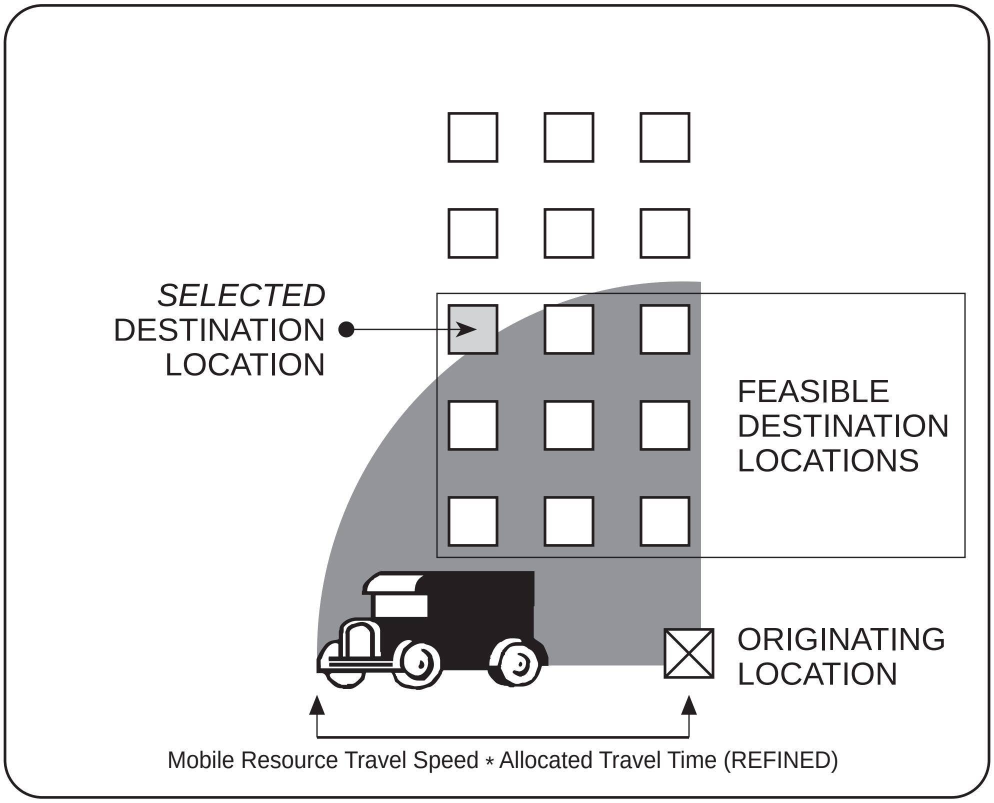


Figure 4.13. Inclusion of *Refined* Travel Time Allocation in a Mobile Resource Reservation.

The basic idea behind our worst-case approach to securing mobile resources is to preserve the options available to the scheduler. In dynamic, unpredictable environments, a conservative approach provides a degree of flexibility that can prove to be valuable in helping to resolve conflict situations. Because resource utilization is also a concern for DSS, a reservation refinement mechanism has been developed for returning to previous uncertain mobile resource reservations and recalculating the exact amount of travel time required once the complete context for their use has been determined. The refinement mechanism is described in Section 4.2.6.

It should also be noted, however, that there are costs associated with our worst-case approach to constructing mobile resource reservations, in terms of the decrease in schedule quality that arises from introducing too many bloated mobile resource reservations into the developing schedule. Unrefined mobile resource reservations are generally larger than they need to be, and as a result they tend to use up valuable resource time more quickly, thereby reducing the availability of those resources and leading to increased numbers of conflicts that result in potentially costly levels of tardiness. The larger the reservations, the less easy it is to find reservations within the desired subproblem allowances. The potential for future work in this subject is discussed in Section 6.3.5.

At this point we begin with our discussion of the three basic resource reservation-securing KSs incorporated into DSS.

4.2.4.2 The Assignment Knowledge Source

The *Assignment* KS is the most basic reservation-securing KS. Its goal is to secure a resource reservation that lies within the confines of the allowance for a service goal.

In the process of building its initial list of candidate resource and reservation pairs, the *Assignment* KS first considers the set of resources of the designated class that are available (by residing in an `UNUSED` state) at any point that overlaps with the service goal's allowance, satisfy the various technological constraints imposed upon the resource selection process by the order and product involved, and satisfy the existing contextual constraints on the subproblem. The technological and contextual constraints act to shrink the pool of resources that may be considered for satisfying a particular resource request. Once the list of candidates has been assembled, the *Assignment* KS checks each resource to see if it is available for the expected amount of time that will be required to perform the desired service.

After all of the possible resource and reservation pairs have been found, the KS proceeds to select a resource and reservation pair through a sequence of steps. It first sorts the list in order to favor those pairs having the latest start time for the complete resource plan (all `EN ROUTE`, `SETUP`, `RESET` and `SERVICING` tasks), so that future arrangements of the list will favor those reservations requiring the shortest setup times. A second sort arranges the pairs so that the earliest reservations for early-shifted or non-late-shifted tasks, or the latest reservations for late-shifted tasks, are moved to the top of the list. This ordering helps to achieve the goal of maximizing order schedule quality.

Finally, depending on the current level of contention for the resource class to which the candidate resources belong, the KS will attempt to optimize either the resource schedules or the order schedules. If contention is high (above 50%), a reservation will be selected that minimizes the introduction of any fragmentation into an existing resource schedule, so that valuable resource time is not wasted. This requires a final sort of the candidate list to find (and return) the reservation that minimizes the amount of time between the candidate reservation and any preceding or succeeding (depending on the task's shift preference) reservation. If contention is low, the reservation at the top of the list will be selected, owing to its highest-ranked satisfaction of the task shift preference. In both cases, an attempt is made to minimize the amount of setup time required for the reservation. The current level of contention for a particular resource class is determined by calculating the average ratio of resources (in the class) that are in use or requested, to the total number of resources of the class that are available, over the time interval specified by the service goal's allowance.

The other reservation-securing KSs implemented in DSS, because they are used in more highly constrained situations, where fewer of the overall scheduling objectives are expected to be properly satisfied, rely on a more order-based approach to selecting reservations, although the attempt to minimize the amount of required setup time is still made.

Figure 4.14 provides a visual description of the value ordering process employed by the *Assignment* KS.

The *Assignment* KS is the quickest of the reservation-securing KSs. Additionally, it requires the least amount of time bounds constraint modification, and forces no changes to existing reservations. It completely localizes its temporal impact to lie within the allowance of the service

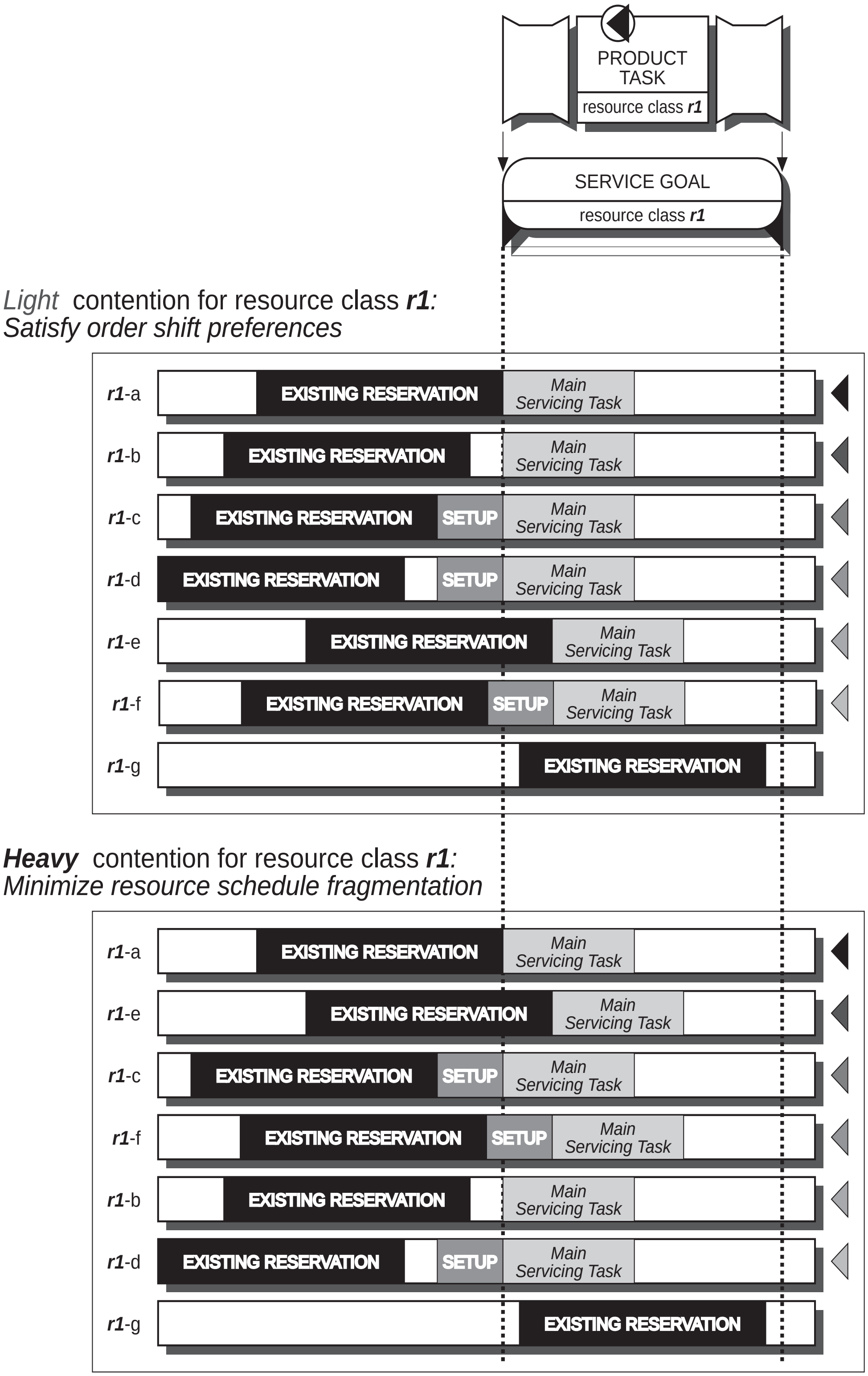


Figure 4.14. *Assignment* Knowledge Source Value Orderings.

goal for the subproblem, and thus does not affect any other subproblems. By selecting only those reservations that reside within the range of the allowance, the KS forces only the standard updating of the timing constraints of any upstream or downstream subproblems within the same order (or any connected orders), while introducing no timing conflicts. Any reservation constructed by this KS helps to maintain flexibility for the scheduler by minimally constraining the current state of problem solving.

Figure 4.15 presents a portion from a DSS execution trace showing the execution of the *Assignment* KS. In this situation, the current allowance for `[REFUEL]-SERVICE-GOAL-#29` is `([Mon 3:50pm] - [Mon 4:12pm])`. A successful reservation involving `Fuel-Truck-#FT-F1` is constructed, which allows the main REFUEL servicing task to occur from `([Mon 3:50pm] - [Mon 4:00pm])`. Notice that the process plan for using a fuel truck for this operation requires that the setup- and travel-related resource tasks begin executing at `[Mon 3:40pm]`. After the standard assignment is made, the pending KSA (`<KSA-112 STANDARD-ASSIGNMENT>`) responsible for trying to secure a pump truck to satisfy `[REFUEL]-SERVICE-GOAL-#29` is obviated, and two related REFUEL service goals and their stimulated reservation-securing KSAs are re-rated.

```
--------------------------------- KSA 129 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([REFUEL]-SERVICE-GOAL-#29)
Resource Assignment ---> [Mon 1:40pm] Fuel-Truck-#FT-F1 assigned to
                         [Task Node: <Task: REFUEL> <[Flight #1412/1412(Mon)D]-#0>]
                         ([Mon 3:40pm] - [Mon 4:00pm] ([Mon 3:50pm] - [Mon 4:00pm])).
Obviated KSA ----------> [Mon 1:40pm] KSA <KSA-112 STANDARD-ASSIGNMENT> stimulated
                         by [REFUEL]-SERVICE-GOAL-#29 <[Flight #1412/1412(Mon)D]-#0>
                         obviated due to {ALTERNATE-RESOURCE-SATISFACTION}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-145 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [REFUEL]-SERVICE-GOAL-#59
                          PUMP-TRUCK STANDARD-ASSIGNMENT)
                         from {1906} to {2571}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-144 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [REFUEL]-SERVICE-GOAL-#59
                          FUEL-TRUCK STANDARD-ASSIGNMENT)
                         from {1907} to {2571}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#59 (FUEL-TRUCK PUMP-TRUCK)
                         from {1907} to {2571}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-136 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [REFUEL]-SERVICE-GOAL-#52
                          PUMP-TRUCK STANDARD-ASSIGNMENT)
                         from {2637} to {3218}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-135 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [REFUEL]-SERVICE-GOAL-#52
                          FUEL-TRUCK STANDARD-ASSIGNMENT)
                         from {2638} to {3218}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [REFUEL]-SERVICE-GOAL-#52 (FUEL-TRUCK PUMP-TRUCK)
                         from {2638} to {3218}.

                                         ⋮
```

Figure 4.15. Execution Trace Showing the Execution of the *Assignment* KS.

4.2.4.3 The Preemption Knowledge Source

The *Preemption* KS extends the process of searching for candidate resource and reservation pairs to include the consideration of resources that are currently reserved by previously satisfied subproblems. As with the *Assignment* KS, the goal of the *Preemption* KS is to secure a resource reservation that lies within the confines of the time interval defined by a service goal's allowance, and to minimize the number of existing reservations that must be canceled in order to facilitate such an assignment.

The process employed by the *Preemption* KS to select initial candidate resources works by considering all resources in the designated class that satisfy all of the relevant technological constraints, whether or not they are available during the time interval defined by the service goal's allowance. After the initial list of candidate resource and reservation pairs has been assembled, the KS applies a series of checks to each candidate to determine whether it may be used to satisfy the subproblem. These checks are as follows:

For each reservation that currently exists during the time interval specified by the bounds of the goal, ensure that:

1. All relevant contextual constraints are satisfied.

2. The existing reservation has not begun executing. All tasks are assumed to be atomic, and therefore may not be interrupted once they have begun execution.

3. The priority of the order to which the existing reservation belongs is less than that of the preempting subproblem's order.

4. The servicing activity portion of the existing reservation does not begin earlier than the lower bound on the allowance of the preempting subproblem's service goal.

5. The goal rating for the existing reservation is lower than the goal rating for the preempting subproblem. It is not desirable for the scheduler to cancel an existing reservation for a subproblem that will be even more difficult to re-solve than the current preempting subproblem.[9]

6. If the existing reservation belongs to the same order as the preempting subproblem, then the existing reservation must be for a downstream subproblem, that is, an operation occurring later in the order's process plan. Preempting an existing upstream reservation to satisfy a downstream subproblem would force the scheduler to devote a potentially substantial amount of effort to re-satisfy the preempted upstream subproblem, and probably increase the amount of tardiness that would eventually be incurred by the affected order.

7. If the existing reservation belongs to a different order than the preempting subproblem, then the due date for the existing reservation's order must be later than the due date for the preempting subproblem's order. For reasons similar to the previous check, the cancellation of a reservation that is part of an order with a due-date that is more urgent

[9] The service goals for all subproblems whose reservations are considered for preemption are rerated immediately prior to performing this check.

than that of the preempting subproblem's order would probably increase the amount of tardiness that would eventually be incurred by the impending order.

8. If the due dates of both the existing reservation's order and the preempting subproblem's order are equal, then the number of the preempting subproblem's order must be higher than the number of the existing reservation's order. This check prevents looping in the preemption process.

After performing the above checks on any existing reservations, the *Preemption* KS then has a list of candidate resource and reservation pairs to consider. The candidates are first sorted to minimize fragmentation in each resource schedule, and then sorted to minimize the maximum possible number of existing reservations that must be canceled to satisfy the preempting subproblem. The KS then looks at each candidate resource (in sorted order), and using the corresponding model-resource projector unit for each resource, simulates the cancellation of one existing reservation at a time, until enough available time is produced with which to construct a feasible reservation for the preempting subproblem. As soon as a feasible reservation is found, the KS stops looking, cancels the preempted reservations, and makes the new assignment to satisfy the preempting subproblem.

Figure 4.16 presents a portion from a DSS execution trace illustrating the execution of the *Preemption* KS. In this situation, the current allowance for `[CLEAN]-SERVICE-GOAL-#182` is `([Thu 7:57am] - [Thu 8:10am])`, and its current rating is 72135. Two existing reservations for `Cleaning-Truck-#CLT3`, one by `<[Flight #1197/1197(Thu)D]-#0>` (with a service goal rating of 23837), the other by `<[Flight #1136/1136(Thu)D]-#0>` (28279), are collectively held for the time period `([Thu 8:03am] - [Thu 8:14am])` The cancellation of these reservations makes room for a reservation of `Cleaning-Truck-#CLT3` that allows the main CLEAN servicing task for `<[Flight #1585/1586(Thu)D]-#0>` to occur from `([Thu 8:06am] - [Thu 8:10am])`, with some initial travel time from `([Thu 8:03am] - [Thu 8:05am])`. Notice that the cancellation of the two previous reservations for `Cleaning-Truck-#CLT3` forces the refinement of the previous travel time allocation for `<[Flight #12/12(Thu)I]-#0>`, because the truck will now have to leave from a different location when reporting to its next assignment. After the preemption assignment is made, the service goals for the canceled reservations are officially rerated, two related CLEAN service goals and their stimulated reservation-securing KSAs are re-rated, and finally the service goals for the two canceled reservations are reactivated.

Computationally, the *Preemption* KS is more expensive than the *Assignment* KS. It also incurs additional costs resulting from the cancellation of existing reservations, because all canceled reservations must be re-secured. The satisfaction of the preempting subproblem, nevertheless, has a localized effect that is identical to that of the *Assignment* KS, because the new reservation is kept within the time range specified by the allowance of the service goal corresponding to the preempting subproblem. Conflicts with previous decisions will never be introduced. From the perspective of the resource schedule, the cost of the *Preemption* KS may be higher, due to the introduction of fragmentation into the schedule for the preempted resource, because a preempted reservation may have started earlier than the one preempting it.

The *Preemption* KS provides a means for the scheduler to achieve its goal of minimizing the constraints imposed by its scheduling decisions. In case the scheduler's previous determination of the urgency of its subproblems was either misdirected or under-informed in some way

```
--------------------------------- KSA 476 ---------------------------------
KS Invocation ---------> STANDARD-PREEMPTION ([CLEAN]-SERVICE-GOAL-#182)
Canceled Reservation --> [Thu 5:50am] Cleaning-Truck-#CLT3 no longer assigned to
                         [Task Node: <Task: CLEAN> <[Flight #1197/1197(Thu)D]-#0>].
Canceled Reservation --> [Thu 5:50am] Cleaning-Truck-#CLT3 no longer assigned to
                         [Task Node: <Task: CLEAN> <[Flight #1136/1136(Thu)D]-#0>].
Succeeding Reservation Refined --> [Thu 5:50am] Resized GOTO-GATE in
                         [Task Node: <Task: CLEAN> <[Flight #12/12(Thu)I]-#0>]
                         on Cleaning-Truck-#CLT3 changed to
                         ([Thu 1:39pm] [Thu 1:40pm]) from ([Thu 1:38pm] [Thu 1:40pm]).
Resource Assignment ---> [Thu 5:50am] Cleaning-Truck-#CLT3 assigned to
                         [Task Node: <Task: CLEAN> <[Flight #1585/1586(Thu)D]-#0>]
                         ([Thu 8:03am] - [Thu 8:10am] ([Thu 8:06am] - [Thu 8:10am])).
Rerated Service Goal --> [Thu 5:50am] <[Flight #1136/1136(Thu)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#96 (CLEANING-TRUCK)
                         from {1413} to {23837}.
Rerated Service Goal --> [Thu 5:50am] <[Flight #1197/1197(Thu)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#124 (CLEANING-TRUCK)
                         from {669} to {28279}.
Rerated KSA -----------> [Thu 5:50am] <KSA-763 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [CLEAN]-SERVICE-GOAL-#196
                          CLEANING-TRUCK STANDARD-ASSIGNMENT)
                         from {64196} to {48216}.
Rerated Service Goal --> [Thu 5:50am] <[Flight #410/410(Thu)I]-#0>
                         [CLEAN]-SERVICE-GOAL-#196 (CLEANING-TRUCK)
                         from {64196} to {48216}.
Rerated KSA -----------> [Thu 5:50am] <KSA-755 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [CLEAN]-SERVICE-GOAL-#189
                          CLEANING-TRUCK STANDARD-ASSIGNMENT)
                         from {27442} to {20936}.
Rerated Service Goal --> [Thu 5:50am] <[Flight #1448/1448(Thu)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#189 (CLEANING-TRUCK)
                         from {27442} to {20936}.
Invoked Precondition --> KS SELECT-RESOURCE-SECURING-KS ([CLEAN]-SERVICE-GOAL-#124)
KS Activation ---------> SELECT-RESOURCE-SECURING-KS ([CLEAN]-SERVICE-GOAL-#124)
Invoked Precondition --> KS SELECT-RESOURCE-SECURING-KS ([CLEAN]-SERVICE-GOAL-#96)
KS Activation ---------> SELECT-RESOURCE-SECURING-KS ([CLEAN]-SERVICE-GOAL-#96)

                                         ⋮
```

Figure 4.16. Execution Trace Showing the Execution of the *Preemption* KS.

(perhaps due to a lack of up-to-date information), the scheduler may use this opportunity to employ the *Preemption* KS to attempt to correct for its previous miscalculations by adjusting its queue of pending subproblems. This adjustment involves exchanging the places of an urgent unsatisfied subproblem with those of a number of less urgent, but previously satisfied subproblems, in what amounts to a form of sacrifice. The checks performed by the KS in the process of building its list of candidate resource and reservation pairs are designed to ensure that the subproblems whose reservations are preempted will be easier for the scheduler to re-satisfy than the preempting subproblem.

4.2.4.4 The Right Shift Knowledge Source

The *Right Shift* KS serves as a kind of last-chance method for securing a resource reservation. A *right shift* describes the movement of an existing reservation from one point in the schedule to another point later in time (downstream). The goal of the *Right Shift* KS is to find the best possible reservation for a subproblem, without guaranteeing that the resulting reservation will remain within the bounds of a service goal's allowance. This approach, however, has the additional impact of possibly forcing temporally constrained downstream reservations to be right-shifted themselves to make room for a delayed reservation. The introduction of a right-shifted reservation therefore does not result in a completely localized change in the schedule. Instead, an entire group of previously satisfied subproblems may be affected, thereby forcing the system to handle the propagation of a potentially significant number of constraints throughout the developing schedule. While the *Right Shift* KS takes a clear step forward in the process of constructing the schedule, it is an expensive step in terms of the amount of processing and constraint propagation associated with its decisions.

The *Right Shift* KS carries out its search process by first assembling information about the earliest possible reservation that may be secured using each candidate resource, that is, all resources of the designated class that satisfy the relevant constraints. Task shift preference is ignored, owing to the fact that all previous attempts to find a reservation within the allowance for the service goal have failed. This entire process is performed using the resource projector units associated with each candidate resource, so that the exact extent of each right shift can be established by the KS before a final decision must be made.

The list of candidate resource and reservation pairs is incrementally constructed by selecting each candidate resource projector unit and determining the earliest time at which a reservation for the subproblem may be secured using the corresponding resource. The KS must then determine the amount of right shifting that will be required for any existing downstream reservations as the result of the right-shifted subproblem and any other right-shifted downstream reservations already determined. This is often an expensive search process, because each delayed candidate reservation may induce a significant amount of downstream right-shifting, which in turn may push back the scheduled completion time for both the original subproblem's order and possibly some of the orders connected by right-shifted inter-order subproblems. Fortunately, the available slack in this situation helps to limit this cost. Changes to the scheduled release and completion times of an order may cause additional right-shifting of previously unaffected reservations if there are time bound constraints on those subproblems that are based on these times.

The process of determining the full impact of a single right-shift on an existing reservation requires the *Right Shift* KS to check and see whether the reservation may be performed by the same resource at some time (the earliest possible) downstream in the schedule. For subproblems with parent (aggregate) subproblems providing work areas whose reservation must encompass all child subtask activity, the KS must check to see that the reservations of the parent subproblems can be extended to handle the delay forced by the right-shifting of any of its children. For subproblems that have not yet been satisfied, their service goal's allowances may be shrunk in the process of absorbing delay from preceding (upstream) subproblems. Tasks that do not require the use of resources are shifted as necessary. After the impact of each delayed candidate reservation has been determined (again using the projector units), the KS has a list of candidate resource and reservation pairs that includes the cost of introducing each potential assignment, in terms of how many existing reservations will have to be right-shifted or extended, and how much the release and completion times for all affected orders will be changed.

The *Right Shift* KS selects a delayed reservation from the list of candidate resource and reservation pairs that introduces the least amount of delay for all affected orders. The KS then executes the actual right-shift, and DSS handles the required propagation activities.

Figure 4.17 presents a portion from a DSS execution trace illustrating the execution of the *Right Shift* KS. In this situation, the allowance for `[LOAD-BAGGAGE]-SERVICE-GOAL-#71` is `([Mon 5:01pm] - [Mon 5:25pm])`. All previous attempts to secure a baggage truck reservation within this period of time have been unsuccessful. Under the current circumstances, the earliest possible reservation that also limits the amount of delay incurred by `<[Flight #243/243(Mon)D]-#0>` is for `Baggage-Truck-#BT-BO2` from `([Mon 5:26pm] - [Mon 5:37pm])`, with setup activity occurring from `([Mon 5:07pm] - [Mon 5:25pm])`. To make this delayed reservation, the completion dime for `<[Flight #243/243(Mon)D]-#0>` will be pushed back 12 minutes, and the previous reservation for `Gate-#F2` (where the flight will be serviced) will be extended by 12 minutes as well. The extension of the parent TURNAROUND ACTIVITY task reservation also provides additional slack to the unsatisfied SERVICE task. Finally, the altered `[SERVICE]-SERVICE-GOAL-#69` and its pending *Assignment* KSA are re-rated.

The execution cost of the *Right Shift* KS may be quite expensive, especially when the schedule is densely packed and nearly complete. A single delayed reservation may affect a very large set of existing reservations. From the scheduler's perspective, the cost of propagating the constraint changes caused by right shifting is also expensive in terms of the limitation on flexibility that it imposes by using up valuable slack time within the affected orders to absorb delay. This is a KS that should only be used after other less drastic and expensive methods have failed.

4.2.4.5 Relaxed Reservation-Securing Knowledge Sources

The standard reservation-securing KSs are all restricted to consider only those candidate resources that satisfy all existing contextual constraints on a subproblem. They will therefore not consider any resource that is incompatible with all other contextually constraining reservations.

In the AGSS domain, the servicing of an aircraft takes place at a specific gate. All necessary servicing vehicles must report to the gate's work area in order to perform their assigned servicing tasks. Each reservation for one of these mobile resources constrains the choice of a gate, by the amount of time that has been allocated to the resource for traveling from its previous location to the gate. If some of the mobile resources have already been reserved at the time a gate reservation is sought, then the contextual constraints imposed by those reservations may limit

```
-------------------------------- KSA 186 --------------------------------
KS Invocation ---------> STANDARD-RIGHT-SHIFT ([LOAD-BAGGAGE]-SERVICE-GOAL-#71)
Modified Goal ---------> [Mon 1:45pm] [SERVICE]-SERVICE-GOAL-#69 (SERVICE-TRUCK)
                         allowance now ([Mon 4:51pm] [Mon 5:37pm]);
                         was ([Mon 4:51pm] [Mon 5:25pm]) <[Flight #243/243(Mon)D]-#0>.
Modified Due Date -----> [Mon 1:45pm] <[Flight #243/243(Mon)D]-#0>
                         due [Mon 5:42pm]; originally [Mon 5:30];
                         now running [12 minutes] LATE  <Penalty: 12>.
Resource Assignment ---> [Mon 1:45pm] Baggage-Truck-#BT-BO2 assigned to
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #243/243(Mon)D]-#0>]
                         ([Mon 5:07pm] - [Mon 5:37pm] ([Mon 5:26pm] - [Mon 5:37pm])).
Modified Reservation --> [Mon 1:45pm] [Task Node: <Task: TURNAROUND-ACTIVITY>
                         <[Flight #243/243(Mon)D]-#0>] Gate-#F2
                         ([Mon 4:42pm] - [Mon 5:41pm] ([Mon 4:47pm] - [Mon 5:41pm]));
                         was ([Mon 4:42] - [Mon 5:29pm] ([Mon 4:47pm] - [Mon 5:29pm])).
Rerated KSA -----------> [Mon 1:45pm] <KSA-185 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [SERVICE]-SERVICE-GOAL-#69
                          SERVICE-TRUCK STANDARD-ASSIGNMENT)
                         from {114} to {113}.
Rerated Service Goal --> [Mon 1:45pm] <[Flight #243/243(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#69 (SERVICE-TRUCK)
                         from {114} to {113}.
                                          ⋮
```

Figure 4.17. Execution Trace Showing the Execution of the *Right Shift* KS.

the pool of gates that may be considered for the reservation. While our conservative worst-case approach to securing mobile resources helps to limit the occurrence of such situations, the nullification of a previously secured stationary resource reservation, upon which exact mobile resource reservations have been constructed, can severely limit the options for finding a new stationary resource. Similarly, mobile resources secured to perform servicing at a gate may have succeeding reservations that require them to be at a particular location at a designated future time. Again, the selection of the gate at which such resources are to be used must allow enough time for all currently secured mobile resources to get from the gate to whatever locations are required by their succeeding reservations.

When the standard reservation-securing KSs encounter these situations and are unable to avoid violating the contextual constraints, then consideration of a candidate resource is halted. If no candidate resource is found that satisfies the existing contextual constraints on a subproblem, then each of the standard KSs will fail to produce a reservation. For this reason, we have developed a second set of reservation-securing KSs that perform the necessary relaxation of any violated contextual constraints by either modifying or canceling any offending reservations in order to satisfy the present subproblem.

Figure 4.18 illustrates an example of a situation where the contextual constraints imposed by one reservation preclude the selection of another reservation, thereby warranting the use of a relaxed reservation-securing KS. In this case, a stationary resource reservation is being sought which will provide a destination location for two previously secured mobile resources. Note that the reservation for MOBILE RESOURCE 1 includes enough travel time to permit eight feasible destination locations (each corresponding to a stationary resource). The reservation for MOBILE RESOURCE 2, however, is small enough to deny any of those locations from being selected. The reservation for MOBILE RESOURCE 2 is therefore an *offending* reservation. The approach taken by the relaxed reservation-securing KSs is to make a selection that ignores all offending reservations, and to then select the best reservation among those candidates producing the fewest offenders, that permits each offending reservation to be relaxed (or canceled) to the point where it no longer precludes the candidate reservation. Referring back to Figure 4.18, the solution would be to try to either relax or cancel the existing reservation for MOBILE RESOURCE 2 to permit the selection of one of the eight feasible destination location stationary resources.

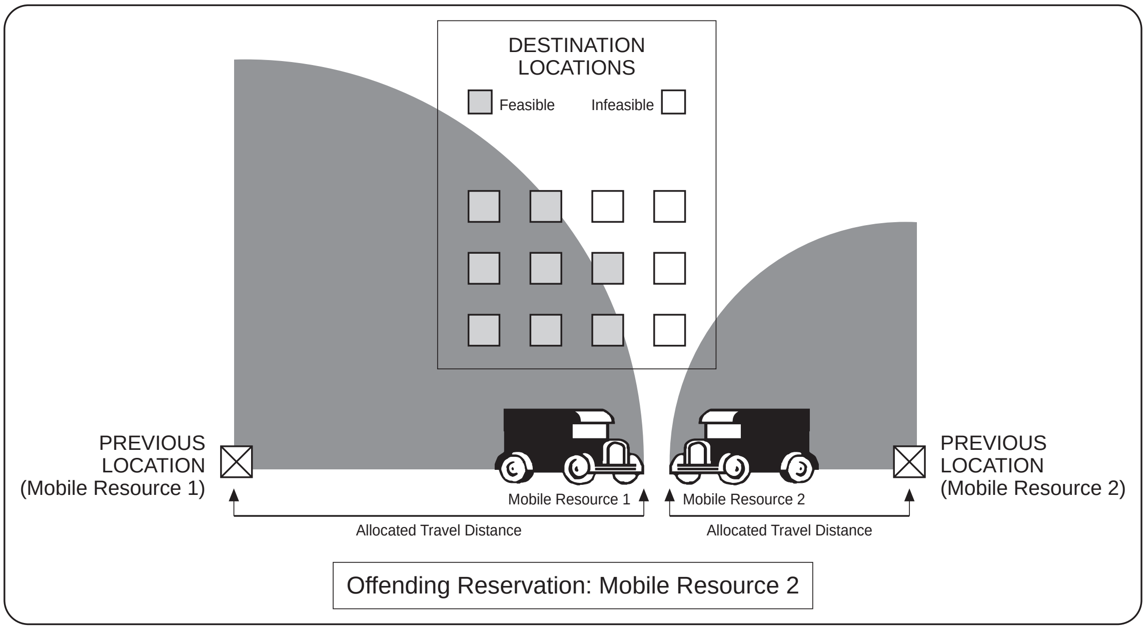

DESTINATION LOCATIONS
Feasible
Infeasible
PREVIOUS LOCATION (Mobile Resource 1)
PREVIOUS LOCATION (Mobile Resource 2)
Mobile Resource 1
Mobile Resource 2
Allocated Travel Distance
Allocated Travel Distance
Offending Reservation: Mobile Resource 2

Figure 4.18. A Situation Requiring the Use of Relaxation Techniques to Secure a Stationary Resource Reservation

For the *Relaxed Assignment* KS, the search process runs similarly to that of the *Assignment* KS, except that the contextual constraint check is not initially performed on the candidate resources. Instead, these constraints are ignored until the KS has assembled its list of candidate resource and reservation pairs. The list is first sorted so that those reservations involving the fewest number of offenders are moved to the top. It is then (stably) sorted so that the earliest reservation with the least amount of setup time occurs first in the list. The KS then proceeds down the list, attempting to relax all of the offenders for each resource and reservation pair until success is achieved. If all of the offenders for a pair cannot be relaxed, then the pair is discarded. If all pairs are discarded, then the KS fails.

Again, as is the case with the *Assignment* KS, the relaxed version does not leave the scheduler with any newly re-stimulated and potentially difficult subproblems to solve, because it does not cancel any of the offending reservations. By relaxing the offenders, it actually restores some flexibility to the developing schedule by loosening their contextual constraints. The main additional cost incurred by this KS (when compared to its standard version) is the time devoted to determining whether all of the offenders for a particular reservation may be relaxed. This is an expensive filter that may (in the worst case) have to be applied to each candidate resource and reservation pair. The relaxation process, however, does slightly reduce the level of resource availability by enlarging previous reservations.

For the *Relaxed Preemption* KS, the list of candidate resource and reservation pairs is assembled using the same approach as in the standard *Preemption* KS. The list of candidates is then sorted so that the resources providing the earliest reservation for the preempting subproblem will be considered first. The sorted list is then processed one candidate at a time (as in the standard version) to determine how many existing reservations will have to be preempted. All current contextual constraints are ignored at this time. The KS then proceeds down the candidate list, attempting to find the first successful preemption case where all offending reservations may be relaxed. If success is achieved, the offenders are relaxed, the reservations to be preempted are canceled, and the new assignment for the preempting subproblem is made. If no candidate reservation is found that allows all of the offenders to be relaxed, then the KS fails.

The costs associated with the *Relaxed Preemption* KS are similar to those for the *Relaxed Assignment* KS. An additional expense arises from the increased computation required to perform the preemption calculations.

The *Relaxed Right Shift* KS serves as the absolute last-chance method for finding a resource reservation for a subproblem after every other available method has been exhausted. This KS is basically designed to find a reservation at any cost. Except for ignoring any existing relevant contextual constraints, it behaves like the standard *Right Shift* KS up to the point of determining the penalties of introducing a delayed reservation. The difference arises when determining the need to shift or modify existing downstream reservations. If an existing reservation cannot be shifted to make room for a delayed reservation, it is added to the list of offenders.

Once the list of candidate resource and reservation pairs has been assembled, the KS then processes the list (ordering it as in the standard method). The offenders are then tested to see whether they may be relaxed to permit the potential delayed reservation. This is where an important feature of the *Relaxed Right Shift* KS comes into play. When an offender cannot be relaxed, its corresponding reservation is *canceled*. As a result, this KS should never fail to find a reservation, as long as an instance of the proper resource class exists. It may, however, insert a number of potentially difficult-to-solve offending subproblems back into the scheduling process. In terms of computational expense, this KS should only have to process the first of the candidates, thereby avoiding a complete traversal through the list, and placing it at roughly the same cost as the standard *Right Shift* version. The major impact of its execution is the potential cancellation of previously secured reservations with no regard for the level of difficulty that may be expected in the process of re-satisfying them. It should be noted, however, that any delay to the order introduced by a relaxed right shift may serve to enlarge the window of time within which the canceled reservations must be re-satisfied, meaning that they may become less difficult to solve than they originally were.

4.2.5 Re-Securing Resource Reservations

Existing resource reservations may be canceled either as part of the actions taken during the execution of certain KSs, as the result of the unexpected breakdown of previously secured resources. In the case of a reservation cancellation, the affected subproblem is re-stimulated by the reactivation of its corresponding service goal. The reactivation of the service goal restarts the process of attempting to secure a resource reservation for the subproblem by triggering the (previously described) *Select Reservation-Securing Method* KS. Upon being reactivated, a service goal is re-rated before triggering any of the reservation-securing KSs.

The conflicts introduced by the cancellation of a previous resource reservation should not lead to an over-reaction on the part of the scheduler. In DSS, the affected subproblem is analyzed on its own merits, and its urgency is placed within the context of the urgency of all other existing unsatisfied subproblems, so that it may be properly inserted into the current agenda. A situation that had previously caused a particular subproblem to be addressed and solved early on in the scheduling process may no longer exist, thereby lessening the urgency with which it must be re-solved. Other subproblems may have become more urgent in the meantime. We allow the multiple-perspective view of the scheduling process, as implemented by the service goal rating mechanism, to determine the course of action for the scheduler to take in response to a reservation cancellation, just as it does under ordinary circumstances.

4.2.6 Reservation Refinement

One of the ways that DSS maintains flexibility throughout the scheduling process is to equip the reservation-securing KSs with the ability to produce reservations for mobile resources that include enough travel time for the secured resource to get from its previous location to its required destination whether or not either of those locations has been determined. When a reservation of this type is made, the subproblem for which the resource has been secured is marked as being *unrefined*, indicating that possibly more than enough travel time has been allocated as part of the reservation. This kind of situation may occur in any scheduling domain involving mobile resources and reserved work areas. If a mobile resource is secured before the work area to which it must report has been determined, then the inclusion of extra travel time as part of the mobile resource reservation will preserve for a later time the unconstrained choice of work areas.

The flexibility built into the schedule in this way is maintained as long as any uncertainty exists regarding an unrefined reservation. At any point when the context of an unrefined reservation is further established as the result of other related scheduling decisions, the refinement mechanism is employed to go through the entire set of related subproblems and determine whether any of the unrefined reservations among them may be refined and compacted to free up resource time and eliminate unnecessary fragmentation in the affected resource schedules. The task structures defined as part of the domain description provide the information that DSS uses to determine when the opportunity for refinement exists.

The reservation refinement process may be described as follows, referring back to our discussion of how resource reservations are secured (Section 4.2.4). In uncertain situations, unrefined reservations of the kind shown in Figure 4.12 are constructed for mobile resources. At the point when both the selected destination location for a mobile resource *and* its location prior to an assignment have both been determined, the reservation refinement mechanism converts an unrefined reservation into a refined reservation of the kind shown in Figure 4.13, thereby freeing up previously maintained but now unneeded flexibility for use in satisfying other resource requests.

We have implemented the *Reservation Refinement* KS to provide DSS with the means to periodically stop and make use of its maintained flexibility. This KS is triggered as the result of the activation of the refinement goal corresponding to a resource-requiring product task. The activation process works in a top-down fashion, and is triggered whenever the service goal for an aggregate task is satisfied, at which point all of the refinement goals for the subproblems representing satisfied but unrefined child product tasks (both intra- and inter-order), and those of their children, are activated as well. A single KSA is instantiated for each relevant unrefined subproblem.

The *Reservation Refinement* KS can be activated as long as there is slack to be exploited. It will attempt to shift a reservation to the left (upstream) or to the right (downstream), depending on the shift preference of the task, the current level of contention for the resource involved during the affected time period, the current tardiness for the order involved, and the amount of slack currently available. If a task has an early shift preference, or there is high resource contention and the order involved is currently tardy, a left-shift is attempted. Otherwise, a right shift is attempted. A left shift can be attempted unless there is no slack available to the left of the task, or there is high resource contention and more slack to the left than the right. That is, a left shift is attempted, where possible, unless it would decrease the size of a valuable block of slack that could be used for some other purpose. Similarly, a right shift can be attempted unless there is no slack available to the right of the task, or there is high resource contention and more slack to the right than the left. As part of the refinement process, the KS recalculates the actual amount of processing time required for a reservation given the new context within which the reservation exists. The result of the execution of this KS is often an improvement in the resource utilization of the affected resource class, achieved by either reducing some amount of fragmentation built into the schedule by a previous worst-case decision, or at the very least, a more efficient use of the resource due to a shortening of a reservation's overall processing duration.

Figure 4.19 presents a portion from a DSS execution trace illustrating the execution of the *Reservation Refinement* KS. In this situation, the satisfaction of `[TURNAROUND-ACTIVITY]-SERVICE-GOAL-#8` during the invocation of KSA 29 (*Assignment*) is achieved by a reservation of the stationary resource `Gate-#G1`. As an aggregate, the TURNAROUND ACTIVITY task provides potentially important contextual information for its child reservations. Now that a resource has been secured for this task, any previously secured but still unrefined mobile resource reservations may be refinable. In Figure 4.19, it turns out that only two previously secured and still unrefined child reservations exist, those for the UNLOAD BAGGAGE and BAGGAGE TRANSFER tasks. Two *Reservation Refinement* KSAs are therefore instantiated, one for each task. The invocation of the first KSA leads to a refinement of the UNLOAD BAGGAGE reservation that clips 8 minutes from the previous unrefined reservation of `Baggage-Truck-#BT-F2`. The invocation of the second KSA, however, produces no change in the existing BAGGAGE TRANSFER reservation, because the remaining refinable portion of the BAGGAGE TRANSFER reservation does not involve the initial travel activity for the baggage truck. The refinement in this situation actually involves a subtask that does not directly require a resource (the baggage truck), and must instead be handled as a special case (defined in the following section).

The need to refine the processing durations of tasks that do not require resources can occur as the result of the introduction of a stationary resource reservation. Just as mobile resource reservations are often dependent on stationary resources providing work area locations to and from which they must report, non-resource-requiring tasks may also be dependent on these stationary resources. For example, in the AGSS domain, the amount of travel time allocated for the PASSENGER TRANSFER operation between two connecting flights is determined by the two gates used to service the flights, even though no resource is required for performing the actual

```
--------------------------------- KSA 29 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#8)
Resource Assignment ---> [Mon 1:36pm] Gate-#G1 assigned to
                         [Task Node: <Task: TURNAROUND-ACTIVITY>
                                     <[Flight #1447/1447(Mon)D]-#0>]
                         ([Mon 4:32pm] - [Mon 5:19pm] ([Mon 4:37pm] - [Mon 5:19pm])).
Invoked Precondition --> KS RESERVATION-REFINEMENT
                         ([UNLOAD-BAGGAGE]-REFINE-GOAL-#63)
KS Activation ---------> RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#63)
Invoked Precondition --> KS RESERVATION-REFINEMENT
                         ([BAGGAGE-TRANSFER]-REFINE-GOAL-#55)
KS Activation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#55)
--------------------------------- KSA 30 ---------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#63)
Refined Reservation ---> [Mon 1:36pm] Baggage-Truck-#BT-F2 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE>
                          <[Flight #1447/1447(Mon)D]-#0>]
                         ([Mon 4:39] - [Mon 4:52] ([Mon 4:40pm] - [Mon 4:45pm]));
                         was ([Mon 4:33pm] - [Mon 4:54pm] ([Mon 4:40pm] - [Mon 4:45pm])).
--------------------------------- KSA 31 ---------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#55)

                                      ⋮
```

Figure 4.19. Execution Trace Showing the Execution of the *Reservation Refinement* KS.

passenger transfer. In the case of the BAGGAGE TRANSFER task refinement mentioned above, the task that is affected by the assignment of the gate is actually the non-resource-requiring TRANSFER X BAGGAGE subtask, which serves to indicate the portion of the overall BAGGAGE TRANSFER task that is spent actually transferring baggage. Again, its duration has no effect on the amount of time required to permit the baggage truck to report to its originating service location.

These relationships are identified using the SPECIAL-REFINEMENT-CASE? flag, which indicates when the processing duration for a non-resource-requiring task depends on the processing duration of some other task. When such a relationship exists, an automatic refinement calculation is initiated whenever a reservation upon which the dependent task depends is introduced. In the case of the AGSS domain PASSENGER TRANSFER and BAGGAGE TRANSFER tasks, both are refined at the time when both gates involved in servicing the connected flights have been secured (for example, during the invocation of KSA 29 in Figure 4.19).

4.2.7 Processing Resource Failures

A primary concern in the design of the DSS scheduling approach has been the desire to produce a completely reactive decision-making process. When unexpected events occur within the environment, they are addressed according to their relative urgency within the context of the set of pending scheduling activities. This avoids the initiation of a special process for handling such events that assigns a high priority to the resolution of their resulting conflicts simply because of their unexpected nature. As a result, each such event can be assessed individually and then placed into its proper order in the queue of pending scheduling activities, thereby avoiding an automatic disruption to the existing order.

In Figure 4.20, a DSS execution trace is presented that begins with the failure of resource Baggage-Truck-#BT-BO1. Upon receiving notification of the failure, DSS triggers the *Process Resource Failure* KS, which is responsible for determining the impact of the failure on the existing schedule. Because a resource failure may have a significant impact on the schedule, the *Process Resource Failure* KS is executed immediately. Upon invocation, it cancels all existing pending reservations for the failed resource, including any reservation currently executing, in which case the task will have to be completely re-executed by some other resource. As a result of such cancelations, each service goal corresponding to an affected task is reactivated, and DSS restarts the process of attempting to achieve its satisfaction. The urgency of each reactivated subproblem is therefore based solely on the current state of problem-solving, and not on some predefined static ordering. In Figure 4.20, the cancellation of the LOAD BAGGAGE reservation for <[Flight #370/370(Mon)D]-#0> results in the reactivation of [LOAD-BAGGAGE]-SERVICE-GOAL-#63, the satisfaction of which will be achieved at some appropriate time in the future.

⋮

```
Resource Breakdown ----> [Mon 4:40pm] Baggage-Truck-#BT-BO1 now unavailable.
Invoked Precondition --> KS PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
KS Activation ---------> PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
------------------------------ KSA 188 -------------------------------
KS Invocation ---------> PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
Canceled Reservation --> [Mon 4:40pm] Baggage-Truck-#BT-BO1 no longer assigned to
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #370/370(Mon)D]-#0>].
Rerated Service Goal --> [Mon 4:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {11960} to {233}.
Invoked Precondition --> KS SELECT-RESOURCE-SECURING-KS ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> SELECT-RESOURCE-SECURING-KS ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
```

⋮

Figure 4.20. Execution Trace Showing the Execution of the *Process Resource Failure* KS.

In a situation involving a resource failure that affects a number of existing reservations, it is possible that the current conditions within the environment may be such that some of those reservations—for example, those with immediately pending operation times—will require immediate addressing, while others, being less urgent, may be able to wait until later.

DSS can accommodate unexpected events into its basic control scheme in such a way that every scheduling task is performed as necessary according to its urgency, and independently of its specific origin. It is thus able to move seamlessly between predictive and reactive scheduling tasks, and thereby achieve a high degree of flexibility that permits it to react effectively to a dynamic environment.

4.3 A Summary

The multi-faceted scheduling approach of DSS addresses a number of important issues of flexibility. The least-commitment preservation of slack time acts to preserve future scheduling options, the fine-grained opportunism enables quick and effective reaction to dynamic environments, and the consultation of multiple, relevant scheduling perspectives produces an informed decision-making process that can detect developing conflicts before they become more serious.

In the following chapter, we present the results of a number of experiments designed to illustrate the performance benefits and generic capabilities of the DSS scheduling approach. In a comparison against ISIS and the initial OPIS system using the common benchmarking data presented in [Chiang *et al.*, 1990], we demonstrate better performance by DSS in terms of producing lower average tardiness costs, and lower percentages of tardy orders. In addition, we demonstrate the ability of our least-commitment decision-making process to produce average work-in-process times that remain well within the average target lead times, despite the preservation of slack time within those developing schedules.

A second set of experiments uses a reactive, dispatch situation to demonstrate the benefits of our multiple-perspective urgency-determination heuristic, and the adaptability of our fine-grained opportunistic problem-solving approach, to both decrease tardiness and increase computational efficiency under changing environmental conditions. In these experiments, we contrast the results obtained by our multiple-perspective heuristic with those of a number of classical scheduling heuristics implemented within the DSS framework. We then continue this comparison of heuristics under conditions that include unexpected resource failures, to model more closely the dynamics of a real-world environment.

The final set of experiments evaluate further the ability of the DSS multiple-perspective urgency-determination heuristic to understand fully the various textures of the current state of problem-solving, and the ability of its least-commitment decision-making approach to preserve sufficient flexibility for accommodating unexpected developments.

CHAPTER 5

EXPERIMENTAL EVALUATION OF DSS

Our intent throughout this research has been to demonstrate that the performance of a scheduling system can be enhanced by preserving and utilizing the flexibility available within the problem, and considering a broad range of perspectives on the state of problem solving. In this chapter, we rate the performance of DSS according to both the quality of its schedules and the computational efficiency of its approach. Furthermore, we evaluate the effectiveness of DSS within the context of dynamic environments that produce random unexpected events, thereby demonstrating its ability to perform well in situations where the entire collection of orders is not known in advance, and when resources are either in short supply, or are susceptible to breakdowns.

We begin in Section 5.1, with a discussion of the primary performance objectives for DSS, and continue in Section 5.2 with a description of the various measures we have used to evaluate the success of our approach. To prepare for the discussions of the actual experiments, Section 5.3 provides descriptions of the set of classical scheduling heuristics we have implemented within the DSS scheduling mechanism for the purpose of comparing a variety of heuristic approaches. Following that, Sections 5.4 and 5.5 introduce the two DSS-based scheduling application systems that will be used to evaluate the performance of the basic DSS scheduling approach across substantially different domains. The evaluation of the experiments and results is provided in Section 5.6. In that section, we analyze the results of three different experiments using two DSS-based scheduling application systems. Finally, we summarize our results in Section 5.7.

5.1 Primary Objectives

While schedule quality can be judged in many ways, the focus is generally on the degree to which order due dates are met. DSS focuses on the minimization of tardiness, in the form of low average tardiness costs per order, and low percentages of tardy orders. In terms of processing, the idea behind the design of DSS has been to produce a scheduling system that understands fully the current state of problem solving at all times, and is therefore able to react quickly and appropriately to unexpected changes within its environment. The ability with which a scheduler is able to understand the state of problem solving can be evaluated by analyzing its problem-solving behavior. The use of fewer and more efficient decision-making steps to produce a schedule is evidence of a keen understanding of the entire problem. DSS attempts to minimize the total number of decision-making steps, and arrange in which those steps are taken so that the majority of its decisions can be made quickly, and without causing too much costly disruption to the existing schedule.

In the following two sections, we discuss each of the primary performance objectives in detail.

5.1.1 Minimizing Tardiness

The primary *schedule quality* objective incorporated into DSS is the minimization of the degree to which order due dates are exceeded. This objective manifests itself in the minimization of the average tardiness cost per order, and the incidence of tardiness across the set of all orders. Schedules with low average tardiness costs and few overdue orders incur fewer penalties resulting from missed due dates.

The tardiness cost for an order is determined by combining its scheduled tardiness with its application-specific priority. Let $\mathcal{O}$ indicate the set of all orders O_i, and let D_i be the due date, and C_i the scheduled completion time, for each order O_i. We then define the average tardiness cost per order for an entire schedule as follows:

$$\frac{\sum_{O_i \in \mathcal{O}} \max(\,0,\ (C_i - D_i)\,)}{|\mathcal{O}|}$$

For the purpose of determining tardiness cost, orders that are completed prior to their due dates are still considered to be *on time*, thereby incurring a tardiness cost of zero. Remember that DSS makes no attempt to minimize order flow times or schedule makespan.[1]

The incidence of tardiness across the set of all orders is measured as the fraction of tardy orders within the set, as follows:

$$\frac{|\{\,O_i \text{ s.t. } C_i > D_i\}|}{|\mathcal{O}|}$$

To achieve the goal of limiting tardiness across the set of all orders, DSS considers the anticipated effect of its scheduling decisions on the currently scheduled completion times for all orders involved. Some of the reservation-securing KSs can introduce delays for one or more jobs in the process of assigning a resource to a particular operation at a certain time. DSS is designed to avoid the introduction of tardiness by organizing its variable-ordering decision-making process to address over-constrained subproblems at the time when the broadest range of options are available for their solution. In addition, its value-ordering heuristics are designed to favor resource reservations that avoid the introduction of new delays or the further extension of existing ones.

5.1.2 Maximizing Computational Efficiency

The primary *computational* objective is to provide DSS with the ability to react efficiently to developments in a changing environment so that costly backtracking and constraint-relaxation activities can be avoided. We can measure this ability by considering the number of KSAs that must be invoked in the course of producing a schedule, and the specific kinds of KSs used to secure all required resource reservations.

We have discussed how the order in which scheduling subproblems are solved affects the overall quality of the resulting schedule. To limit the computational expense involved in producing a schedule, the order in which scheduling subproblems are addressed must be

[1] This is a possible direction for future work, specifically in trying to balance the need for schedule compaction with the need to preserve schedule flexibility, according to the current state of problem solving.

frequently and properly reorganized according to the changing state of problem solving. A significant portion of the computational expense incurred during the scheduling process results from the selection *and* execution of the various reservation-securing KSs, and the propagation of the effects of their scheduling decisions. As a subproblem becomes more tightly constrained, the kind of KS required for its solution becomes increasingly expensive, as do the propagation activities it triggers. In addition, the more frequently that backtracking is required as a result of changing environmental conditions or earlier scheduling decisions, the more often that previously solved subproblems will have to be re-solved at potentially greater computational expense.

Metrics recorded by GBB are used to provide information on the total number of KSAs executed during the course of problem solving. These numbers help indicate the total amount of computational effort that DSS devotes to the process of solving an RCSP. A number of additional metrics can be calculated from an analysis of satisfaction records maintained by all created service goals. Each satisfaction record maintains a history of the size of the goal's allowance, and the reservations and means by which those reservations were secured, over the lifetime of the goal. By analyzing this history, the number of times a particular subproblem was solved, and at what computational cost, can be determined. Service goals that are re-satisfied on numerous occasions throughout the scheduling process indicate a high level of backtracking involving their particular subproblems, and may suggest an inappropriate placement of the subproblem in the scheduler's queue of scheduling activities. Service goals that are solved only once during the scheduling process indicate that their placement within the scheduling activity queue was entirely appropriate.

5.2 Problem-Solving Metrics for Evaluating DSS

There are six basic problem-solving metrics that are used to evaluate the performance of DSS, in terms of both schedule quality and computational efficiency. These metrics appear in the graphs that accompany the experiments with which DSS has been evaluated. We describe each of them below.

- **Average Tardiness Cost per Order**

  The *Average Tardiness Cost per Order* measure is considered in every experiment. It indicates one of the ways in which tardiness manifests itself in a completed schedule. While *tardiness* is measured in units of time, *tardiness cost* is measured in cost units determined by the particular problem domain.

- **Percentage of Tardy Orders**

  The *Percentage of Tardy Orders* measure is also considered in every experiment. It indicates another way in which tardiness manifests itself in a completed schedule. More importantly, it can be used to ensure that a low average tardiness cost per order is not achieved by spreading tardiness across many orders. Tardy order percentage is charted as the fraction of tardy orders within a complete set of orders.

- **Average Work-in-Process Time per Order**

  The *Average Work-in-Process Time per Order* measure is considered as a means of ensuring that DSS does not sacrifice WIP times by failing to adequately compact its schedules.

This metric is used in the evaluation of the TCP job-shop application (described (below) in Section 5.4) to indicate that reasonable WIP times do result from DSS's scheduling process. Average work-in-process times are measured in units of time dictated by the particular problem domain.

- **Total Number of KSA Invocations**

  The *Total Number of KSA Invocations* measure provides important information about the amount of decision-making effort that is devoted to a particular RCSP. The process of reserving a resource for an operation requires a certain amount of KS activity, which can be measured in terms of the invocation of KSAs. Each KSA invocation represents a fine-grained decision point. Higher numbers of invocations can also indicate both ineffective variable-ordering (because many options were explored to solve subproblems) and a high level of backtracking (because many subproblems had to be re-solved).

- **Percentage of Standard *Assignment* Reservations**

  The *Percentage of Standard Assignment Reservations* measure provides important additional information about the specific nature of the problem-solving efforts required to solve a particular RCSP. The standard *Assignment* KS introduces the least number of constraints into the existing state of problem solving, and requires no constraint relaxation. A higher frequency of subproblem satisfaction using the standard *Assignment* is thus evidence of a more efficient variable-ordering process, in which the subproblems are arranged in such a way as to permit the use of the least-costly reservation-securing method for the greatest number of subproblems. Standard *Assignment* reservation percentage is charted as the fraction of all service goal satisfactions made using the standard *Assignment* KS.

- **Elapsed Processing Time**

  The *Elapsed Processing Time* measure provides the final view of DSS's computational efficiency. While the total number of KSA invocations indicates one measure of scheduling effort, it ignores any associated activities that must be performed as part of the variable-ordering process. Because of DSS's broadly informed service goal urgency-rating function, a significant amount of processing effort can be required. The *Computational Effort* measure accounts for this possibility.[2]

5.3 Classical Scheduling Heuristics Implemented in DSS

For the purpose of evaluation, DSS is equipped with a variety of rating schemes for guiding its variable-ordering decision-making process. These rating schemes control the process of ordering the scheduling subproblems that must be addressed by the KSs responsible for creating (and occasionally modifying or canceling) resource reservations. By selecting different rating schemes, we are able to implement any of a group of classical scheduling heuristics, in addition to our own multiple-perspective approach (DSS(MPH)). The classical heuristics employ a limited analysis of the current state of problem solving, and while they therefore tend to require

[2] Computational effort is measured in terms of elapsed processing time (in minutes) on a Digital Alpha 3000 workstation running Harlequin Lispworks (version 3.1). All appropriate trademark disclaimers apply. Very little effort has been made to optimize the code, leaving the potential to obtain some performance gains.

less processing time, they generally pay the price in the form of decreased schedule quality and increased incidence of backtracking and relaxation. The DSS(MPH) heuristic performs an in-depth analysis of both existing and anticipated scheduling conditions, and rewards this extra effort by producing better quality schedules, and efficiently responding to dynamic environments.

The following heuristics from the project scheduling literature, specifically the work of [Davis and Patterson, 1975], provide us with the means to compare the behavior of DSS(MPH) with some classical heuristic approaches. Davis and Patterson ran an experiment that involved 83 different order sets, where the schedules produced using each of a set of eight scheduling heuristics were compared with optimal schedules for each problem. The heuristics were ranked according to their ability to produce schedules that were either optimal or sufficiently close. The set of control strategies that we will use for comparison is a subset of those heuristics described by Davis and Patterson.[3] Each of these heuristics are defined below. Some have been slightly modified to work within the special constraints of our dynamic RCSPs.

- **Minimum Earliest Starting Time** (DSS(MINEST))

  $$\text{DSS(MINEST)} \rightarrow \min(\ EST_{ij}\ )$$

  The DSS(MINEST) heuristic gives priority to those subproblems having the most imminent EST. The earlier the EST, the greater the priority. Whenever the EST for the subproblem changes, the rating of its corresponding service goal must be updated.

- **Minimum Late Finishing Time** (DSS(MINLFT))

  $$\text{DSS(MINLFT)} \rightarrow \min(\ LFT_{ij}\ )$$

  The DSS(MINLFT) heuristic gives priority to those subproblems having the most imminent LFT. The earlier the LFT, the greater the priority. Whenever the LFT for the subproblem changes, the rating of its corresponding service goal must be updated.

- **Shortest Operation First** (DSS(SOF))

  $$\text{DSS(SOF)} \rightarrow \min(\ d_{ij}\ )$$

  The DSS(SOF) heuristic gives priority to those subproblems having the shortest expected duration. For non-aggregate subproblems, the expected duration is calculated by taking the average of the minimum and maximum expected durations over all resource classes able to perform the task. For aggregate subproblems, the expected duration is dictated by the critical path of the child subproblems. Whenever the critical path of an aggregate's child subproblems changes, the rating of its corresponding service goal must be updated. The rating of a non-aggregate service goal does not change.

[3] Some of the heuristics in [Davis and Patterson, 1975] require finding solutions to small integer programs, a capability that has not been provided to DSS.

- **Minimum Job Slack** (DSS(MINSLK))

$$\text{DSS(MINSLK)} \rightarrow \min(\ LFT_{ij} - \max(\ EST_{ij},\ *\textsf{CURRENTTIME}*\ ) - d_{ij}\ )$$

The DSS(MINSLK) heuristic gives priority to those subproblems having the minimum amount of available slack given their allowance. The amount of slack available to a non-aggregate subproblem is determined by taking the difference between its LFT and its EST, and subtracting the expected duration of the task. The expected duration is calculated by taking the average of the minimum and maximum expected durations over all resource classes able to perform the task. The amount of slack available to an aggregate subproblem is calculated by first determining the critical path among the child subproblems. The duration of the critical path is then subtracted from the difference between the LFT and the EST. Whenever the amount of slack available to a subproblem changes, the rating of its corresponding service goal must be updated. DSS(MINSLK) is based on the notion that subproblems having little flexibility in their means of satisfaction should be handled as quickly as possible, thereby putting off until later the handling of those subproblems having more room with which to maneuver. Subproblems with little available slack represent tightly constrained subproblems that require attention before they become even more constrained, to the point that localized solutions (those solutions that do not further significantly constrain other related subproblems) are no longer available. Nearly all of the work on heuristic scheduling has recognized the importance of focusing attention as early as possible on the most tightly constrained subproblems.

- **First Come First Served** (DSS(FCFS))

$$\text{DSS(FCFS)} \rightarrow \min(\ \textsf{RECEPTIONTIME}(\ \textsf{ORDER}(\ T_{ij}\ )))\ \circ\ \min(\ EST_{ij}\ )$$

The DSS(FCFS) heuristic forces the sequential scheduling of each order, giving priority within each order to those subproblems having the most imminent ESTs. Orders are sequenced according to the time at which they are received by the scheduler. Unlike DSS(MINEST), where the modification of a subproblem's EST causes its corresponding service goal to be re-rated, the rating of a service goal under DSS(FCFS) never changes. DSS(FCFS) implements an order-based scheduling approach similar to that used in ISIS, in that each order is scheduled in its entirety by extending the schedule forward, one operation at a time, from the ready time to the due date of the order. An important feature of DSS(FCFS) is that its rating scheme all but prohibits preemption of existing reservations. Higher-rated subproblems cannot be preempted, and because DSS(FCFS) guarantees that service goal ratings monotonically decrease, satisfied subproblems will never have lower ratings than their potential preempting subproblems. The only possibility for preemption exists among subproblems within the same order having identical initial ESTs.

5.4 The TCP System

The TCP (*Turbine Component Plant*) scheduling application implements a job-shop scheduling domain involving a simplified production facility for producing three families of airplane propeller turbine blades. It is based on an application used in the evaluation of the initial OPIS system [Ow, 1986, Ow and Smith, 1988]. The objectives for this job-shop RCSP include the minimization of the average tardiness cost per order, the minimization of the average work-in-process time per order, and the minimization of the total number of setup operations required for all secured resources. Further details of the job-shop scheduling problem are discussed in Section 1.3.2.

A total of three process routings are defined for this domain. Six different products (two in each blade family) can be produced. Two of the process routings consist of six sequential resource-requiring operations. The remaining process routing consists of three sequential resource-requiring operations. The second operation in each job can be performed by one of two different types of resources (depending on the blade family involved). There is no intra-order parallelism in the production process, and there are no aggregate tasks of the kind described in Section 3.4.2.2.

Figure 5.1 provides a visual description of the three defined process routings, in addition to a basic factory layout that consists of eleven work areas containing a total of 33 machine resources.

The processing duration for each type of operation is defined on a per-lot basis, with lot size being an attribute of every order. The processing durations for all machine setup operations are fixed according to resource type. Note that setup activity is required to accompany each servicing activity *unless* the immediately preceding assignment (if one exists) for the resource in question involved an operation on a product within the same blade family.

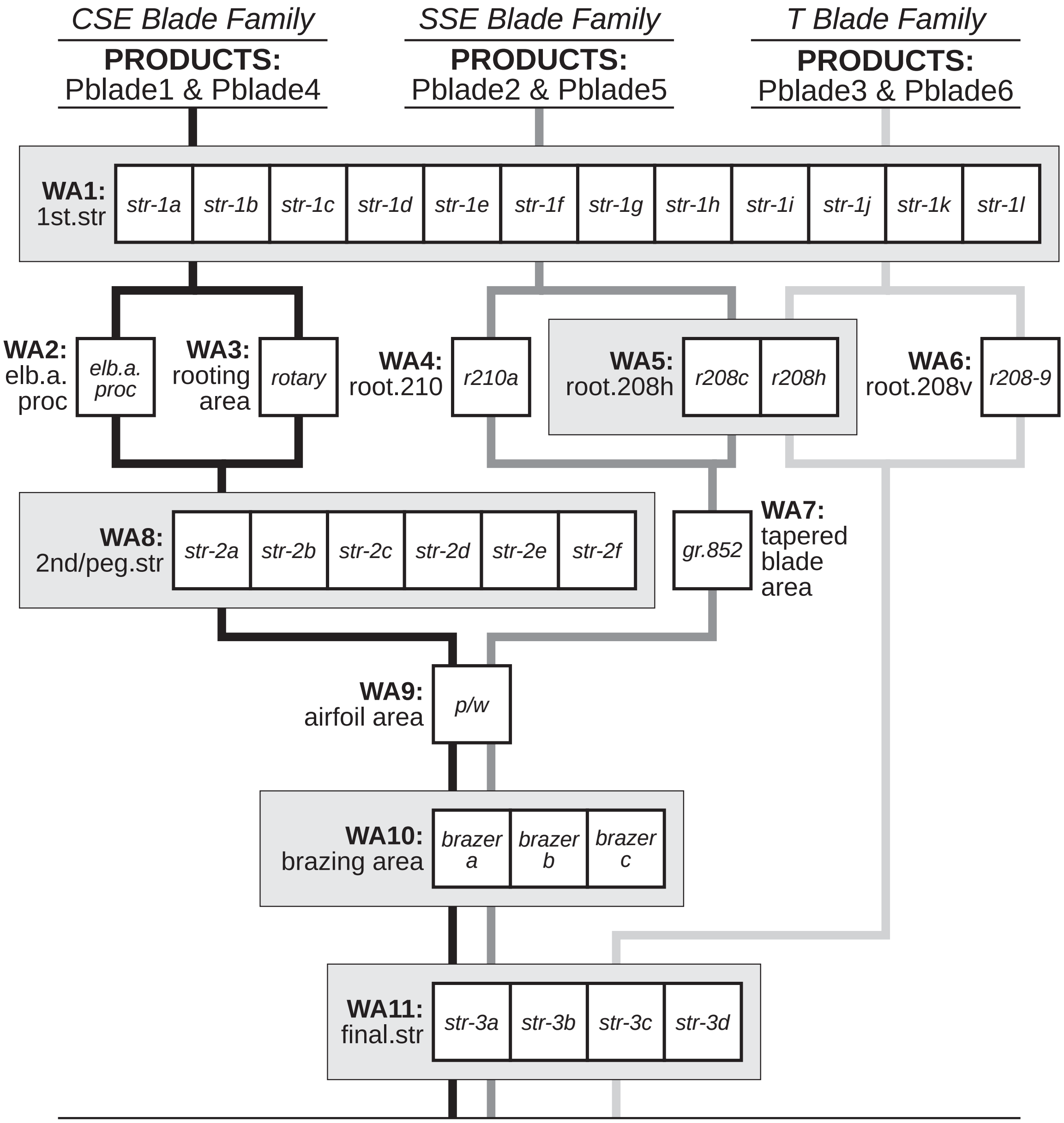
A collection of benchmarking data for this application, consisting of 22 order sets organized into 18 different categories, is defined in [Chiang *et al.*, 1990].[4] Each category corresponds to a group of parameter settings that define the product mix, priority class, order lead time, order release pattern (daily, weekly, or at exponentially distributed intervals), and batch size (orders are released periodically to the system) for for a set of orders. Two of the order sets contain 85 orders, while the remaining sets contain 120. Each order belongs to one of six priority classes, indicating the importance of meeting its due date. Tardiness penalties are determined according to these priority classes.

A detailed summary of the experiments for which this benchmark data was developed can be found in [Ow, 1986]. The specific implementation details of the TCP system are provided in Section A.2.

5.5 The ARM System

The ARM (*Airport Resource Management*) scheduling application implements the airport ground service scheduling domain described in Section A.1.1, and is capable of producing ground servicing schedules for three different types of airline passenger flights. The primary objective in the AGSS domain is to minimize tardiness. Flight departure times must therefore be met as often as possible. There is, however, no reward for the early departure of a flight, signaling an important distinction between this domain and many other RCSP domains (such as the job-shop implemented by the TCP system). The priority of every flight in ARM is identical (set to 1), so that the average tardiness cost is represented in terms of units of time (minutes).

[4] Four of the 18 categories have double order sets.



This figure is based on Figure 2 (page 3) from CMU-RI-TR-90-05, ***Factory Model and Test Data Descriptions: OPIS Experiments****,* *by Whay-Yu Chiang, Mark S. Fox, and Peng Si Ow.*

Figure 5.1. TCP Process Routings and Factory Layout

The three flight types handled by ARM are *Arrival*, *Departure*, and *Turnaround*. Their required ground servicing activities consist of anywhere from three to seven resource-requiring operations, and three to four non-resource-requiring operations, and more, depending on connecting flight relationships. Any flight may be linked to any number of connecting flights, requiring an additional resource-requiring operation, and four non-resource-requiring operations, for each connection. Each service type includes a single aggregate resource-requiring task, which provides a work area (an airport gate) at which all other ground servicing activities take place. In addition, some of the operations, such as LOAD BAGGAGE and ENPLANE, have a preference for being completed as late within their jobs as possible. Finally, most of the ground servicing work is performed in parallel.

The processing duration for each type of operation is dependent on the type of product involved, specifically the size of the aircraft. Because the AGSS domain includes mobile resources, much of the setup activity associated with the servicing operations involves travel throughout the airport environment. The duration of such activity depends on the locations involved, and the traveling speed of the resource classes. Occasionally, travel activity is not required, in case a resource already available at a desired location. Otherwise, all required setup activities must always be performed. Finally, there are a limits on the pairing of resources and operations, based on various technological constraints.

A variety of layouts containing a varying number of stationary and mobile resources can be defined. Figure 5.2 presents a simplified model of the Northwest Airlines terminal at Detroit's Metropolitan Wayne County Airport. Note that only the assorted shop locations and maximal set of stationary gate resources are shown. All defined mobile resources are initially placed at the GARAGE shop location.

The flight timetables used in the experiments with ARM are subsets of a complete daily timetable of flights run by Northwest Airlines (and its client airlines) through the Detroit Metropolitan Wayne County Airport. The entire Northwest Airlines timetable consists of more than 4330 weekly takeoffs and landings, averaging over 600 per day.[5] We have assembled a set of ten flight timetables and a collection of corresponding airport layouts for evaluation purposes. Each timetable corresponds to a particular period of time during the day, and includes all flights that either arrive or depart during that period. Figure 5.3 shows the breakdown (according to the type of ground servicing required) of the flights in each of the ten timetables.

The ten corresponding layouts for these timetables were constructed by determining, for each timetable, the least number of resources of each type required by DSS(MPH) or any of the assorted DSS scheduling heuristics (those described in Section 5.3) for producing a schedule with no tardiness. This information was used to constructed the *Minimum Supply* layouts, which are described in Table 5.1. These layouts are used in all of the ARM experiments.

The specific implementation details of the ARM system are provided in Section A.1.

[5] It should be noted that because explicit flight schedules identifying the exact matchings of arriving and departing flights with specific aircraft have been unavailable to us, the schedules used in these experiments were constructed by hand from a readily available public timetable. Arriving and departing flights with identical numbers have been linked to single aircraft to form *Turnaround* flights, as are flights having numbers that differ by one, and arrive from, and depart to, the same city. All other flights were left as separate *Arrival* or *Departure* flights.

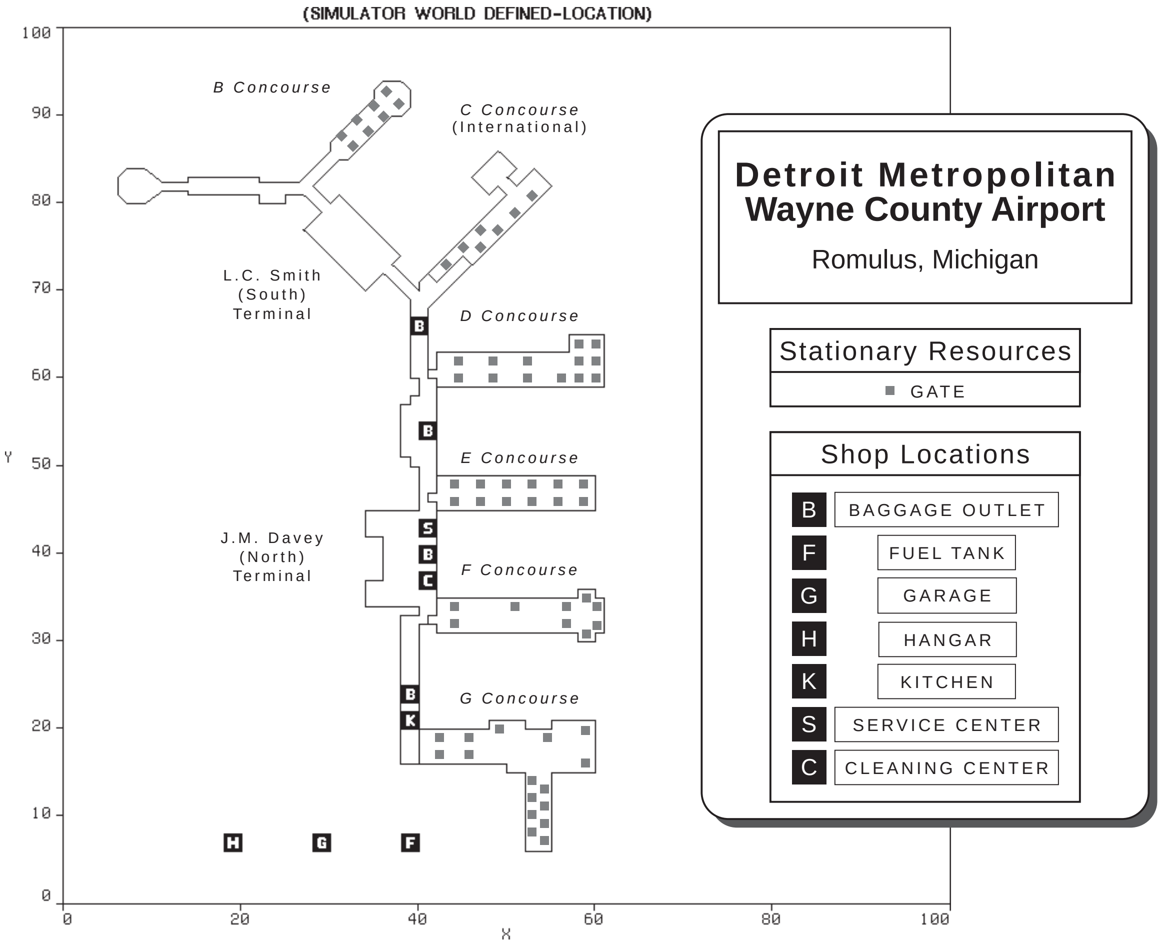


Figure 5.2. (Simplified) Detroit Metropolitan Wayne County Airport Showing All Defined Stationary Gate Resources and Shop Locations.

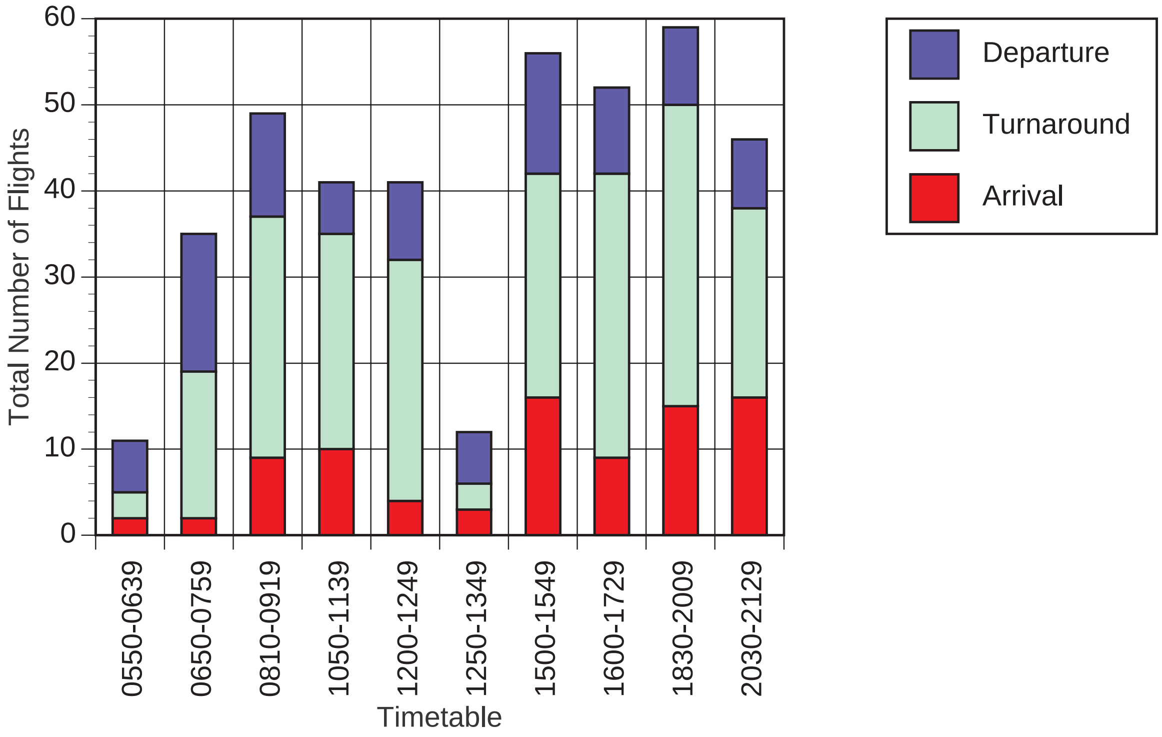


Figure 5.3. ARM Experimental Flight Timetables.

Table 5.1. ARM *Minimum Supply* Layouts

Timetable	Number of Flights	Gates	Baggage Trucks	Catering Trucks	Cleaning Trucks	Fuel Trucks	Service Trucks
		Domestic (Intl.)					
0550-0639	11	11 (0)	10	5	2	4	1
0650-0759	35	33 (2)	24	11	5	16	4
0810-0919	49	45 (4)	33	12	8	18	5
1050-1139	41	39 (2)	30	10	7	13	4
1200-1249	41	37 (4)	30	12	7	18	5
1250-1349	12	10 (2)	5	2	2	4	1
1500-1549	56	50 (6)	33	11	9	15	4
1600-1729	52	45 (7)	32	13	8	18	5
1830-2009	59	58 (1)	30	10	8	15	4
2030-2129	46	41 (5)	37	10	10	13	5

5.6 Experiments and Evaluation

One of the primary contributions made by DSS to the field of knowledge-based scheduling involves the development of a multi-faceted approach to solving dynamic RCSPs that applies to a variety of scheduling domains. At specific and frequent decision-making points throughout the scheduling process, the proper consultation of relevant scheduling perspectives coupled with the intent to preserve the options necessary for handling future conflicts, helps DSS(MPH) to make better-informed decisions in spite of the uncertainty that exists within the environment. This approach permits DSS to react quickly and effectively to unexpected events. In this section, we will present a series of experiments designed to support the contributions represented by our approach.

We begin by reiterating the specific research contributions of our multi-faceted approach to solving dynamic RCSPs.

- Use of slack time to preserve flexibility and limit schedule disruption.
- Quick and effective reaction to dynamic environments.
- Consultation of multiple perspectives in all scheduling and control decisions.
- Accommodation of additional domain complexities.

We now present a description of the four experiments we have designed, executed, and evaluated, to achieve these contributions.

- **Comparisons with Common Benchmarks**. In this experiment, we make use of the experimental data developed for the evaluation of the initial OPIS system [Chiang *et al.*, 1990], to place the performance of DSS (using DSS(MPH)) within the context of a number of important knowledge-based scheduling systems.
- **Dispatch Scheduling**. This experiment tests the ability of DSS(MPH) and the assorted DSS heuristics to continually accommodate unexpected orders into their developing schedules, and illustrates the degree to which each approach is dependent on a preliminary analysis of the order set. This experiment closely models dynamic, real-world factory conditions.
- **Coping with Resource Failures**. This experiment tests the ability of DSS(MPH) and the assorted DSS heuristics to react quickly and effectively to the problems that arise when resources break down during the execution of a schedule.
- **Unexpected Order Sets**. In this experiment, we test the ability of DSS(MPH) and the assorted DSS heuristics to comprehend fully a continually narrowing solution space.

The benchmark experiment shows that DSS (using DSS(MPH)) outperforms a number of existing knowledge-based scheduling systems, in minimizing both the average tardiness cost per order, and the percentage of tardy orders. These results indicate that DSS(MPH) is able to organize its scheduling activities in such a way as to limit the introduction of tardiness. It does so in two important ways. First, it maintains flexibility throughout the scheduling process by preserving available slack time, which eases the constraints on the current state of problem solving. Secondly, it consults multiple, relevant scheduling perspectives at key decision-making points, to provide it with the means to understand fully the state of problem solving, and thereby organize its decision-making strategy accordingly. Furthermore, these results show that even within scheduling domains where the compaction of the schedule is an objective of the scheduling process, that the basic approach of DSS is not hampered by such objectives nor its inability to intentionally compact its developing or completed schedules.

In the dispatch scheduling experiments, we show that DSS(MPH) is able to minimize both tardiness and computational effort, despite the need to react continually to the developing nature of the problem. The comparisons between DSS(MPH) and the assorted DSS heuristics indicate that the frequent consultation of multiple scheduling perspectives provides DSS(MPH) with the edge in understanding better the nature of the current overall scheduling problem. The ARM system used in these experiments also demonstrates the wide applicability of DSS, specifically its ability to represent a variety of complex RCSP domain requirements. Finally, experimentation with both the TCP and ARM systems demonstrates a consistency in performance results that extends across substantially different domains.

The results from the resource failure experiments indicate that the well-informed DSS(MPH) heuristic is better able to react to the difficulties presented by unexpected machine breakdowns, in terms of maximizing both schedule quality and computational efficiency. DSS(MPH) achieves a consistently lower average tardiness cost per order, and even produces shorter average WIP times, despite the loss of important resources. It also limits the total number of KSA invocations, and maximizes the use of its most efficient reservation-securing KS, evidence of a full understanding of the changing conditions within the environment.

Finally, our experiments with unexpected order groups provide a final example of the value of consulting multiple scheduling perspectives, as performed by DSS(MPH). As the solution space tightens with the scheduling of each order set, DSS(MPH) continues to successfully minimize tardiness and maximize its computational efficiency.

Both the collection and filtering of data for the metrics described in Section 5.2 and the definition and running of the experiments involving DSS described within this section were accomplished using the Common Lisp Instrumentation Package (CLIP) [Westbrook *et al.*, 1992].

5.6.1 Comparison with Common Benchmarks

In the following experiments, TCP was run in a batch scheduling mode, with the entire set of unchanging orders known from the outset. There were no surprise orders and no resource failures. These experiments illustrate the degree to which each scheduling approach is able to maintain flexibility in its developing schedules, and organize its decision-making strategy according to its understanding of the overall scheduling problem. We now compare the performance of DSS(MPH) (through TCP) with that of OPIS 0, ISIS, and COVERT, according to each of the stated OPIS scheduling objectives.

5.6.1.1 Average Tardiness Cost per Order

Figure 5.4 shows a graph comparing the tardiness cost results obtained by DSS(MPH) and the OPIS 0, ISIS, and COVERT systems.[6] DSS(MPH) produced lower average tardiness costs per

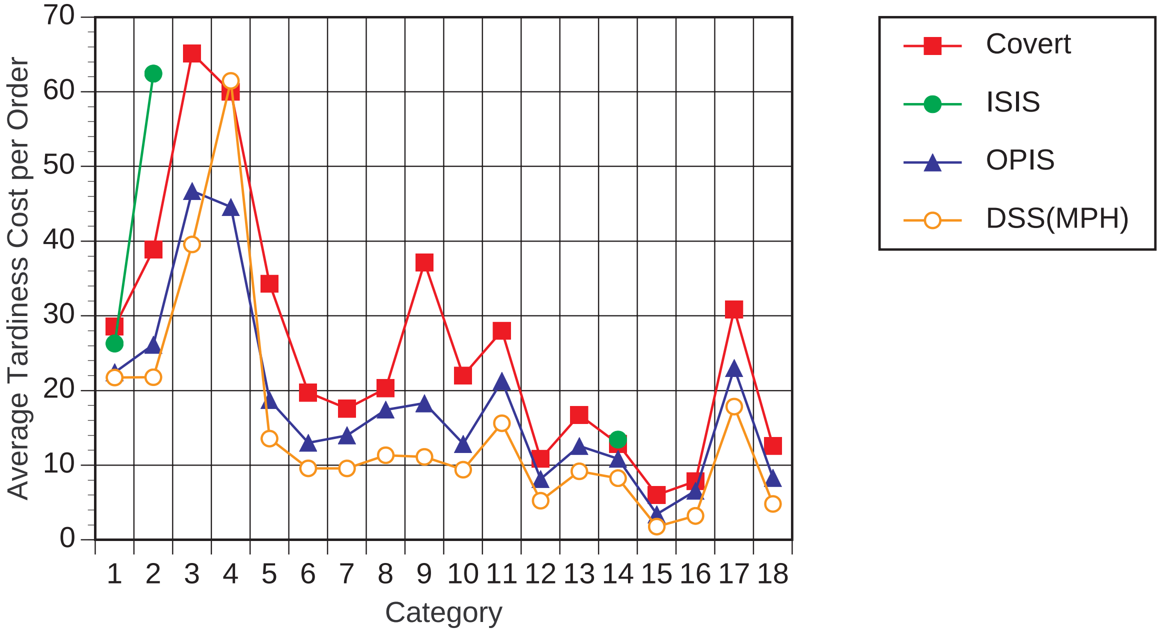


Figure 5.4. Comparison (via TCP) of the Average Tardiness Cost per Order Between DSS(MPH) and the OPIS 0, ISIS, and COVERT Systems (Lower is Better).

order for all but one of the 18 order set categories. DSS(MPH) was able to produce schedules with lower average tardiness costs per order, at an average level of approximately 26% below the costs achieved by OPIS 0. In addition, the percentage of tardy orders produced by DSS(MPH) across all of the order set categories ranged between 9.17 and 62.5, with an average of 29.57. The percentage of tardy orders produced by OPIS 0 ranged "fairly evenly" between 20 and 67 [Ow and Smith, 1988, page 104].

These are very important results for confirming the success of the least-commitment DSSscheduling approach in maintaining flexibility through the preservation of slack time in its developing schedules, to provide attractive scheduling options for solving outstanding subproblems. In fact, even when DSS(MPH) was replaced with the classical heuristics described in Section 5.3, DSS, running in batch mode, was still occasionally able to outperform OPIS 0 on average, in terms of minimizing tardiness. DSS(MINSLK) achieved average tardiness costs per order that were only approximately 26% higher, on average, than those costs achieved by OPIS 0, and outperformed OPIS 0 in 8 of the 18 categories. The results achieved by DSS(MINEST) were approximately 4% lower, outperforming OPIS 0 in 13 categories, while the DSS(MINLFT) results were approximately 10% lower, outperforming OPIS 0 in all but 4 of the categories. Regardless of the heuristic being used, the flexibility afforded by the basic least-commitment DSS scheduling approach, and the resulting benefits, are clearly evident.

[6] Data from the OPIS, ISIS, and COVERT comparison experiments has been extrapolated from the graphs presented in [Ow and Smith, 1988]. All numerical comparisons (except those involving the percentage of tardy orders) are therefore based on estimated values.

5.6.1.2 Average Work-in-Process Time per Order

As shown in Figure 5.5, the WIP results obtained by DSS(MPH) were worse than both ISIS and OPIS 0 (though better than COVERT). This is because DSS does not currently have the ability to compact its schedules by removing any slack time that may have been incorporated into its jobs. In addition, the reservation-securing KSs implemented for DSS can be severe in the amount of tardiness they introduce when a reservation does not reside within the current allowance of a service goal. For example, the *Right Shift* KSs perform a right shift on the order schedule, instead of the resource schedule, thereby causing operations to be shifted occasionally some distance downstream until an available resource is found. Additionally, once the due date has been extended for an order, DSS is currently unable to reset that date if the cause of the original extension is removed. Both ISIS and OPIS are designed to compact their developing order schedules throughout the scheduling process. As a result, while these systems are able to achieve lower WIP times, we have seen from the tardiness cost results in Figure 5.4 that this ability comes at the expense of maintaining important problem-solving flexibility, and by extension, schedule quality.

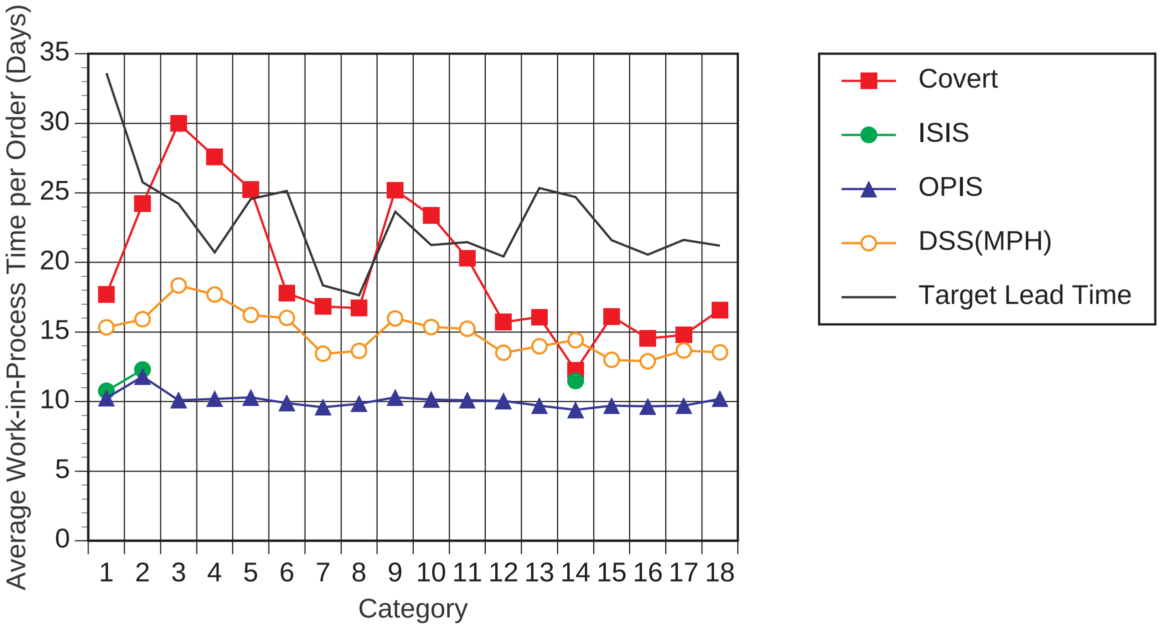


Figure 5.5. Comparison (via TCP) of the Average Work-in-Process Time per Order Between DSS(MPH) and the OPIS 0, ISIS, and COVERT Systems (Lower is Better, Below Target is Acceptable).

The average mean WIP times achieved by DSS(MPH) were approximately 47% above those times achieved by OPIS 0. It should be noted, however, that DSS(MPH) still managed to achieve average WIP times that were within the average target lead times for each order set category.

5.6.1.3 Total Number of Machine Setups

The following comparisons involve the total numbers of machine setups required for a given schedule.

Figure 5.6 shows the total number of machine setups in the schedules produced by DSS(MPH), OPIS 0, ISIS, and COVERT. DSS(MPH) attempts to limit the total number (and processing duration) of machine setups in a schedule by equipping its reservation-securing KSs with value-ordering heuristics that favor those reservations containing the least amount of setup operation processing. While this approach does help minimize the total number of setups, its success is limited by the tendency for DSS(MPH) to jump opportunistically across resource classes in the course of problem solving, instead of working on subproblems in a strictly resource-based fashion that is conducive to minimizing the need for setups.

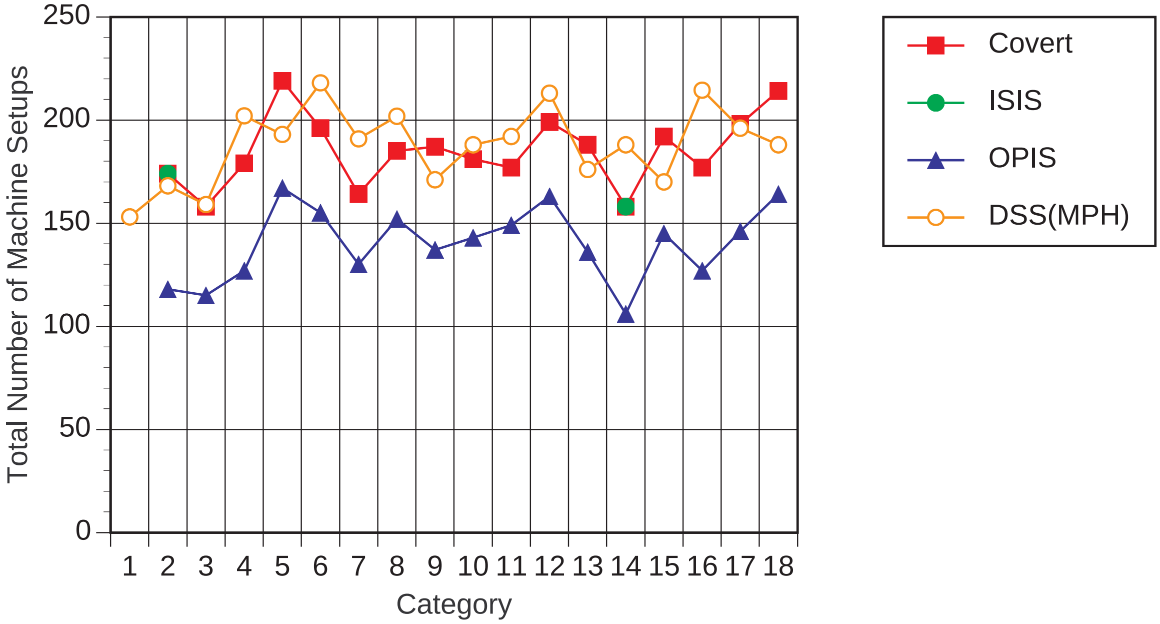


Figure 5.6. Comparison (via TCP) of the Total Number of Machine Setups Required by DSS(MPH) and the OPIS 0, ISIS, and COVERT Systems (Lower is Better).

Figure 5.7 shows the total number of *bottleneck* machine setups in the schedules produced by DSS(MPH), OPIS 0, ISIS, and COVERT. Referring back to Figure 5.1, the five bottleneck work areas are WA2, WA3, WA4, WA5, and WA6, as stated in [Chiang *et al.*, 1990, page 2]. These work areas contain a total of six machines responsible for handling the second operation in the process routing for any turbine blade. They were classified as bottlenecks based on the OPIS benchmark data order sets.

With its two-perspective scheduling approach, OPIS 0 first produces the schedules for all declared bottleneck resources, and then completes schedules for each order using the order-based scheduling mechanism implemented in ISIS. And overall, the OPIS 0 approach does exhibit the best performance among ISIS, COVERT, and DSS(MPH), in terms of minimizing the total number of machine setup operations *across all resources*. DSS(MPH), however, outperforms the other three systems when it comes to minimizing the total number of *bottleneck* machine setups. Interestingly enough, a comparison of the graphs in Figures 5.6 and 5.7 suggests that the OPIS 0 resource-based scheduler, which is responsible for scheduling the bottleneck resources, is not as successful in minimizing setups as is its order-based scheduler. On average, the total number of setup operations for bottleneck resources in the schedules produced by OPIS 0 accounts for

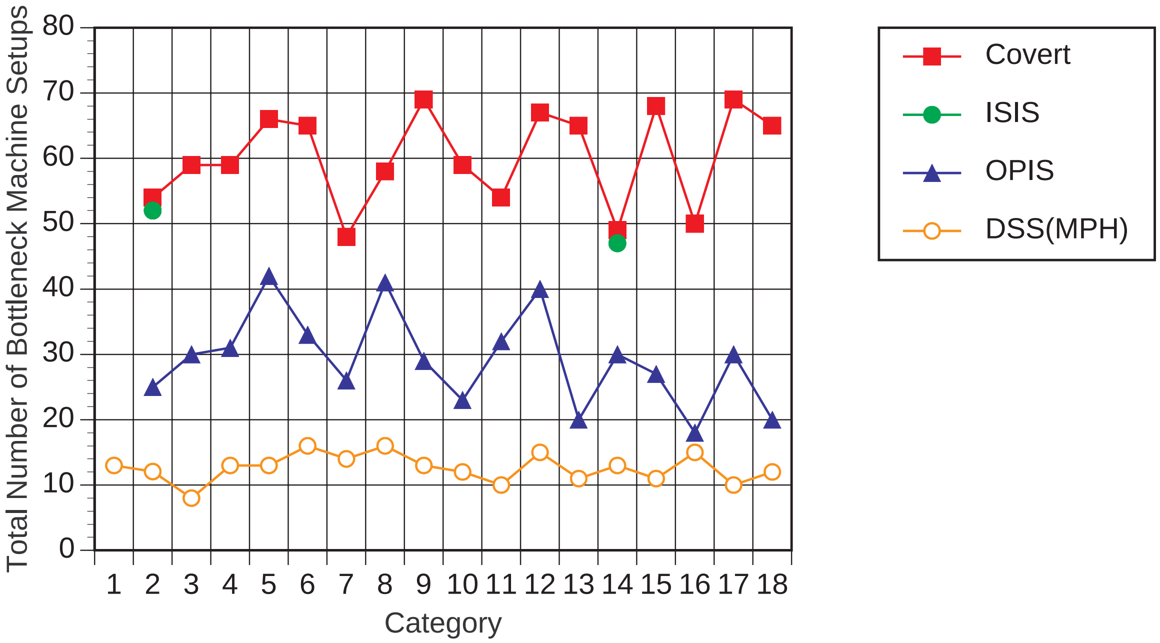


Figure 5.7. Comparison (via TCP) of the Total Number of Bottleneck Machine Setups Required by DSS(MPH) and the OPIS 0, ISIS, and COVERT Systems (Lower is Better).

approximately 21% of the total number of setup operations across all resources. The ratio for both ISIS and COVERT exceeds 30%. DSS(MPH), however, achieves a very low ratio of 6.69%. The variable-ordering capabilities of DSS(MPH) and the value-ordering heuristics of DSS therefore contribute greatly to minimizing the total number of required setup operations for bottleneck resources *while bottleneck conditions exist.* After those conditions begin to recede, the total number of machine setups required for all non-bottleneck resources begins to rise.

5.6.2 Reactive Analysis

In this section, we focus on one of the major goals of this research, namely to test the effectiveness of our approach in reacting to various kinds of unexpected events that occur within real-world scheduling environments. In the first set of experiments, we test the basic reactive capabilities of DSS(MPH) (using TCP) when operating in a dispatch mode, where the order set develops gradually, thereby precluding a complete view of the overall problem, and forcing the scheduler to do the best with what it knows at any point in time. In the second experiment, we evaluate the performance of DSS(MPH) (again using TCP) when having to react to unexpected resource failures (and still in dispatch mode).

5.6.2.1 Dispatch Scheduling

Schedulers often do not have the luxury of knowing in advance the complete set of orders for which they must produce a schedule. In most job-shop scheduling environments, orders are received by the shop during the execution of a working schedule. These new orders must be incorporated into the existing schedule as it executes. These dispatch situations limit the options available to the scheduler by narrowing the search space as previously made decisions are executed. While this reduces the scope of the problem at every point, the ability to anticipate future conflicts is also lost, because knowledge of future orders does not exist. These situations force a scheduler to understand to the best degree possible the current state of problem solving when making its scheduling decisions, so that it can be prepared for whatever situation may develop. Dispatch mode in DSS allows orders to be received throughout the entire scheduling process, either individually, or in groups. Upon receiving new orders, DSS(MPH) analyzes the updated conditions to determine the need for modifying its present decision-making strategy.

In the two experiments below, we evaluate the performance of DSS(MPH) using both the TCP and ARM systems.

5.6.2.1.1 TCP Experiments

To simulate a dispatch scheduling mode in TCP, orders are received at a time immediately prior to their ready time.[7] TCP begins scheduling new orders immediately following their reception, and their execution may begin as early as the next time unit.

Figure 5.8 shows the average tardiness cost per order achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. In dispatch scheduling mode, DSS(MPH) produces average tardiness costs per order that are 69.18% higher, on average, than those achieved in batch mode, indicating that there is a definite penalty in limiting the information made available to the scheduler. Compared to the OPIS 0 results, however, DSS(MPH)'s average tardiness costs are only approximately 23% higher, with DSS(MPH) outperforming OPIS 0 in 4 of the 18 categories. Compared to the results of the other heuristics, DSS(MPH) produced the lowest

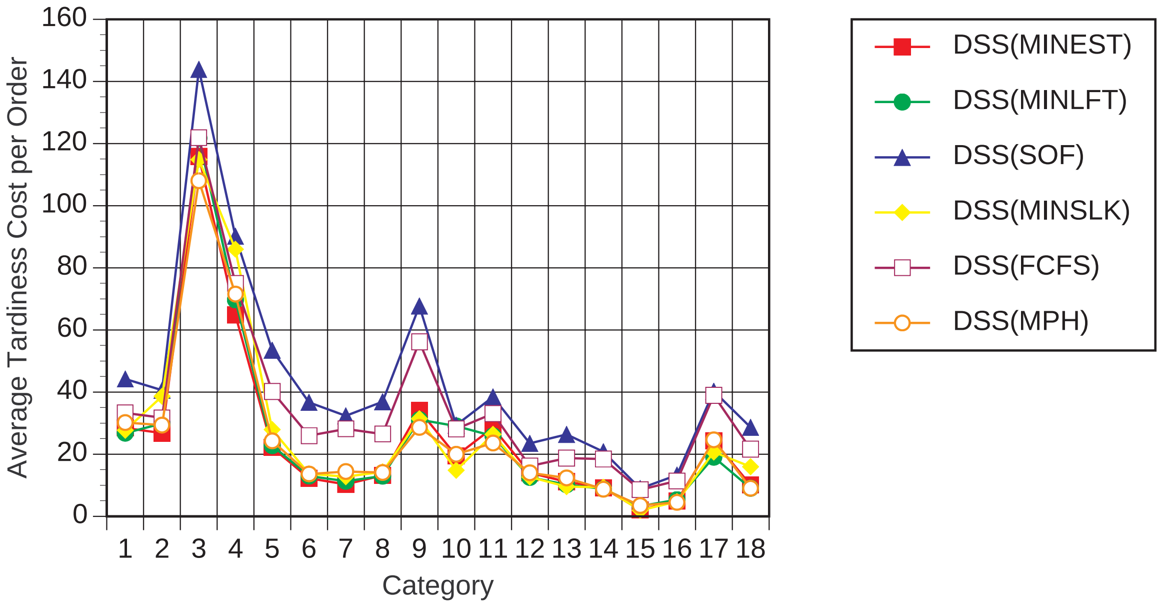


Figure 5.8. Comparison (via TCP) of the Average Tardiness Cost per Order Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Lower is Better).

[7] The advance time amount may be set to any constant time (greater than zero) to simulate the dispatch simulation.

average tardiness costs per order in 6 of the 18 categories, while DSS(MINEST), DSS(MINLFT), and DSS(MINSLK) achieved the lowest costs in 5, 4, and 3 categories (respectively). A final observation is the difference in performance between DSS(SOF) and DSS(FCFS) as compared to that of the other heuristics. Both DSS(SOF) and DSS(FCFS) are static heuristics,[8] thereby limiting severely their ability to adapt to a changing environment.

Figure 5.9 shows the percentage of tardy orders achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. In dispatch scheduling mode, the percentage of tardy orders produced by DSS(MPH) ranges between 13.33 and 57.5, with an average of 29.9, which compares closely with the range of 9.17 to 62.5, and average of 29.57, achieved in batch mode. More importantly, DSS(MPH) outperforms the other heuristics in 10 of the 18 categories (including 3 ties), while DSS(MINEST) performs the best in 3 categories, and ties for best with DSS(MPH) in 2 categories.

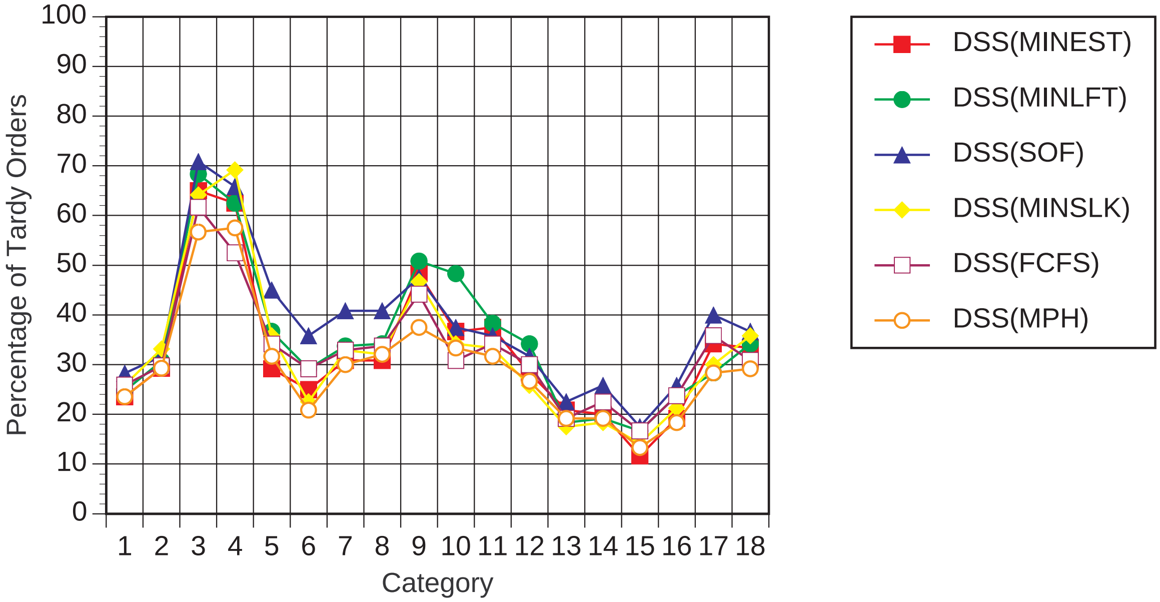


Figure 5.9. Comparison (via TCP) of the Percentage of Tardy Orders Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Lower is Better).

Figure 5.10 shows the average work-in-process time per order achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. In dispatch scheduling mode, DSS(MPH) and the other heuristics all experience longer average WIP times per order, with DSS(MPH) and all but DSS(MINSLK) exceeding the average target lead time in 2 of the 18 categories (DSS(MINSLK) exceeds the average target lead time only once). DSS(FCFS) achieves the best overall results, owing to its tendency to produce the schedule by performing a generally uniform traversal forward through time.

Figure 5.11 shows the total number of KSA invocations achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. DSS(MPH) outperforms the other heuristics in 8 of the 18 categories. DSS(MINLFT) performs best in 4 categories, while

[8] DSS(SOF) is effectively static in the TCP system, owing to the lack of aggregate tasks in the domain.

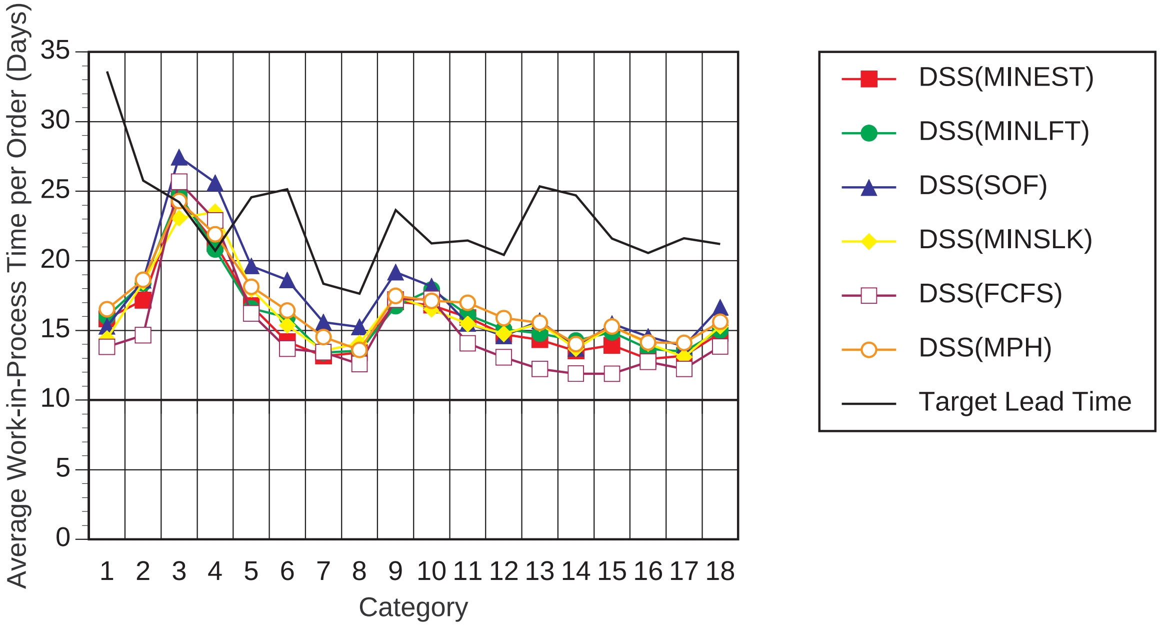


Figure 5.10. Comparison (via TCP) of the Average Work-in-Process Time per Order Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Lower is Better, Below Target is Acceptable).

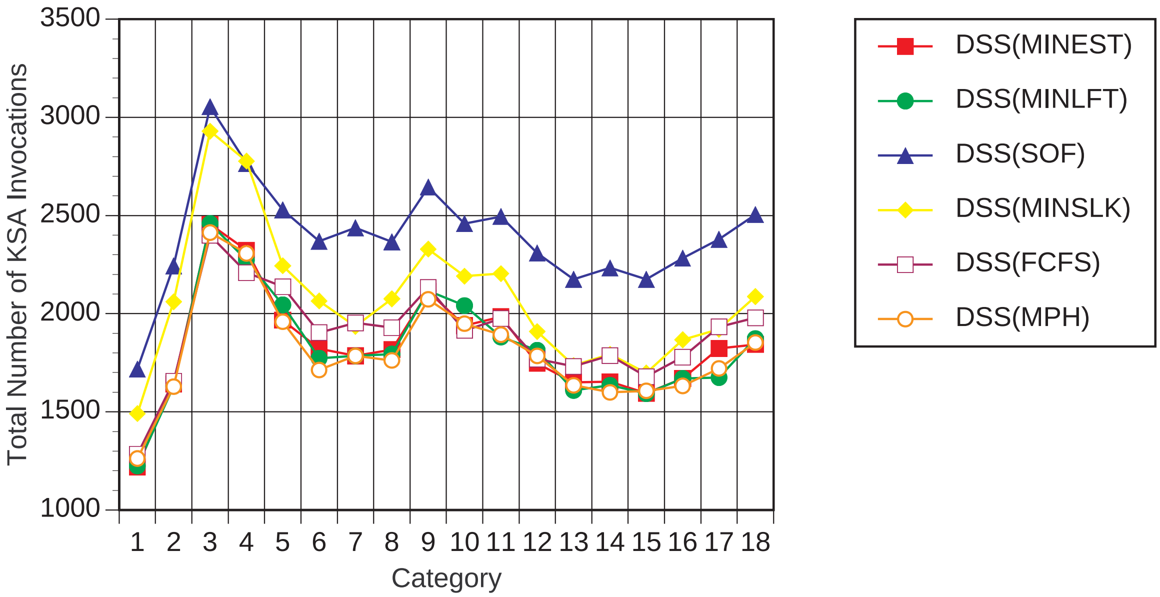


Figure 5.11. Comparison (via TCP) of the Total Number of KSA Invocations Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Lower is Better).

DSS(MINEST) and DSS(FCFS) tie with 3. An important relationship exists between DSS(MPH) and DSS(FCFS), in that DSS(MPH) outperforms DSS(FCFS) in 14 of the 18 categories, indicating that the variable-ordering of DSS(MPH) is efficient enough to use fewer KSAs than a heuristic that performs no backtracking. Finally, we note the penalty paid by DSS(SOF) for its static nature, and the penalty paid by DSS(MINSLK) for its lack of a resource-based perspective to balance its order-based view.

Figure 5.12 shows the percentage of standard *Assignment* reservations achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. The comparison of the percentage of standard *Assignment* reservations, as with the total number of KSA invocations, demonstrates the benefits of the variable-ordering capability of DSS(MPH). DSS(MPH) outperforms the other heuristics in 13 of the 18 categories, sometimes losing closely to either DSS(MINEST) or DSS(MINLFT), but often winning by large margins.

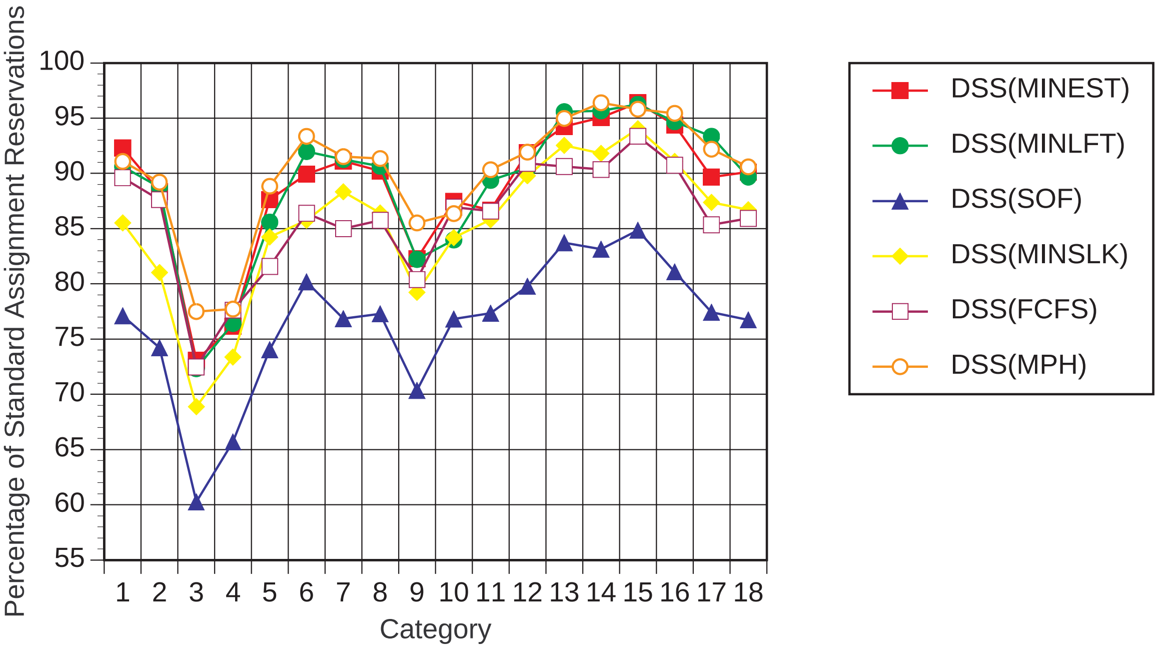


Figure 5.12. Comparison (via TCP) of the Percentage of Standard *Assignment* Reservations Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Higher is Better).

Figure 5.13 shows the elapsed processing time (in minutes) achieved by DSS(MPH) and the assorted DSS heuristics, in dispatch scheduling mode. While these results indicate the price paid by DSS(MPH) for supporting its well-informed variable-ordering process, the penalty is not as severe as might be expected. DSS(MPH) expends the most computational effort in 5 of the 18 categories, as do, however, both DSS(MINSLK) and DSS(MINLFT). DSS(MINEST) expends the most effort in 3 categories. Interestingly enough, in more than half of the categories (by a ratio of 8 to 5) where DSS(MPH) does not use the most amount of processing time, it outperforms 2 or more of the other heuristics. Finally, the value of a static heuristic in minimizing processing time is demonstrated by the generally lower results of both DSS(SOF) and DSS(FCFS).

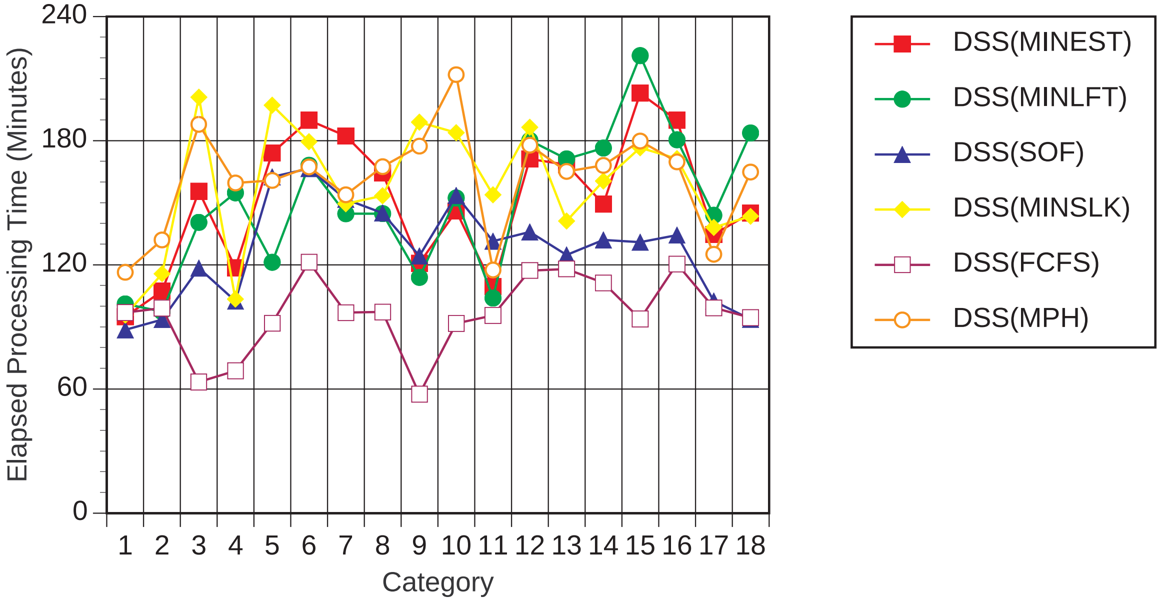


Figure 5.13. Comparison (via TCP) of the Elapsed Processing Time Among the Assorted DSS Heuristics in Dispatch Scheduling Mode (Lower is Better).

5.6.2.1.2 ARM Experiments

For the ARM experiments, we make use of the ten timetables described in Figure 5.3 and the corresponding *Minimum Supply* layouts described in Table 5.1. To simulate a dispatch scheduling mode, orders are received by ARM at different times according to their flight type. For *Arrival* and *Turnaround* flights, notification is received when the aircraft takes off from its originating city. For *Departure* flights, notification is received at a fixed point prior to their due date (departure time). Note that airport ground service scheduling is generally performed in advance, according to established timetables. As a result, the dispatch scheduling described in this section serves an experimental purpose only.

The results of the ARM experiments differ from those of the TCP experiments, owing to the existence of mobile resources in the airport ground service scheduling RCSP. As described in Section 4.2.4, mobile resources are secured by DSS a way that preserves flexibility for future related scheduling decisions. As a result, when making reservations of mobile ground-servicing resources before their designated gate work areas have been secured, the maximum possible amount of travel time for the mobile resource to get to any possible gate location is included. While this approach does preserve the scheduling options for securing stationary resources in the future, it also temporarily over-commits the mobile resources, thereby limiting their availability.

The DSS mobile resource reservation policy therefore impacts the performance of the assorted heuristics with which we are comparing DSS(MPH). It should also be noted that the timetables used in the ARM experiments are generally uniform, containing orders with similar servicing requirements that are to be processed during short scheduling horizons. Additionally, the *Minimum Supply* layouts provide a consistent supply of resources for all of the necessary tasks, preventing any particular resource class from becoming a bottleneck. We describe the affect of these features on the assorted heuristics below:

- DSS(MINEST) has a tendency to secure gates early, owing to their relatively early starting times. This allows future mobile resource reservations to be more compact, thereby providing more scheduling flexibility, and minimizing order tardiness.

- DSS(MINLFT) suffers for the same reason that DSS(MINEST) performs well. Gates also have large finishing times, so that they are secured late in the scheduling process. As a result, most mobile resource reservations are larger, thus reducing their availability.

- DSS(SOF) also suffers because of the time at which gate resources are secured. Gate-requiring subproblems have the largest processing durations, and are therefore secured last.

- DSS(MINSLK) does not suffer from its lack of a resource-based perspective, primarily because the consistent supply of resources provided by the *Minimum Supply* layouts. As a result, the consideration of slack time proves quite sufficient in accurately determining subproblem urgency.

- DSS(FCFS) performs quite similarly to DSS(MINEST). Gates are the first resources secured in each order, owing to their earliest starting times.

DSS(MPH) still performs well in this environment, despite its frequent need to over-commit in the reservation of mobile resources, but the effectiveness of its variable-ordering mechanism more than compensates for this fact.

Figure 5.14 shows the average tardiness cost per order achieved by DSS(MPH) and the assorted DSS heuristics, using the *Minimum Supply* layouts in dispatch scheduling mode. DSS(MINSLK) outperforms both DSS(MPH) and the other heuristics in 7 of the 10 timetables, while DSS(MPH) does so in 3 categories (tying once with DSS(MINSLK)). DSS(FCFS) achieves the lowest results in 2 categories (also tying once with DSS(MINSLK)). DSS(MPH), however, places a close second to DSS(MINSLK) in 6 of the 10 timetables, with an average increase of 15.74% in mean tardiness cost per order above the DSS(MINSLK) results. DSS(MINEST) remains in the pack with results generally similar to those of DSS(FCFS) (with one extreme exception), while both DSS(MINLFT) and DSS(SOF) lag behind.

Figure 5.15 shows the percentage of tardy orders achieved by DSS(MPH) and the assorted DSS heuristics, using the *Minimum Supply* layouts in dispatch scheduling mode. In this measure, DSS(MINSLK) outperforms both DSS(MPH) and the other heuristics in 9 of the 10 timetables, while DSS(MPH) does so in 4 categories (tying three times with DSS(MINSLK)). DSS(FCFS) achieves the lowest results in 3 categories (tying twice with DSS(MINSLK)). DSS(MPH) again places a close second to DSS(MINSLK) with an average increase of 25.67% in the percentage of tardy orders above the DSS(MINSLK) results. DSS(FCFS) achieves an average increase of 33.42% above the DSS(MINSLK) results. DSS(MINEST) again finishes at the fringe of the top of the pack, leaving DSS(MINLFT) and DSS(SOF) behind.

Figure 5.16 shows the total number of KSA invocations achieved by DSS(MPH) and the assorted DSS heuristics, using the *Minimum Supply* layouts in dispatch scheduling mode. The total numbers of KSA invocations serve primarily to demonstrate the division among DSS(MPH) and the other heuristics. DSS(MPH), DSS(MINSLK), DSS(FCFS), and DSS(MINEST) perform at one level, while DSS(MINLFT) and DSS(SOF) perform at another.

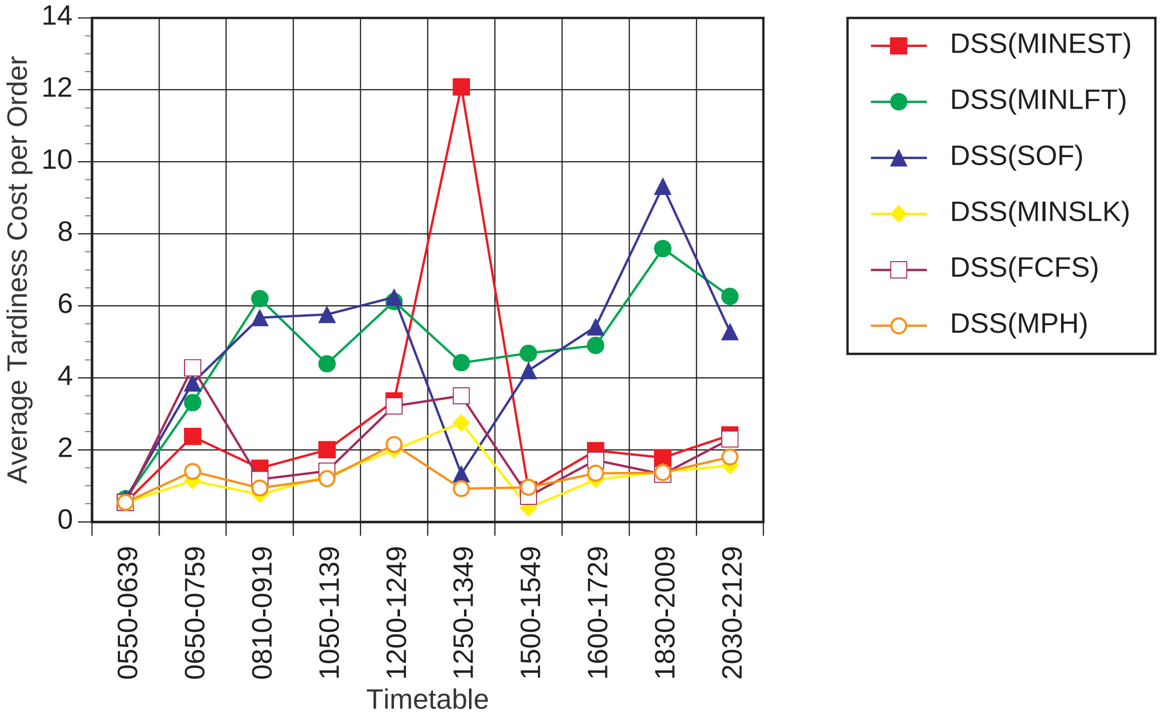


Figure 5.14. Comparison (via ARM) of the Average Tardiness Cost per Order Among the Assorted DSS Heuristics Using *Minimum Supply* Layouts in Dispatch Scheduling Mode (Lower is Better).

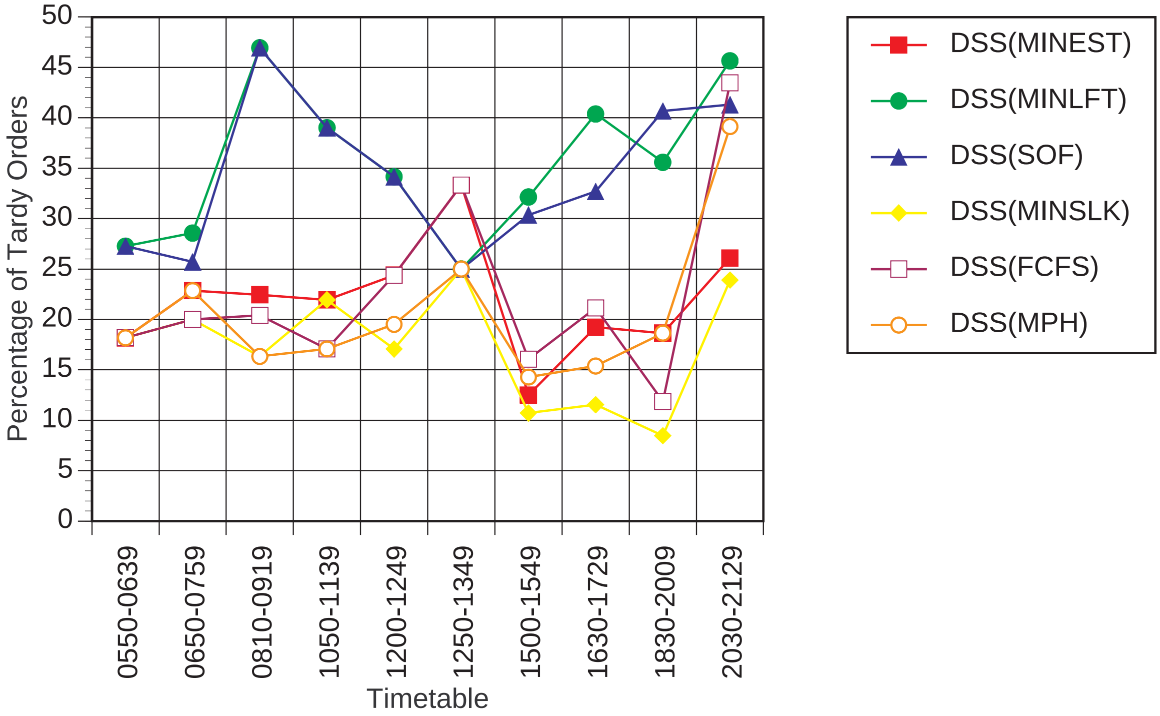


Figure 5.15. Comparison (via ARM) of the Percentage of Tardy Orders Among the Assorted DSS Heuristics Using *Minimum Supply* Layouts in Dispatch Scheduling Mode (Lower is Better).

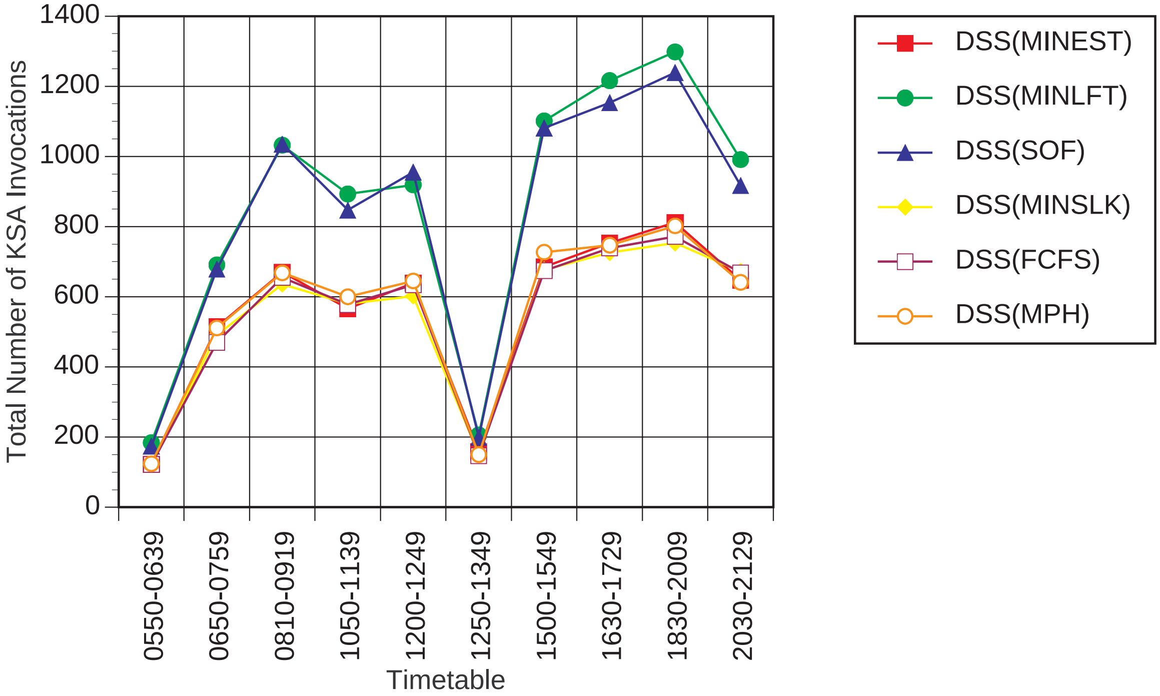


Figure 5.16. Comparison (via ARM) of the Total Number of KSA Invocations Among the Assorted DSS Heuristics Using *Minimum Supply* Layouts in Dispatch Scheduling Mode (Lower is Better).

Figure 5.17 shows the percentage of standard *Assignment* reservations achieved by DSS(MPH) and the assorted DSS heuristics, using the *Minimum Supply* layouts in dispatch scheduling mode. In this measure, DSS(FCFS) outperforms both DSS(MPH) and the other heuristics in 7 of the 10 timetables. DSS(MINSLK) produces results that average 1.8% lower, while DSS(MPH) averages 1.94% lower, and DSS(MINEST) averages 2.32% lower.

Figure 5.18 shows the elapsed processing time achieved by DSS(MPH) and the assorted DSS heuristics, using the *Minimum Supply* layouts in dispatch scheduling mode. This final measure, as expected, again demonstrates the increase in computational effort that is expended by DSS(MPH) to maintain its well-informed view of the state of problem solving. In contrast to the results from the preceding experiments with TCP, DSS(MPH) expends the most effort in the clear majority of the timetables (8 out of 10), with an average 24.22% increase in elapsed processing time over the lowest time, and an average 77.5% increase over the second-highest processing time, in those 8 cases. These increases, however, are not overly substantial, nor prohibitive, within this particular context.

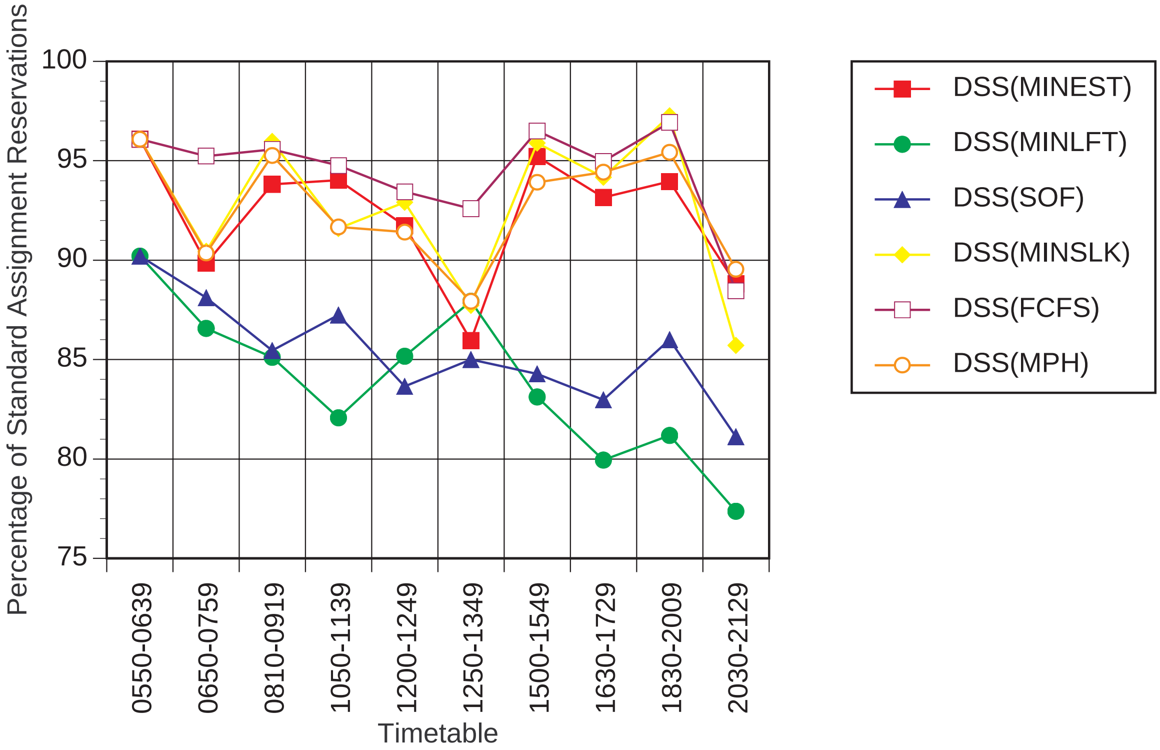


Figure 5.17. Comparison (via ARM) of the Percentage of Standard *Assignment* Reservations Among the Assorted DSS Heuristics Using *Minimum Supply* Layouts in Dispatch Scheduling Mode (Higher is Better).

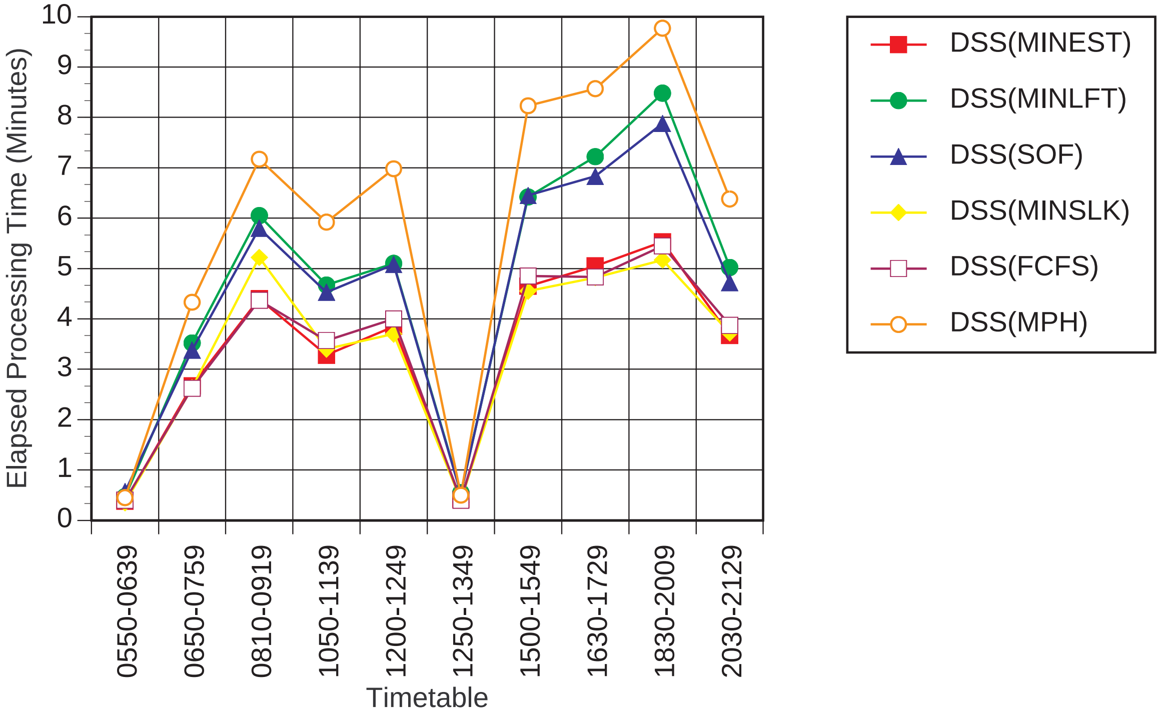


Figure 5.18. Comparison (via ARM) of the Elapsed Processing Time Among the Assorted DSS Heuristics Using *Minimum Supply* Layouts in Dispatch Scheduling Mode (Lower is Better).

5.6.2.2 Coping with Resource Failures

The ability to react effectively to resource failures is a primary means of judging the real-world applicability of a scheduling system. When resources break down, a scheduler must reevaluate both the new state of problem solving, and its current scheduling strategy, and then adjust its focus of attention accordingly. The key to the approach taken by DSS is to incorporate the reactions to resource failures into the basic problem-solving routine so that all conflicts are resolved within the context of the current state of problem solving.

The experiments in this section use the TCP system to compare how well DSS(MPH) is able to adapt to unexpected resource failures. We have selected five machines to fail at certain points along the scheduling horizon. Referring back to Figure 5.1, we arrange to have three of the FIRST STRAIGHTENING machines in work area WA1, namely STR-1A, STR-1C, and STR-1E, fail at a point exactly two and a half days into the scheduling horizon, as determined by the earliest ready time in the order set. WA1 provides the machines used to perform the first operation in the process routings for each blade family. The three failures account for 25% of the FIRST STRAIGHTENING machines. At the five-day mark, two of the FINAL STRAIGHTENING machines in work area WA11, namely STR-3B and STR-3D, also fail. WA11 provides the machines used to perform the final operation in the process routings for each blade family. The two failures account for 50% of the FINAL STRAIGHTENING machines.

Figure 5.19 shows the average tardiness cost per order achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. In terms of minimizing the average tardiness cost per order, and despite the five resource failures, DSS(MPH) clearly performs the best, producing minimal costs in 15 of the 18 categories, and coming in second-best in 2 others. In those 15 categories, DSS(MPH) achieved average tardiness costs per

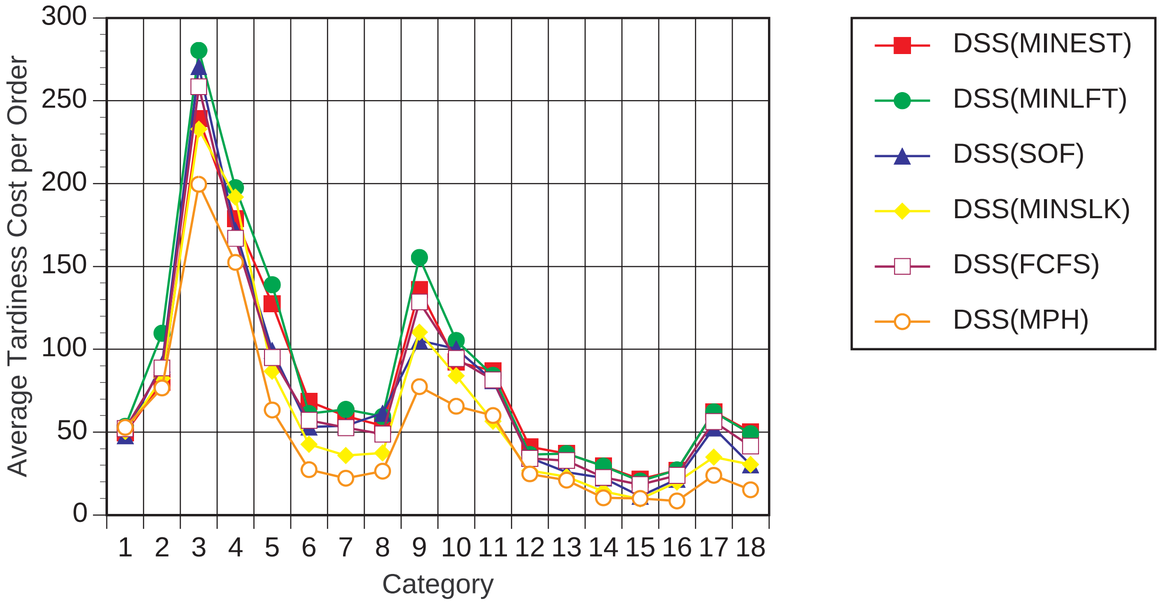


Figure 5.19. Comparison (via TCP) of the Average Tardiness Cost per Order Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Lower is Better).

order that were 25.87% lower than the second-lowest costs. DSS(MINSLK) achieves a strong second-place showing, coming in second-best in 11 of the 18 categories. Both DSS(MPH) and DSS(MINSLK) benefit from their informed analyses of the state of problem solving, although the performance of DSS(MINSLK) is achieved largely because of worse performances by the other heuristics.

Figure 5.20 shows the percentage of tardy orders achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. The percentages of

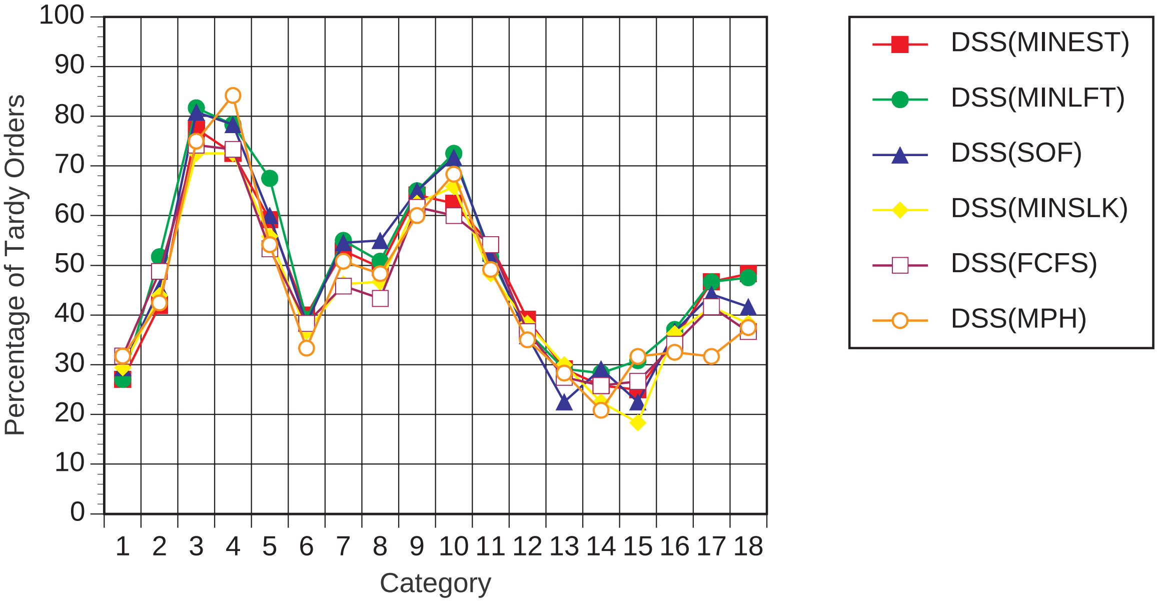


Figure 5.20. Comparison (via TCP) of the Percentage of Tardy Orders Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Lower is Better).

tardy orders achieved by DSS(MPH) and the other heuristics indicate uniform performance by each approach. Tardy orders range from between an average of 20% to 80% of the order set, with an average mean of 47%. While DSS(MPH) outperforms the other heuristics in 6 of the 18 categories, DSS(FCFS), DSS(MINSLK), and DSS(MINEST) achieve the lowest results in 5, 4, and 3 categories (respectively).

Figure 5.21 shows the average work-in-process time per order achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. The WIP results are in line with the average tardiness cost results. DSS(MPH) outperforms the other heuristics in 9 of the 18 categories, while DSS(MINSLK) does so in 7. Furthermore, both DSS(MPH) and DSS(MINSLK) exceed the average target lead times in only 4 of the 18 categories, while DSS(FCFS) does so in 5 categories, and each of the rest does so in 9. The results do demonstrate, however, a marked improvement in the ability of DSS(MPH) to minimize average WIP times per order relative to the other heuristics, under extreme conditions of the kind caused by unexpected resource failures. A look back to the results of Figure 5.10 indicates that DSS(MPH) appears more capable than the other heuristics of controlling work-in-process times in these highly constrained situations.

Figure 5.22 shows the total number of KSA invocations achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. The total

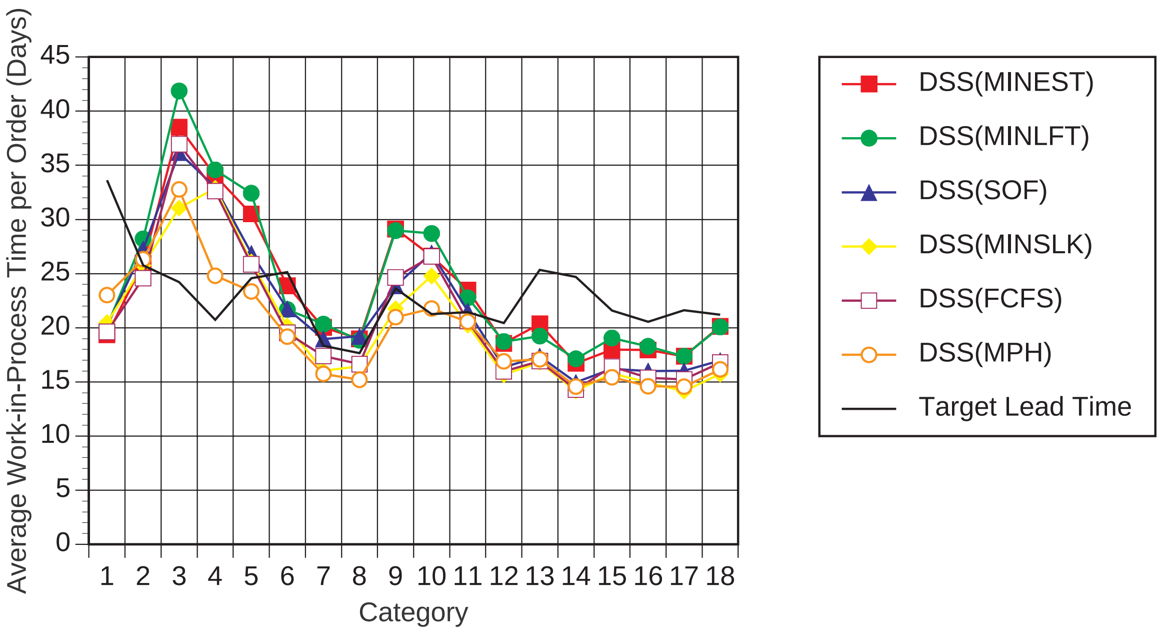


Figure 5.21. Comparison (via TCP) of the Average Work-in-Process Time per Order Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Lower is Better, Below Target is Acceptable).

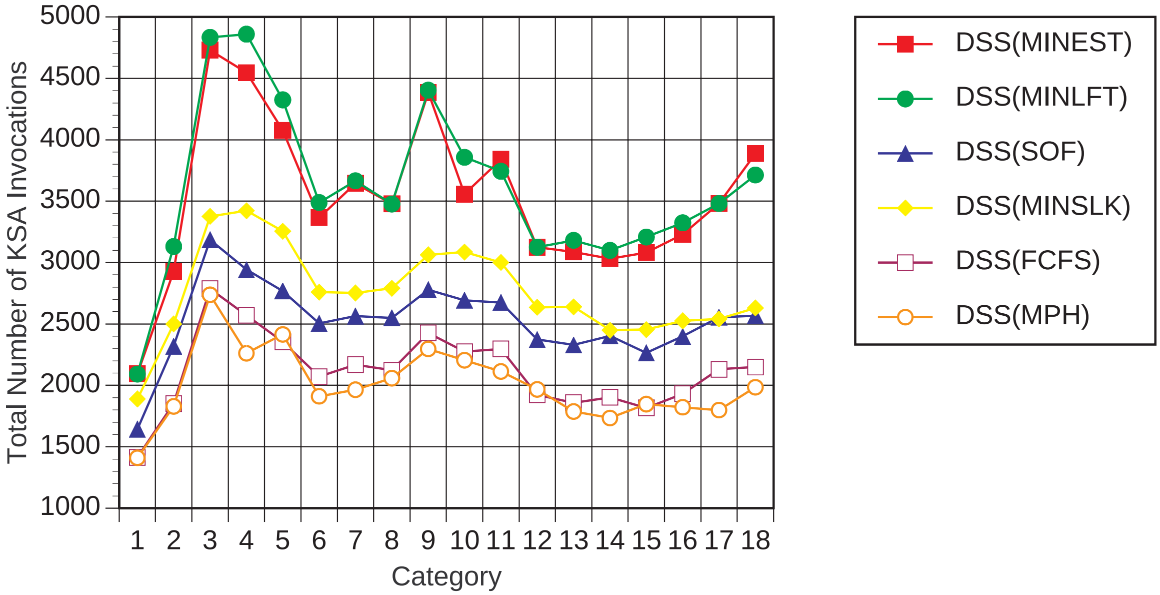


Figure 5.22. Comparison (via TCP) of the Total Number of KSA Invocations Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Lower is Better).

numbers of KSA invocations demonstrate the overall effectiveness of DSS(MPH), as well as the separation of the DSS heuristics into three performance classes. DSS(MPH) and DSS(FCFS) achieve the lowest total KSA invocations, with DSS(MPH) outperforming the other heuristics in 15 of the 18 categories, and DSS(FCFS) doing so in the remaining 3. These results place DSS(MPH) at the top performance level, owing to its well-informed variable-ordering decision-making capabilities. DSS(FCFS) performs nearly as well, the result of its utter avoidance of backtracking, which automatically limits the KSA invocation count. DSS(SOF) maintains its previous status, owing to its static nature and its lack of proper perspective on the problem, but it appears at the middle level because of worse performances by other heuristics. DSS(MINSLK) joins DSS(SOF) because of its own failure to consider an important resource-based perspective. Finally, DSS(MINEST) and DSS(MINLFT), with their reliance on an even more limited order-based perspective, clearly occupy the bottom performance level.

Figure 5.23 shows the percentage of standard *Assignment* reservations achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. DSS(MPH) outperforms the other heuristics in all of the 18 order set categories. The stratification mentioned earlier is likewise in evidence here. The performance of DSS(FCFS) is again achieved because of its lack of backtracking, which avoids producing the kinds of schedule upsets that are generally solved inefficiently.

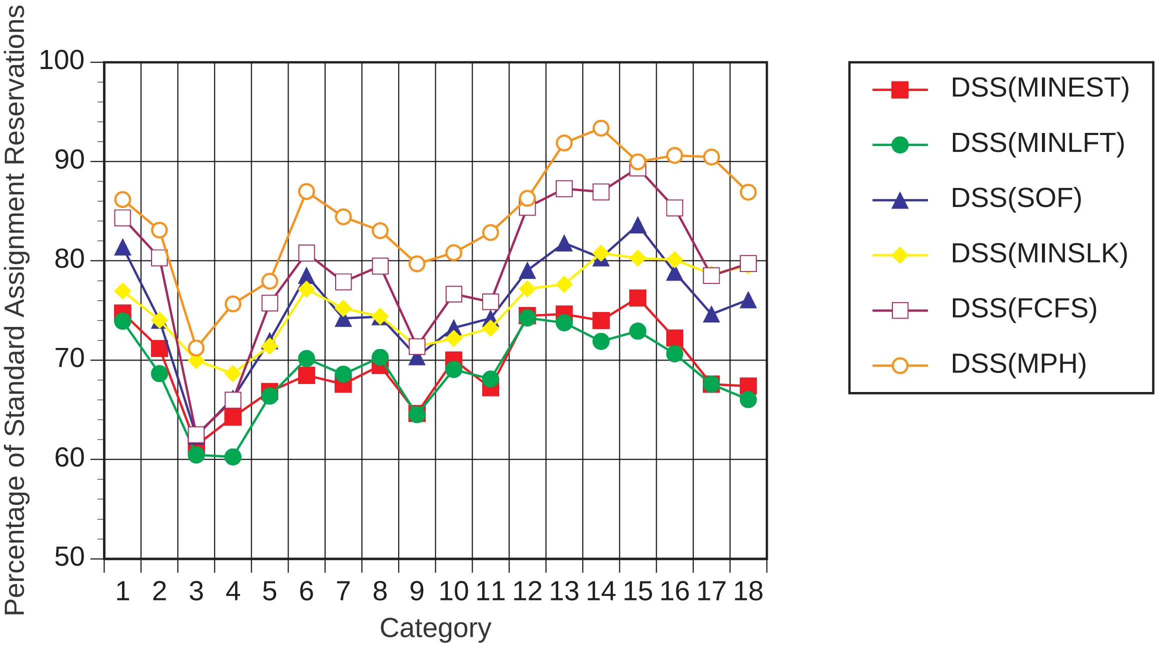


Figure 5.23. Comparison (via TCP) of the Percentage of Standard *Assignment* Reservations Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Higher is Better).

Figure 5.24 shows the elapsed processing time (in minutes) achieved by DSS(MPH) and the assorted DSS heuristics, with five resource failures in dispatch scheduling mode. The elapsed processing times indicate the price paid by DSS(MPH) to maintain its well-informed view of the state of problem solving. Both DSS(MPH) and DSS(MINEST) expend the most computational

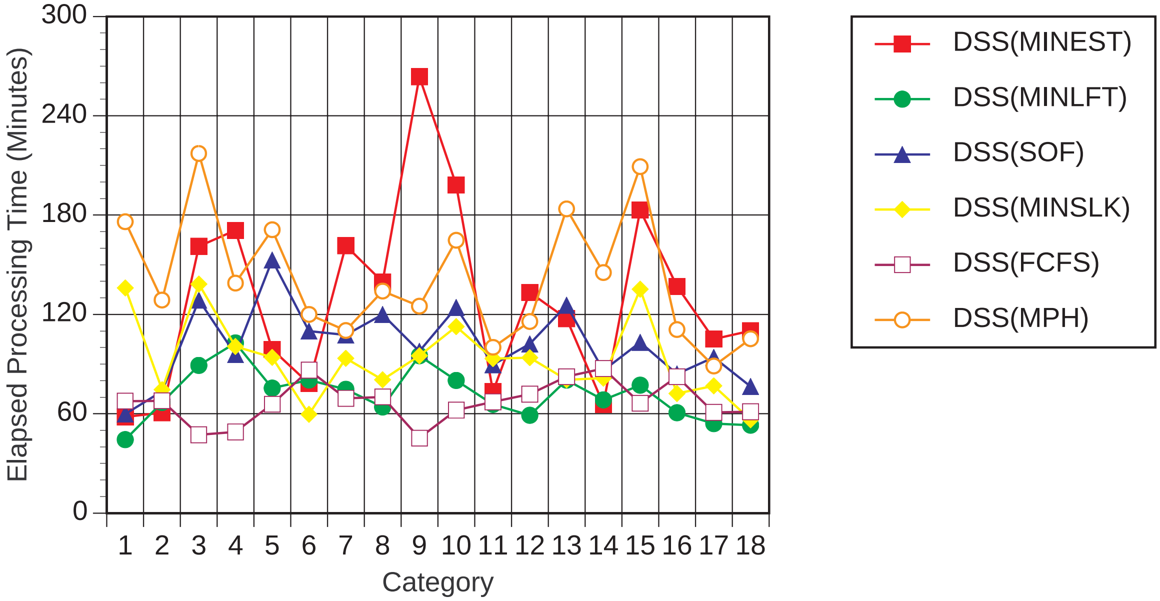


Figure 5.24. Comparison (via TCP) of the Elapsed Processing Time Among the Assorted DSS Heuristics With Five Resource Failures in Dispatch Scheduling Mode (Lower is Better).

effort in 9 of the 18 categories, with DSS(MPH) placing second-worst in 8 of the remaining categories. The computational effort of DSS(MPH), however, is balanced by successes in schedule quality and variable-ordering efficiency.

5.6.3 Accommodating Unexpected Order Sets

In these experiments, we use the ARM system to further test the performance of DSS(MPH) and the assorted DSS heuristics in highly constrained scheduling situations. DSS (via ARM) is run in batch scheduling mode, to provide the most comprehensive view of the search space for each approach. The set of orders is randomly divided into distinct subsets to be scheduled to completion, one at a time. Once a subset of orders has been scheduled, its schedule cannot be changed. Existing reservations from preceding order subsets may not be preempted. Orders in succeeding subsets must therefore manage with a continually restricted solution space. The purpose is to provide a means for judging the ability of each strategy to develop an accurate understanding of the state of problem solving, and thereby enhance its ability to produce quality solutions in response to unforeseen developments. These experiments also model real-world situations where orders arrive unpredictably, and must be incorporated into existing schedules without disrupting any previous decisions.

We make use of the ten timetables described in Figure 5.3, and the corresponding *Minimum Supply* layouts described in Table 5.1. Two experiments are described, the first involving a division of each timetable into two order groups, and the second involving a division into three order groups.

Figure 5.25 shows the average tardiness cost per order achieved by DSS(MPH) and the assorted DSS heuristics, using two order groups in batch scheduling mode. DSS(MPH) outperforms the other heuristics in 7 of the 10 timetables, clearly demonstrating its success at incorporating unforeseen orders into existing schedules when little room for maneuvering is

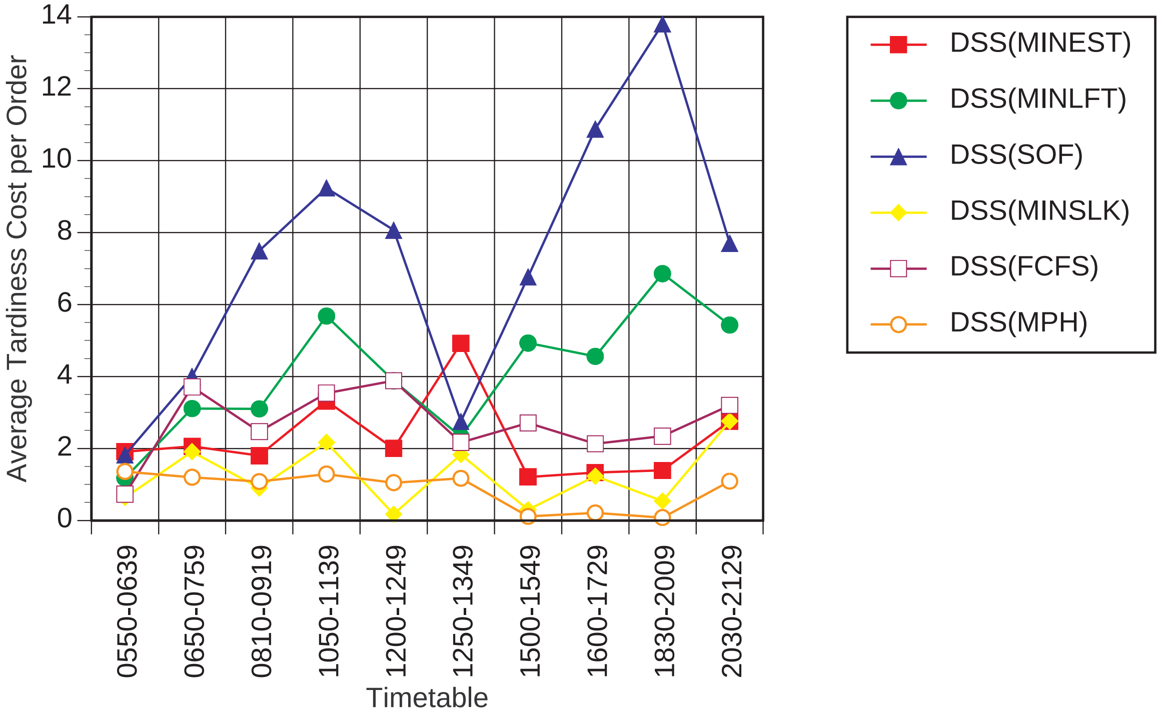


Figure 5.25. Comparison (via ARM) of the Average Tardiness Cost per Order Among the Assorted DSS Heuristics Using Two Order Groups in Batch Scheduling Mode (Lower is Better).

available. DSS(MINSLK), consistent with the previous dispatch scheduling analysis of DSS(MPH) using ARM, achieves the lowest average tardiness cost in the three remaining categories.

In the following three measures, the behavior of DSS(MPH) and DSS(MINSLK) is again quite similar to previous results, with the two approaches consistently outperforming the other heuristics, and then either alternating or sharing in achieving the best results. Figure 5.26 shows the percentage of tardy orders achieved by DSS(MPH) and the assorted DSS heuristics, using two order groups in batch scheduling mode. Figure 5.27 shows the total number of KSA invocations, and Figure 5.28 shows the percentage of standard *Assignment* reservations, under the same circumstances.

We now move to the experiments involving the division of each timetable into three order groups. Figure 5.29 shows the average tardiness cost per order achieved by DSS(MPH) and the assorted DSS heuristics, using three order groups in batch scheduling mode, and Figure 5.30 shows the percentage of tardy orders, under the same circumstances. In minimizing the average tardiness cost per order, DSS(MPH) outperforms the other heuristics in 8 of the 10 categories. In minimizing the percentage of tardy orders, DSS(MPH) outperforms the other heuristics in 7 categories, an improvement over the results using two order groups. The results achieved by DSS(MPH) and the other heuristics, in both minimizing the total number of KSA invocations, and maximizing the percentage of standard *Assignment* reservations, using three order groups, are similar to the results from the two-group experiments.

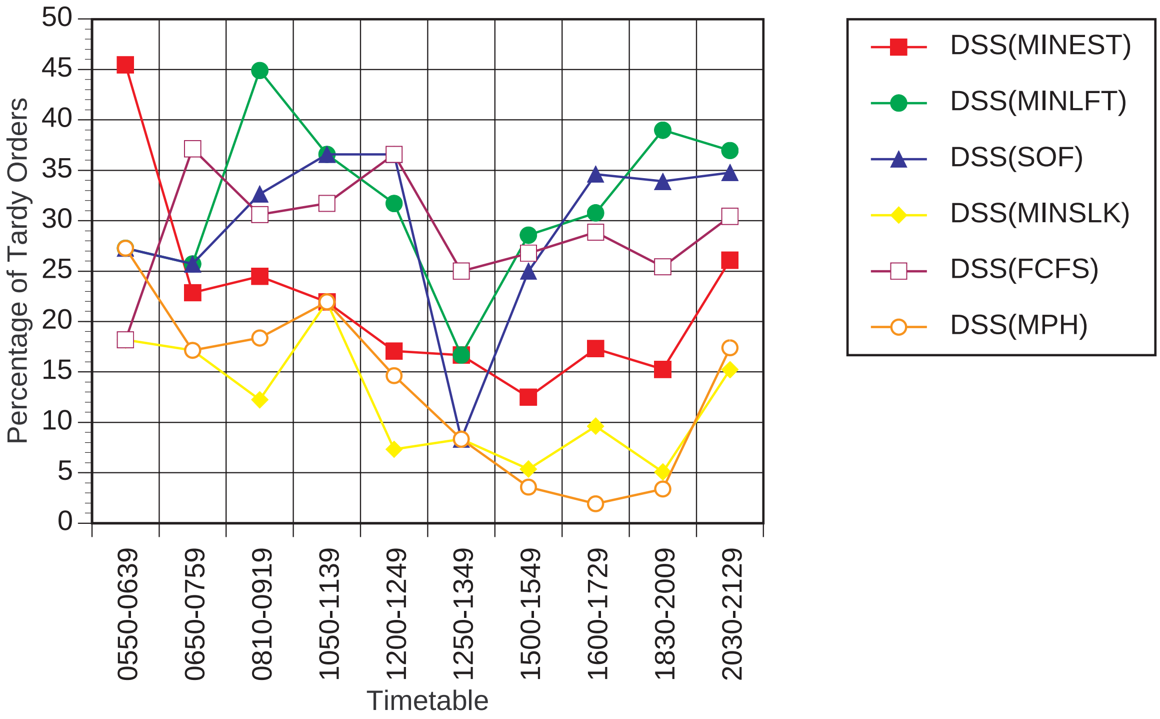


Figure 5.26. Comparison (via ARM) of the Percentage of Tardy Orders Among the Assorted DSS Heuristics Using Two Order Groups in Batch Scheduling Mode (Lower is Better).

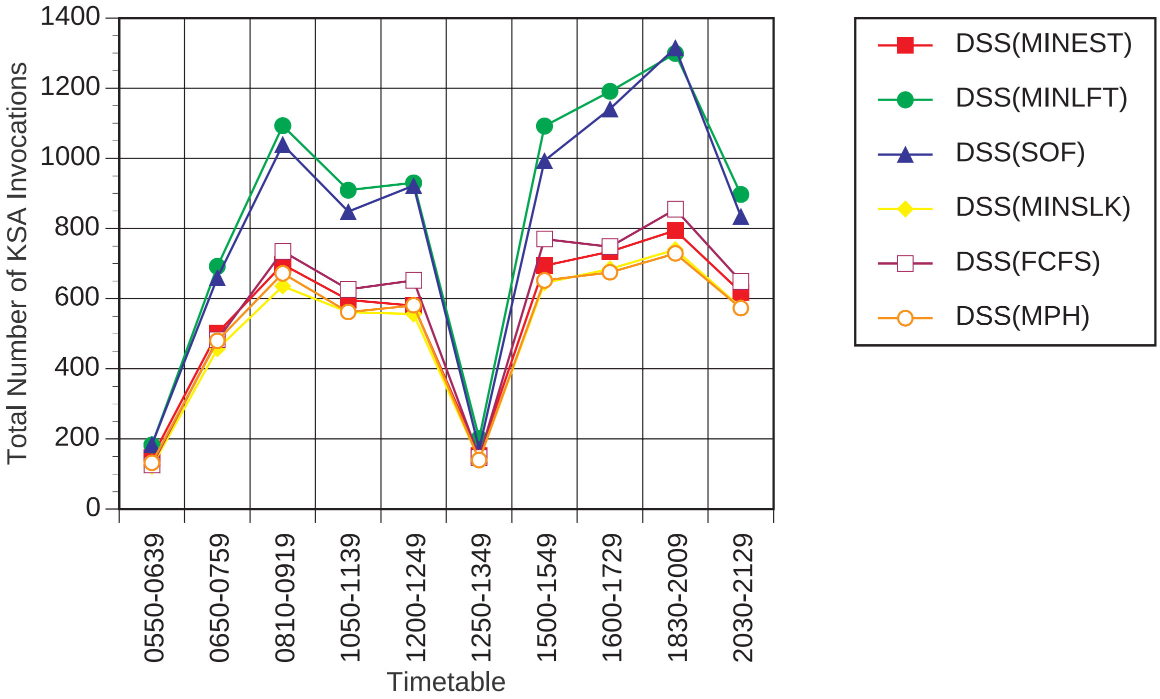


Figure 5.27. Comparison (via ARM) of the Total Number of KSA Invocations Among the Assorted DSS Heuristics Using Two Order Groups in Batch Scheduling Mode (Lower is Better).

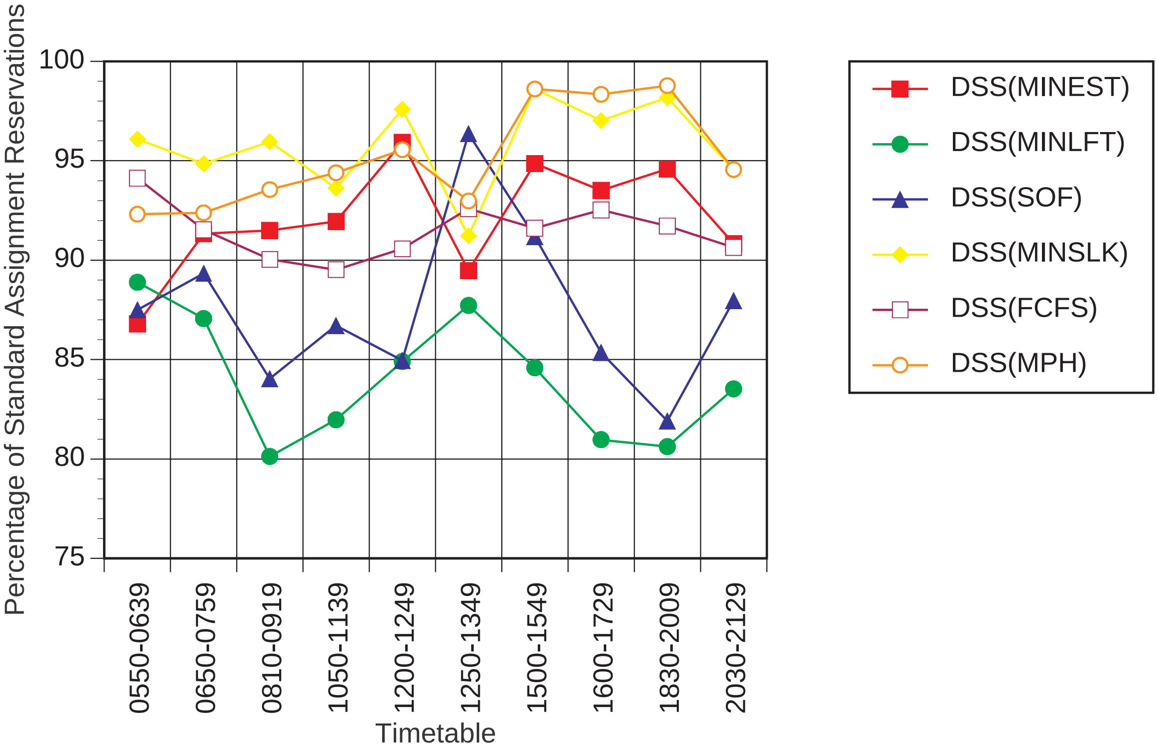


Figure 5.28. Comparison (via ARM) of the Percentage of Standard *Assignment* Reservations Among the Assorted DSS Heuristics Using Two Order Groups in Batch Scheduling Mode (Higher is Better).

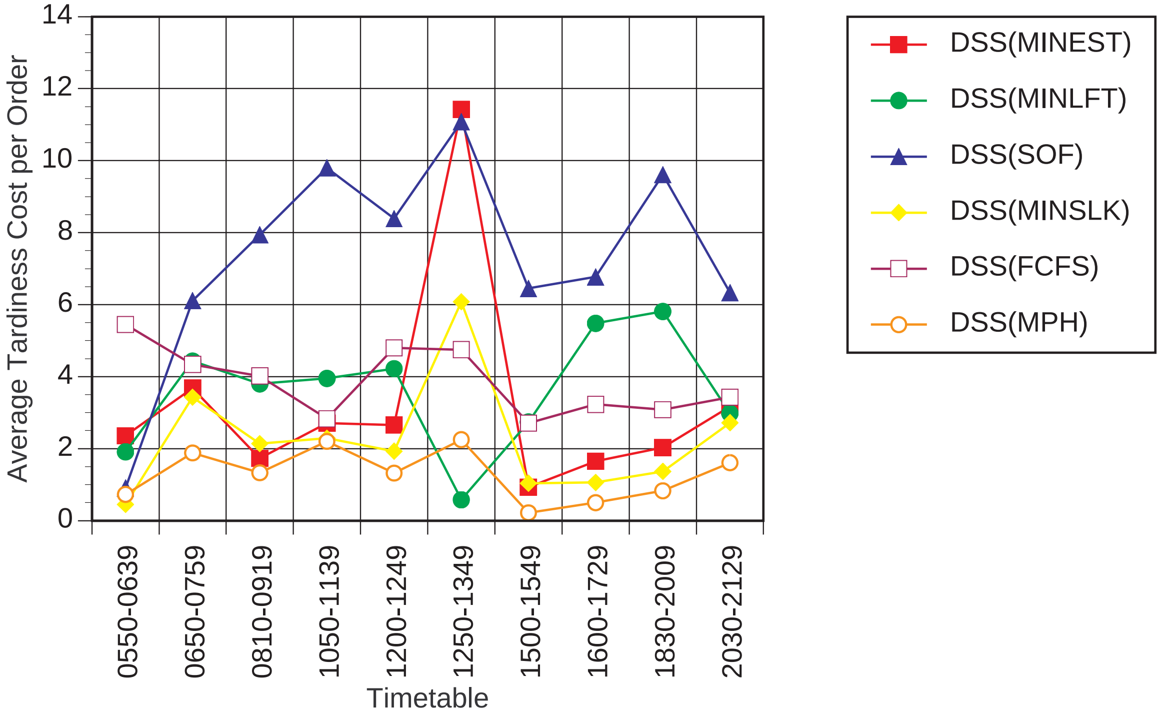


Figure 5.29. Comparison (via ARM) of Average Tardiness Cost per Order Among the Assorted DSS Heuristics Using Three Order Groups in Batch Scheduling Mode (Lower is Better).

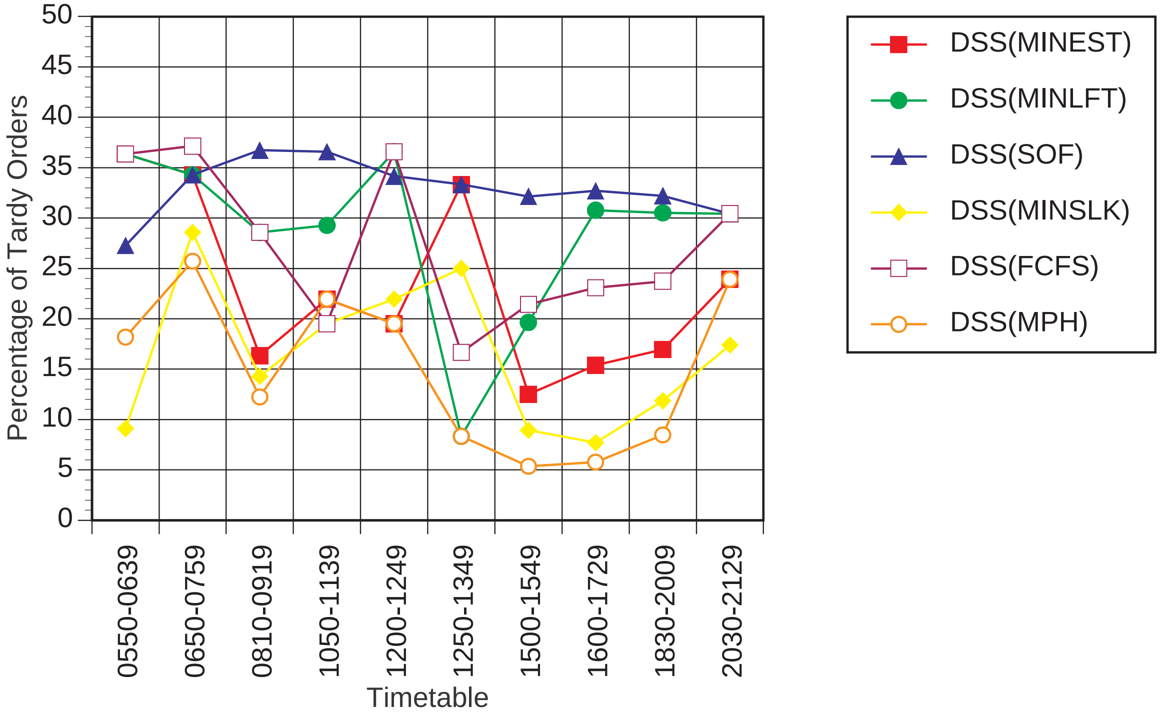


Figure 5.30. Comparison (via ARM) of the Percentage of Tardy Orders Among the Assorted DSS Heuristics Using Three Order Groups in Batch Scheduling Mode (Lower is Better).

5.7 A Summary

In this section we take the opportunity to summarize the various graphs presented in Sections 5.6.2 and 5.6.3. Table 5.7 provides a summary of the results from the dispatch scheduling experiments in Section 5.6.2.1. In the TCP results, DSS(MPH) achieves the lowest percentage of tardy orders in 10 of the 18 categories. Furthermore, the rankings for the total number of KSA invocations and the percentage of standard *Assignment* reservations are the best among the other heuristics. While the average tardiness cost ranking for DSS(MPH) is slightly below that of both DSS(MINEST) and DSS(MINLFT), it does produce the lowest average tardiness cost per order in one third of the categories. In the ARM results, the average tardiness cost ranking for DSS(MPH) improves slightly, while the percentage of tardy orders ranking slips below that of DSS(MINSLK). The computational efficiency rankings demonstrate the penalty

Table 5.2. Summary of the *Dispatch Scheduling* Experimental Results (via TCP and ARM) Showing the Average Ranking (Lower Number is Better) and Total Number of Best Finishes (Higher is Better) per Metric Achieved by the Assorted DSS Heuristics.

Heuristic Scheduling Approach	Average Tardiness Cost per Order	Percentage of Tardy Orders	Average Work-in-Process Time per Order	Total Number of KSA Invocations	Percentage of Standard *Assignment* Reservations	Elapsed Processing Time
Reactive Analysis: *Dispatch Scheduling* (TCP)						
DSS(MINEST)	2.39 (5)	2.89 (5)	2.39 (1)	2.56 (3)	2.44 (3)	4.06
DSS(MINLFT)	2.28 (4)	4.17 (1)	3.44 (2)	2.28 (4)	2.56 (2)	4.00
DSS(SOF)	6.00	5.56	5.22	5.94	6.00	2.56 (3)
DSS(MINSLK)	2.89 (3)	3.06 (3)	3.39 (2)	4.94	4.44	4.44
DSS(FCFS)	4.83	3.17 (2)	1.78 (13)	3.44 (3)	4.17	1.33 (15)
DSS(MPH)	2.56 (6)	1.50 (10)	4.78	1.83 (8)	1.39 (13)	4.61
Reactive Analysis: *Dispatch Scheduling* (ARM)						
DSS(MINEST)	3.7 (1)	2.9 (1)		3.0 (2)	3.2 (1)	2.2 (2)
DSS(MINLFT)	5.1	5.0 (1)		5.8	5.4	5.0
DSS(SOF)	5.2	4.6 (1)		5.2	5.2	4.4
DSS(MINSLK)	1.5 (7)	1.2 (9)		1.8 (5)	2.4 (3)	1.5 (6)
DSS(FCFS)	2.9 (2)	2.9 (3)		1.7 (4)	1.4 (7)	2.2 (3)
DSS(MPH)	1.9 (3)	2.0 (4)		3.2 (1)	2.6 (2)	5.6

incurred by DSS(MPH) as a result of the worst-case mobile resource reservation policy. DSS(MPH) drops consistently below both DSS(MINSLK) and DSS(FCFS) in both the total number of KSA invocations and the percentage of standard *Assignment* reservation rankings.

Table 5.7 provides a summary of the results from the resource failure experiments in Section 5.6.2.2. These results demonstrate the dramatic improvements achieved by

Table 5.3. Summary of *Coping with Resource Failures* Experimental Results (via TCP) Showing Average Ranking (Lower Number is Better) and Total Number of Best Finishes (Higher is Better) per Metric by the Assorted DSS Heuristics.

Heuristic Scheduling Approach	Average Tardiness Cost per Order	Percentage of Tardy Orders	Average Work-in-Process Time per Order	Total Number of KSA Invocations	Percentage of Standard *Assignment* Reservations	Elapsed Processing Time
Reactive Analysis: *Coping with Resource Failures* (TCP)						
DSS(MINEST)	4.67	3.67 (3)	4.89 (1)	5.22	5.22	4.50 (2)
DSS(MINLFT)	5.56	4.78 (1)	5.39	5.67	5.72	1.78 (8)
DSS(SOF)	3.44 (1)	4.06 (1)	4.11	3.06	3.44	3.94
DSS(MINSLK)	2.28 (2)	2.61 (4)	1.94 (7)	3.94	3.22	3.11 (1)
DSS(FCFS)	3.67	2.50 (5)	2.67 (1)	1.83 (3)	2.22	2.22 (7)
DSS(MPH)	1.33 (15)	2.56 (6)	2.00 (9)	1.17 (15)	1.00 (18)	5.44

DSS(MPH) over the previous dispatch scheduling results, in terms of both schedule quality and computational objectives. DSS(MPH) produces the lowest average tardiness cost values in 15 of the 18 categories, and ranks second in percentage of tardy orders (producing the lowest percentage in a third of the categories). The computational efficiency rankings are well above those of the other heuristics, with DSS(MPH) using the fewest number of KSA invocations in 15 of the 18 categories, and achieving the highest percentage of standard *Assignment* reservations in all categories.

Table 5.7 provides a summary of the results from the unexpected order set experiments in Section 5.6.3. In each of these experiments, DSS(MPH) outperforms the other heuristics by producing the lowest average tardiness costs, and ranks either first or second (trading off with DSS(MINSLK)) in terms of producing the lowest percentage of tardy orders. The computational efficiency results are only slightly below those of DSS(MINSLK). Throughout these experiments, the remaining heuristics, namely DSS(MINEST), DSS(MINLFT), DSS(SOF), and DSS(FCFS), rank significantly lower than both DSS(MINSLK) and DSS(MPH).

Throughout this section we have evaluated the performance of the DSS scheduling approach in two substantially different RCSP domains. We have shown how the preservation of slack time allows DSS to preserve its future scheduling options, how its fine-grained opportunism

Table 5.4. Summary of *Accommodating Unexpected Order Sets* Experimental Results (via ARM) Showing Average Ranking (Lower Number is Better) and Total Number of Best Finishes (Higher is Better) per Metric by the Assorted DSS Heuristics.

Heuristic Scheduling Approach	Average Tardiness Cost per Order	Percentage of Tardy Orders	Total Number of KSA Invocations	Percentage of Standard *Assignment* Reservations
Accommodating Unexpected Order Sets: *Two Order Groups* (ARM)				
DSS(MINEST)	3.5	3.2 (1)	3.2	3.5
DSS(MINLFT)	4.5	4.9	5.8	5.8
DSS(SOF)	5.8	4.3 (1)	5.2	4.6 (1)
DSS(MINSLK)	1.7 (3)	1.3 (7)	1.4 (6)	1.7 (5)
DSS(FCFS)	3.8	4.3 (1)	3.6	3.6
DSS(MPH)	1.5 (7)	1.5 (6)	1.7 (5)	1.7 (5)
Accommodating Unexpected Order Sets: *Three Order Groups* (ARM)				
DSS(MINEST)	3.4	3.0 (1)	3.5	4.1
DSS(MINLFT)	4.1 (1)	4.0 (1)	5.6	5.5
DSS(SOF)	5.6	4.9	5.4	5.0
DSS(MINSLK)	2.3 (1)	2.0 (3)	1.4 (7)	1.7 (5)
DSS(FCFS)	4.4	4.0 (1)	2.9 (1)	2.6 (3)
DSS(MPH)	1.2 (8)	1.4 (7)	2.1 (2)	2.1 (2)

enables it to react quickly and effectively to dynamic environments, and how its frequent consultation of multiple scheduling perspectives provides it with the means to understand fully the current state of problem solving. DSS is therefore able to organize its decision-making strategy to both decrease tardiness and increase computational efficiency while responding to the unexpected changes that occur in dynamic, real-world environments. Finally, we have demonstrated the applicability of DSS to a variety of scheduling domains, and verified the consistency of its performance across these domains.

CHAPTER 6

CONCLUSIONS

In this chapter we review the research issues addressed by our work with DSS, and reiterate our contributions to the field of knowledge-based scheduling. We also discuss a number of important directions for future research, concluding with some brief closing remarks.

6.1 Research Issues Revisited

Throughout the course of this research, we have focused on the idea of exploiting flexibility in the scheduling process to deal with the uncertainties of dynamic environments. It remains our claim that the difficulties faced in the process of scheduling within real-world environments can be eased by approaching the problem with a high level of understanding and opportunism, and the intention of preserving solution options for producing high quality results and limiting problem-solving effort.

We began by noting both the questionable real-world value of predictively optimal schedules, and the difficulty involved in trying to adapt such schedules to fit changing environmental conditions. In real-world situations, the order set develops over time, affected by periodic additions and cancellations. The resource supply is unstable, owing to the tendency for machines to break down. Basing a long-term scheduling strategy upon a preliminary analysis of the state of problem solving is therefore of little real value.

Besides affecting the ability to anticipate future developments, the dynamic nature of the scheduling problem also places a premium on the ability of the scheduler to adjust its focus of attention quickly and properly in response to the changing state of problem solving. It must be able to understand the actual situation, and then adjust its decision-making strategy accordingly. It must also be able to react to the new situation without causing too much disruption to the existing schedule.

Detection is an important precursor to reaction. Consultation of multiple relevant perspectives provides a scheduler with the ability to understand the often subtle textures of the problem space, and thereby permit it to detect potential conflicts before they become too serious. Such a flexible decision-making capability prevents the system from having to make myopic traversals through the developing search space. Once the decision is made to modify the course of problem solving to conform to a new decision-making strategy, it is imperative that the system be able to institute the necessary alterations before conditions in the environment change again. Opportunism enables reaction while conditions are still stable. Finally, the scheduler must be equipped with problem-solving mechanisms that can anticipate future conflicts and actively attempt to avoid them. By preserving flexibility, conflicts can be solved more easily when the need arises.

These sources of flexibility in the scheduling process have each been addressed by the approach implemented in DSS.

6.2 Contributions Revisited

In this section, we review the specific contributions to the field of knowledge-based scheduling that have resulted from our work with DSS.

6.2.1 Developing A Multi-Faceted Approach to Solving Dynamic Resource-Constrained Scheduling Problems

DSS brings together a number of important features to create a flexible approach for solving dynamic RCSPs. The approach is notable for its *simultaneous* preservation of flexibility within developing schedules, flexibility in adapting to the dynamics of real-world environments, and flexibility in understanding the varying textures of the developing state of problem solving. Our design for DSS has been fully implemented, as described in Chapters 3 and 4, and extensively evaluated, as described in Chapter 5.

6.2.1.1 Use of Slack Time to Preserve Flexibility and Limit Schedule Disruption

In our experiments using the OPIS benchmark data and the DSS-based TCP scheduling application, we have demonstrated the value of making decisions that preserve future scheduling options, for use in resolving the conflicts that develop throughout the scheduling process. The flexibility that is maintained by this approach, in the form of explicitly represented slack time, preserves choices that help to minimize the loss of schedule quality that can result from having to settle for the least-objectionable among a limited number of unattractive options. Using this approach, DSS is able to produce higher quality schedules, in terms of minimizing both the average tardiness cost per order, and the percentage of tardy orders. Additionally, the preservation of flexibility allows for the more frequent use of less disruptive decision-making mechanisms that execute quickly, without causing substantial, and often costly, constraint propagation. The results presented in Section 5.6.1, where DSS outperformed both ISIS and the initial OPIS system in minimizing tardiness, even when using limited, classical scheduling heuristics, indicate that the sacrifice of minimal flow times for the sake of maintaining flexibility, is entirely worthwhile.

6.2.1.2 Quick and Effective Reaction to Dynamic Environments

In response to the rigid approaches of existing knowledge-based scheduling systems, we have designed the DSS scheduling approach to be highly opportunistic in directing its focus of attention during problem solving, specifically in its resolution of the conflicts triggered by unexpected events. The experiments in Section 5.6.2 demonstrate the ability of DSS to adapt to unexpected events by determining their extent quickly, and then integrating the means for their solution into the existing problem-solving agenda. The reliance on special-purpose resolution mechanisms that automatically suspend potentially urgent scheduling activities regardless of the severity of a conflict, is thereby avoided. The importance of dealing with a specific conflict is determined according to the relative urgency of all currently outstanding activities.

6.2.1.3 Consultation of Multiple Perspectives in All Scheduling and Control Decisions

In Section 4.2.2.2 we identified a number of key decision-making points in the scheduling process that provide the opportunity for a scheduler to detect changes in the current state of problem solving, and develop an informed view of the overall problem being addressed. In Section 1.4.3.3 we described a broad range of scheduling perspectives that can be brought to bear at these decision-making points. We have developed a scheduling approach that considers a broad range of relevant information throughout the decision-making process, to help produce quality schedules and limit the required computational effort.

The results of our experiments in Section 5.6 indicate that when a scheduler's variable-ordering and value-ordering decisions are based on a limited consultation of perspectives, less information is brought to bear on the problem at hand, causing the scheduler to misdirect its efforts, generate lower quality solutions, and increase the probability that whatever decisions it does make will have to be reconsidered and corrected at a later time. Our approach was tested in dynamic situations where unexpected events such as new orders and resource failures occur, thereby placing a premium on the ability of the scheduler to maintain an accurate view of the changing conditions within the environment.

6.2.1.4 Accommodation of Additional Domain Complexities

As part of our work on the development of DSS, we have attempted to address some of the common complexities found in RCSP domains, which have been ignored by existing knowledge-based scheduling systems. In particular, the airport ground service scheduling domain requires the management of both inter-order tasks, and mobile resources with significant travel requirements. We have developed a representation scheme that accommodates task-interconnection, and thereby permits a greater level of interdependence among otherwise separate orders. Detailed examples of the ARM system (including the necessary inter-order task specifications for the AGSS domain) can be found in Section A.1. Mobile resources with significant travel times present additional computational requirements for DSS. The reservation-securing mechanisms described in Section 4.2.4.1 must consider travel requirements in the construction of mobile resource reservations, and the refinement mechanism described in Section 4.2.6 is required for collapsing earlier mobile resource reservations made to preserve flexibility under uncertain conditions. Experiments with the ARM system described in Sections 5.6.2.1 and 5.6.3 demonstrate the functionality of these various mechanisms.

6.2.2 Demonstrating the Generic Capabilities of DSS

The applicability of DSS to a wide variety of dynamic RCSPs has been established by its ability to represent a number of diverse scheduling domains, including the following.

- The airport ground service scheduling domain implemented by ARM (the *Airport Resource Management* system) and described in Section A.1
- A variety of job-shop scheduling domains, including the OPIS benchmark domain implemented by TCP (the *Turbine Component Plant* scheduling system) and described Section A.2
- A large-scale transportation planning domain (not described in this document)

DSS critics blame lack of resources for failure

Boston Globe

Report finds 'breakdown' at DSS, calls for overhaul

Boston Globe

Specialists' study says DSS in need of a restructuring

Boston Globe

Figure 6.1. Miscellaneous Newspaper Headlines (Seemingly) Unrelated to the *Dynamic Scheduling System*.

Our goal has been to ensure that the techniques used by DSS to represent dynamic RCSPs are sufficient for representing the information that defines a variety of RCSP domains, and that its collection of generic scheduling knowledge sources is sufficient for reasoning with this information to produce high-quality schedules. Our experience has shown that with surprisingly little difficulty, DSS can be successfully applied to a variety of dynamic RCSP domains. The results presented in Section 5.6 show that DSS can produce quality schedules in each of those domains.

6.3 Directions for Future Research

In addition to the many improvements in efficiency that can be achieved for the DSS implementation by means of some thoughtful reprogramming, there is plenty of room for future research involving both DSS and the solution of dynamic RCSPs. (This may or may not be evident from the headlines in Figure 6.1).[1] In this section we describe a number of the important directions for future research that we can envision for improving the performance of DSS and extending its scheduling capabilities.

6.3.1 Improving the Management and Utilization of Slack Time

There is a great deal of work that can be done involving the issue of slack time, in terms of where, when, and how it should be maintained, which could have a major impact on the ability of DSS to minimize its computational effort. When flexibility is provided in the right place, the scheduler gains access to a wider range of streamlined solution options that help minimize schedule disruption.

The current approach used by DSS to maintain slack time within developing order schedules works by first determining the total amount of time available to process the order, as indicated by the target lead time (the difference between the ready time and the due date). The actual amount of available slack is then determined by subtracting the length of the order's critical path from the target lead time. Slack is then allocated to the various resource-requiring tasks

[1] In Massachusetts, DSS also stands for the Department of Social Services.

based on their earliest and latest starting and finishing times. The slack creates a window within which a resource can be secured for a particular task without disrupting other parts of an order's schedule. One possible modification would be to use the existing and anticipated levels of resource contention in deciding where slack time is allocated, in that operations requiring bottleneck resources would benefit from larger allowances.

Remember that slack time is a shared commodity. Several tasks within an order can each lay claim to the same amount of slack time. Currently, the first of these tasks for which a reservation is secured can use that slack as desired, owing to its higher degree of urgency. That task relinquishes control of the slack to the other tasks down the line, following the introduction of its own reservation. It may, however, use a large enough portion of the existing slack to significantly affect the urgency of its neighboring tasks (all of which must share the slack). This situation suggests another factor that should be considered in the reservation-securing process, namely the degree of constraint that accompanies the reservation, upon its incorporation into the schedule.

Finally, the reservation-securing knowledge sources could be provided with the option of occasionally including slack time as part of its reservations, to prepare for situations where future difficulties involving a task or particular resource class are anticipated. The process of maintaining slack time could therefore be more closely controlled, to provide flexibility in the places where it is needed most.

On a somewhat different level, there is also the question of how much slack is required to provide the necessary flexibility for a given scheduling situation. At the moment, the least-commitment approach taken by DSS prohibits full schedule compaction. In cases where the compaction of the order schedules is a desired objective, the amount of available slack could be decreased, by collapsing the target due dates, to encourage the earlier delivery of jobs. The key issue to determine is the value of the scheduling flexibility that is afforded through the preservation of slack. Questions such as, how much computational efficiency is lost by decreasing the amount of available slack, and whether the loss of flexibility is compensated by the compacted results, are important to answer. Conditions in a dynamic real-world environment play an important role in determining the answers to these questions. An investigation into the impact on schedule quality and computational efficiency caused by systematically decreasing the amounts of available slack time is an important first step in this process.

6.3.2 Developing an Adaptive Scheduling Strategy

The class of dynamic RCSPs contains a wide range of problems. These problems differ according to their desired objectives, the nature of their orders, and the availability and reliability of their resources. In some cases, orders may need to be compacted to minimize flow times and deliver products as early as possible, while in others, the orders may need to meet their desired ready times and due dates exactly. Resource supply may or may not be capable of meeting anticipated demand. Finally, while it is sometimes possible for a schedule to be completed in advance of its intended execution, more often the production of the schedule must take place within a dynamic environment, requiring a scheduler to react to unexpected developments while still in the process of building the schedule.

A dynamic scheduling problem is an evolving entity that changes with the pattern of order reception, the availability of resources, the scheduling objectives, and the conditions within the environment, all of which may change during the course of scheduling. Previous work on classical, heuristic-based approaches for solving related job-shop and factory scheduling problems has led to the observation that no single scheduling strategy is appropriate for all problems [Davis and Patterson, 1975]. While one heuristic may work well in a particular situation, it may perform abysmally in another. Quite simply, different situations call for different strategies. Our own experiences in producing schedules for a variety of domains, including the airport ground service scheduling domain, a standard job-shop, and a large-scale transportation planning environment, have led us to the same conclusion.

An important direction for future research would be to try and identify the various aspects of a dynamic RCSP that suggest the use of a particular scheduling strategy, given the current scheduling objectives. The goal of this endeavor would be the development of the mechanisms necessary for both assessing the current state of problem solving at any point, and modifying the current scheduling strategy as necessary. For example, the current level of resource contention provides information that can be used to choose among different scheduling strategies. If resources are in abundant supply, then scheduling decisions might be tailored towards the optimization of order schedules. When resource contention is high, on the other hand, the focus might shift towards the optimization of the schedules for bottlenecked resources. The result would be a scheduling approach that is truly reactive in every sense of the word.

Another aspect of this particular research direction is the issue of learning. It is possible that DSS could be engineered to record and classify the important attributes of the current state of problem solving, and then develop a model for indicating which scheduling strategies perform the best under different conditions. Case-based reasoning techniques would certainly be applicable to this process.

Finally, determining the appropriate scheduling strategy for any situation still depends on having a full understanding of the current state of problem solving. Just as in the current implementation of DSS, the consultation of multiple, relevant scheduling perspectives would assist in the process of informing the scheduler about specific conditions in the environment. While we have taken an important first step in this direction with the value-ordering heuristics in the DSS reservation-securing knowledge sources, there is still much more to be done.

6.3.3 Accommodating Resource-Based Schedule Quality Objectives

Currently, the DSS schedule quality objectives are all order-based. The minimization of tardiness focuses on the due dates for the orders, as does the minimization of flow times (which we have not yet implemented). Missing from the notion of schedule quality in DSS is the degree of resource utilization, including the minimization of external fragmentation in the resource schedules, the minimization of travel time for mobile resources, and the minimization of the incidence of optional setup activities. Although each of these concerns is partially addressed by the value-ordering heuristics in the current implementation of DSS's reservation-securing knowledge sources, their optimization is not an overriding priority of the DSS variable-ordering process.

An early attempt to represent resource-based schedule quality concerns in DSS involved the creation of *resource goals* responsible for voicing the concerns of the resources throughout the decision-making process. The intent was for these goals to contribute to the urgency of certain subproblems depending on conditions within the environment related to the resources. Resources with specific openings in their schedules to accommodate certain outstanding subproblems would be able to attract the attention of the scheduler in an attempt to target those subproblems for the openings. These resource goals would also be capable of assisting the value-ordering process by contributing better-enhanced resource-centered perspectives. The ability to address both order-based and resource-based schedule quality objectives would represent an important enhancement to the overall functionality of DSS.

6.3.4 Avoiding Worst-Case Mobile Resource Reservations

A DSS domain description is required to include information about the anticipated behavior of all classes of mobile resources operating in a factory environment, specifically their basic progression through factory locations in the course of performing their required servicing activities. This information is used by the domain-supplied duration-calculation methods to provide DSS with both the expected and exact processing durations for each task to be performed, based on a specific resource (if provided), and the product and order involved. When calculating processing durations that involve the movement of mobile resources, these methods must include enough time for a mobile resource to get from any location at which it may initially reside to any location at which it may be needed. This process provides DSS with reservations that do not constrain the selection of the stationary resources that will provide the exact origin and destination locations for any previously secured mobile resources. As a result, the options for securing these work areas in the future are not further limited by constrictive mobile resource travel ranges.

The scheduling process in DSS is then augmented by a reservation-refinement mechanism that is triggered whenever additional contextual information regarding a mobile resource reservation is determined, generally as the result of securing a stationary resource that provides a work area to (or from) which the mobile resource must report. These relationships are implicit in the domain description. The reservation-refinement process is responsible for compacting mobile resource reservations made under uncertain conditions. The flexibility made available to the scheduler for securing stationary work area resources is thus preserved until it is no longer needed, at which point it is released and made available for use in satisfying other outstanding resource requests.

Experiments with DSS, however, such as those described in Section 5.6.2.1 involving the ARM system, demonstrate the price that is paid by using this worst-case mobile resource reservation approach. Oversized reservations make it more difficult to satisfy outstanding mobile resource requests, which in turn makes it more difficult for DSS to minimize tardiness. A solution to this problem would be to incorporate travel time into mobile resource reservations depending on the availability of both the mobile and stationary resource classes involved. If, for example, the stationary resources are in abundant supply, then the mobile resource reservations need not include maximal travel time allocations. In the case where the supply of mobile resources is low, then the mobile resource reservations should include only the minimal travel time allocations. Different allocation procedures should be put into place for different conditions. The goal of avoiding the tendency to over-constrain outstanding subproblems could still be maintained, but it would be tempered by the desire to improve the overall performance of the system through a more careful mobile resource allocation process.

6.3.5 Enhancing the Least-Commitment Decision-Making Process

In relation to the topic of the preceding subsection, the process of preserving scheduling options for the satisfaction of future subproblems needs to be extended to consider the impact of equipment compatibility constraints and technological constraints in addition to just temporal concerns. Currently, the worst-case reservation of mobile resources preserves the scheduling options for solving outstanding stationary resource subproblems. As evidenced by the AGSS domain, however, there are equipment compatibility constraints that can affect the ability to secure future reservations. Consider the AGSS refueling operation. Given a choice between using a pump truck or a fuel truck, an important concern is whether the gate at which the refueling is to occur is equipped with the necessary equipment to support the use of a pump truck. A fuel truck can be used at any gate, so if a fuel truck is secured first, there is no impact on the future ability to secure a gate. If, on the other hand, a pump truck is secured first, then the impact on the future ability to secure a gate is constrained according to the supply of gates with underground fuel tanks that can support the use of the pump truck. Assuming that this set of gates is a subset of the entire set of gates, the decision to use the pump truck has acted to limit the choice of a gate.

The problem can be extended to consider the similar impact of technological constraints between products and resources. Returning to the AGSS domain, consider the situation where two adjacent gates must share overlapping work space, such that the assignment of a large aircraft to one gate limits the size of aircraft that can be serviced at the other gate during the same period of time. In this case, the assignment of the first gate to an order involving a large flight acts to limit the set of gates that can be assigned to other orders with large aircraft.

Both of these problems warrant the attention of our attempts to preserve flexibility within the developing schedule. A better-informed decision-making process, able to anticipate accurately the impact of its scheduling decisions, would contribute greatly to the computational efficiency of the approach.

6.3.6 Introducing Planning Issues

The applicability of DSS to an even wider range of real-world dynamic scheduling environments would be enhanced by incorporating some basic planning abilities into its scheduling approach. Currently, DSS is unable to modify process routings during the scheduling process. Regardless of conditions within the environment, DSS is forced to schedule operations according to the static process routing templates provided by the domain descriptions. If conditions in the environment change drastically during the scheduling process, DSS has no ability to modify the order set on its own.

An important capability for DSS would be to allow it some flexibility in dealing with the process routing templates. Consider the situation where a particular resource class is in extremely short supply. And suppose that an operation requiring that type of resource is not an integral part of its process routing. DSS should have the option of removing that task from the queue of outstanding subproblems, at perhaps some expense of schedule quality, so that other more important subproblems can be addressed. Given clear scheduling objectives, such flexibility would improve the ability of DSS to adapt to a greater variety of dynamic situations.

6.3.7 Enhancing the Urgency-Determination Mechanism

There are a number of directions that could be taken to enhance the urgency-determination mechanism in DSS.

Presently, DSS considers only the existing demand for resources when generating its resource-based perspective in the process of determining subproblem urgency. The known rate of failure per resource class (if available), would provide additional valuable information about the stability of the resource supply, and thereby act to modify assumptions about the bottleneck status of the resources. If a particular resource class, in good supply, suffers from a high failure rate, its anticipated level of supply should be discounted to account for expected failures, to prevent subproblems requiring that resource class from being arranged improperly on the queue of outstanding scheduling activities. If the failure rate is high, the urgency of those subproblems should be boosted accordingly.

Another consideration that could play an important part in the urgency-determination process would be the expected pattern of job arrivals to the system. Such information could be used to indicate time periods in which to expect high contention for certain resource classes, which cannot be detected from the existing order set. In many real-world scheduling domains, the pattern of job arrivals can be determined over time, and used to prepare a scheduler for certain difficult conditions that may occur. The AGSS domain implemented in the ARM system provides an example of when the order set is essentially known far in advance (often on a per-month basis) of the actual reception of the orders. By incorporating this knowledge, when available, into the scheduling process, DSS could bring the performance of its dispatch scheduling mode closer to that of its batch scheduling mode in the same environment.

6.3.8 Extending the Problem Definition

The real-world applicability of DSS would be enhanced by extending our current RCSP definition to include a number of important additional complexities.

The issue of resource capacity provides an obvious direction for enhancement of our RCSP definition. Currently, implicit resource capacity constraints in DSS allow only a single operation to be performed (or serviced) on any resource at the same time. Yet many scheduling domains involve resources that can accommodate a number of operations at once. In many transportation planning domains, where container resources, such as trucks, boats, or aircraft, are used to transport cargo between specific locations, the same container resource may be used to transport the cargo for a number of jobs. Adding a capacity dimension to the concept of resource availability would increase the variety of real-world domains in which DSS could be used.

The expected processing duration any task is currently assumed to remain constant during the course of scheduling. Conditions in the environment, however, can often have a substantial impact on the actual time it takes to perform certain tasks. In the AGSS domain, weather conditions can wreak havoc on a schedule by increasing the amount of time it takes to perform all ground servicing activities. The ability to modify the current assumptions about how long certain tasks will take to execute would permit DSS to better react to changing environmental conditions, by adjusting its scheduling strategy to account for the changes in resource demand and contention that occur when processing duration expectations are altered.

6.3.9 Distributing the Flexible Scheduling Approach

Many RCSPs can be treated as distributed problems. A set of independent, though basically cooperative, scheduling agents, each with its own resource supply, may share responsibility for a set of orders. Individual orders could be assigned to scheduling agents according to some organizational model, perhaps with each agent accepting complete responsibility for a subset of the orders, or working together with other agents on shared order sets. An overall schedule is completed through the collective actions of all scheduling agents.

When agents experience difficulty from a lack of resources, they may request resource loans from other agents to complete their own scheduling assignments. Issues of negotiation come into play in this environment, where each agent must now deal with external resource requests in addition to the needs of its own assigned subproblems. Agents must also decide when to ask other agents for resource loans instead of trying to manipulate their own schedules.

An important factor in the negotiation process is the issue of slack time. By including slack time along with the loan of a resource, that is, by loaning the resource for a time in excess of the time requested, lending agents can temporarily release their control over a resource, thereby allowing the borrowing agent more flexibility in the use of the resource. The more autonomy that is provided in this fashion, the lower the communication costs to each of the agents involved in the process. The inclusion of slack time within resource loans also depends on the local state of problem solving for the lending agent. If its own resources are in short supply, it is more likely to loan out resources for only the time requested, if at all.

A distributed version of DSS has already been constructed, using a distributed version of the AGSS domain [Neiman *et al.*, 1994]. The scheduling agents in this domain correspond to the various airlines operating out of a particular airport. Each airline is given its own supply of resources, but may be occasionally forced to request the loan of a resource from another airline. Many of the issues involving negotiation and slack time have yet to be addressed.

6.3.10 Evaluating DSS in Real-World Scheduling Environments

A final direction for future research is the application of DSS to an actual real-world scheduling domain. While we have experimented with a variety of different RCSPs, and simulated a number of dynamic environmental conditions, we have yet to experience the demands of a real scheduling environment. The operation of DSS within such an environment would provide important information about its ability to react quickly to changes, and produce schedules in a timely fashion.

6.4 Closing Remarks

We have implemented DSS as a means of evaluating our flexible, knowledge-based approach to solving dynamic RCSPs. While our approach has exhibited success applied to a number of scheduling environments under dynamic environmental conditions, there is considerable work yet to be done in enhancing both its effectiveness and applicability. Nevertheless, the current design and implementation of DSS stands as a solid base upon which to build.

APPENDIX A

SCHEDULING APPLICATION SYSTEMS BUILT WITH DSS

In this appendix, we describe two different RCSP domains, and the corresponding scheduling applications we have built on top of DSS to evaluate the effectiveness of our flexible scheduling approach. The purpose of these discussions is to provide general information about each of these RCSPs, and demonstrate how the mechanisms provided by DSS have been used to represent their required domain information.

In each of the following sections, we describe an RCSP domain and the corresponding application system we have implemented. We describe each application system by first explaining the basic assumptions about the problem, then presenting the resources, shop locations and products involved in the domain, and finally explaining the kinds of operations performed by the resources, and the sequences of these operations that are used to produce the desired products.

In the first section, we describe the ARM system, used for scheduling the airport ground servicing activities at a large airport. Section A.2 describes the TCP system, which implements the simplified job-shop scheduling domain used to evaluate the initial OPIS system.

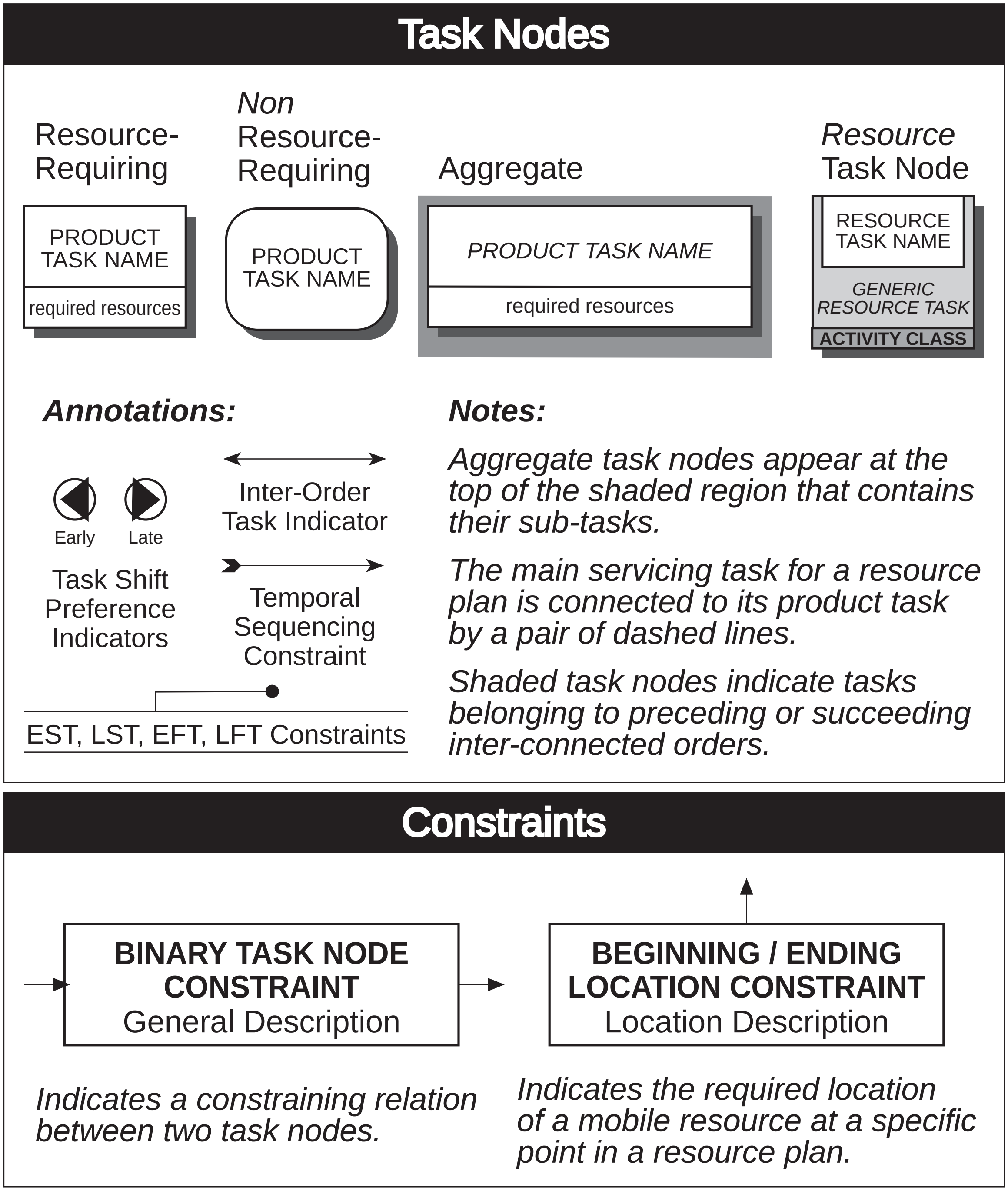
Figure A.1 provides a legend for understanding the resource and process plans presented throughout this chapter.

A.1 ARM*: The Airport Resource Management System*

The development and implementation of DSS originated from an investigation into the problem of scheduling the resources for performing the various required ground servicing activities at large airports. As it turns out, the *airport ground service scheduling* (AGSS) domain is an ideal example of a real-world dynamic RCSP. The scheduling objectives for the AGSS domain are common to many job-shop and factory scheduling environments. This observation has guided us in the process of designing DSS for use in representing and solving a broad range of RCSPs. Finally, the AGSS domain has provided us with some additional scheduling issues to consider, such as the use of mobile resources to perform some of the servicing tasks, and the existence of inter-order tasks.

A.1.1 The Airport Ground Service Scheduling Domain

Flight schedules are prepared periodically by airline companies. Gate and other resource capacities at the airports involved are considered in this process. Once the daily flight schedules are determined for an airline at a specific airport, local ground controllers generally take over the responsibility of assigning gates and other resources to the flights in advance of their arrival. When problems occur during the execution of a daily schedule, the ground controllers are



Aggregate task nodes appear at the top of the shaded region that contains their sub-tasks.

The main servicing task for a resource plan is connected to its product task by a pair of dashed lines.

Shaded task nodes indicate tasks belonging to preceding or succeeding inter-connected orders.

Indicates a constraining relation between two task nodes.

Indicates the required location of a mobile resource at a specific point in a resource plan.

Figure A.1. Key to Resource and Process Plan Figures.

responsible for quickly repairing the schedule to keep all of the scheduled ground servicing activities running as smoothly as possible.

More recently, many airlines have begun leasing entire airport terminals at specific airports (called *hubs*) through which to route most of their flights. The result is that many large airports now have one airline (or more) that uses a significantly large number of the airport's gates solely for its own flights. With the demand for passenger air travel remaining heavy, and airport building and expansion becoming ever more difficult and expensive to undertake, the problem of scheduling increasing numbers of flights using a generally static collection of resources becomes even more difficult to solve.

The ground servicing that is required for the aircraft that arrive and depart from an airport is extensive, involving the utilization of a variety of resources within a relatively short period of time. Passengers must be deplane and enplane, and their baggage must be loaded and unloaded from the aircraft, and transferred between connecting flights. In addition, all aircraft must be cleaned, restocked, refueled and serviced, while remaining docked at an assigned gate. The goal of a scheduler in this environment is to produce a schedule of resource usage that allows all of the required ground servicing activities to be performed in a timely manner to minimize flight delays, and to maximize the utilization of the resources (depending on the varying levels of resource contention per resource class). In the AGSS domain, unlike other job-shop scheduling domains, the penalties for earliness are just as severe as those for tardiness. The penalties for producing low quality schedules are measured in terms of dissatisfied customers in a highly competitive business environment, and the creation of idle time for very expensive aircraft and other equipment.

The related problem of scheduling crew assignments for an airline's flights is an interesting problem, which is not, however, within the scope of the problem we are focused on. Nevertheless, if we were to view the problem from a higher, possibly distributed level, and were to further enhance the constraint specification mechanism, it is conceivable that the assignment of crews to flights is a process similar to the assignment of gates to aircraft. For now, however, we leave the crew assignment problem out of our model of the AGSS domain.

To help experiment with and evaluate our approach to solving dynamic RCSPs, we have built, upon DSS, the *Airport Resource Management System* (ARM). ARM represents and solves dynamic RCSPs within the airport ground service scheduling domain. Throughout the rest of this appendix, we provide a detailed description of ARM, including descriptions of the resources and shop locations, products, operations, orders and process plans that define the dynamic RCSP that exists within the AGSS domain.

We begin our discussion of ARM with a statement of our basic assumptions about the AGSS domain.

A.1.2 Domain Assumptions

The problem of scheduling in the AGSS domain has received the attention of members of the operations research and expert systems fields. As a result, while the overall model of the domain has been the same, there has often been a difference in the kinds of problems actually addressed, and thus a difference in the way that the AGSS domain is represented.

We take the opportunity in this section to identify some aspects of the AGSS domain that we do not represent or consider within ARM. These items are discussed below.

- In ARM, the assignment of a particular kind of aircraft to a specific gate does not affect the range of use for any adjoining gates. That is, the process of assigning gates to aircraft does not consider whether or not a neighboring gate is in use, nor the kind of aircraft that may be docked there. In ARM, individual gate and aircraft compatibility relationships are handled independently of all other gate and aircraft pairings.

- Specific resource preferences for operations are not supported in ARM. Such preferences show up in the AGSS domain, for example, where it is often desirable to schedule certain daily flights at the same gate.

- ARM requires that all of the ground servicing activities for an aircraft be performed at a single gate. As a result, the ability to move an aircraft from the gate at which its arriving phase has been processed to another gate where its departing phase will be processed is not provided.

- The scope of the scheduling performed by ARM is limited to the activities occurring within the ground servicing areas of the airport. That is, ARM is not concerned with the sequencing of flight landings and departures, nor the assignment of runways and taxiways to aircraft. For the purposes of assigning gates, ARM assumes that arriving aircraft will land 5 minutes prior to their expected arrival (ready) times, and that departing aircraft will take off 5 minutes after their actual departure (completion) times. Changes in expected arrival (ready) times may force ARM to modify its existing schedule.

Unfortunately, this research was undertaken without the assistance of any of the large U.S. airline companies.[1] As a result, the entire ARM system is based on a reasonable approximation of the real-world AGSS domain. The flight schedules, the kinds of servicing tasks required and their processing durations, the supplies and attributes of the resources, and the dimensions of the basic airport layout used in our experiments are all based on realistic estimates.

A.1.3 Resources and Shop Locations

The AGSS domain as implemented in ARM includes seven resource classes, six of which are mobile. Five kinds of shop locations are also defined.

The single instance of the stationary resource class is the GATE. A gate is required for all of the ground servicing that is performed in the AGSS domain, because all scheduled activity is performed on an aircraft while it is docked at a gate. All mobile resources must therefore travel from gate to gate in the process of carrying out their assigned ground servicing tasks. Each gate provides a work area for the required mobile resources to perform their required ground servicing tasks, as well as the jetway through which passengers embark upon and disembark from the aircraft.

Gates may be restricted in the size of aircraft they are able to handle. Gate usage may be further restricted according to the types of government inspection services a gate is equipped to provide (for handling customs and related passenger processing), which determines whether or not it may be used for processing international flights. Gates without such facilities are

[1] The scheduling practices of the airlines can be classified as extremely proprietary.

limited to use by domestic flights only. The immediately preceding and succeeding stops for a flight are used to indicate whether it is international or domestic. Finally, some gates may be equipped with underground fuel storage systems that increase the number of refueling options available to the aircraft it services.

The six mobile resource classes required by our implementation of the AGSS domain in ARM are described below.

- FUEL TRUCK

  A fuel truck is used to refuel an aircraft prior to its departure. It may be used for any kind of aircraft or flight.

- PUMP TRUCK

  A pump truck is also used to refuel an aircraft prior to its departure. While a pump truck may be used on any kind of aircraft or flight, it may only be used in conjunction with a gate that is equipped with an underground fuel storage system. The selection of a pump truck to perform the refueling task for a flight therefore constrains the selection of a gate, while the selection of a gate may or may not constrain the selection of a truck to perform the refueling task.

- SERVICE TRUCK

  A service truck is used to perform the various miscellaneous mechanical servicing tasks necessary to prepare an aircraft for departure. It may be used for any kind of aircraft or flight.

- CLEANING TRUCK

  A cleaning truck is used to remove the range of waste products from an aircraft following its arrival. It may be used for any kind of aircraft or flight.

- CATERING TRUCK

  A catering truck is used to restock an aircraft with the necessary food and assorted supplies for its departure. It may be used for any kind of aircraft or flight.

- BAGGAGE TRUCK

  A baggage truck is used for loading and unloading the baggage from an aircraft, and for transferring baggage between connecting flights. A baggage truck may be used for any kind of aircraft or flight.

The five kinds of shop locations required by our implementation of the AGSS domain in ARM are described below.

- FUEL TANK

  The fuel tank is used for filling fuel trucks before they refuel an aircraft. Fuel trucks must report to the fuel tank at the beginning of every refueling assignment.

- KITCHEN

  The kitchen is used for filling catering trucks before they restock an aircraft. Catering trucks must report to the kitchen at the beginning of every restock assignment.

- BAGGAGE OUTLET

  Baggage outlets (there may be more than one) are used in the course of loading and unloading the baggage from an aircraft. Baggage trucks use the baggage outlet that is nearest to the gate assigned to the aircraft they must service. When unloading baggage, a baggage truck reports to the appropriate baggage outlet at the end of its assignment to unload its contents for their transfer to the baggage claim area. When loading baggage, a baggage truck reports to the appropriate baggage outlet at the beginning of the assignment to load all checked baggage from the airline check-in area. When transferring baggage between flights, a baggage outlet is not used.

- HANGAR

  The hangar provides a location for storing aircraft waiting to be used for *Departure* flights that originate at the airport.

- GARAGE

  The garage provides an additional location (besides the gates and the other shop locations) for housing mobile resources. Mobile resources are often initially placed in the garage in order not to influence the gate assignment process (for experimental purposes).

A.1.4 Products

The only products defined in ARM are the aircraft for the flights being serviced. For each flight requiring servicing, the ARM simulator creates a new aircraft of the appropriate type to be used for the flight. The type of aircraft created is based on a number of factors, such as the populations of the cities serviced by the flight, and the distance from the previous stop to the local airport whose ground service scheduling is being handled by ARM (and implicitly, DSS). There is a one-to-one mapping of aircraft to flights, that is, each aircraft is used for only a single flight, and each flight uses only a single aircraft. A total of 14 types of commercial aircraft are defined for ARM. The aircraft type plays an important role in the domain code that provides DSS with the required and estimated durations of the various ground servicing tasks.

A.1.5 Operations

In this section, we provide a description of the specific airport ground servicing tasks performed in the AGSS domain as implemented in ARM. For each of these tasks, we describe its overall purpose, the sequence of resource tasks that comprise its resource plan(s), and any of the constraints that it introduces.

A.1.5.1 Docking Support Activities for Arrival*,* Departure*, and* Turnaround *Flights*

These product tasks represent the required docking support activities provided by the *Arrival*, *Departure* and *Turnaround* flights (described in Section A.1.6). This support includes providing a work area for the required ground servicing to take place, as well as a jetway for getting passengers onto and off of the aircraft. Figure A.2 provides an illustration of the common resource plan shared by the ARRIVAL ACTIVITY, DEPARTURE ACTIVITY and TURNAROUND ACTIVITY product tasks. These tasks require the use of a GATE.

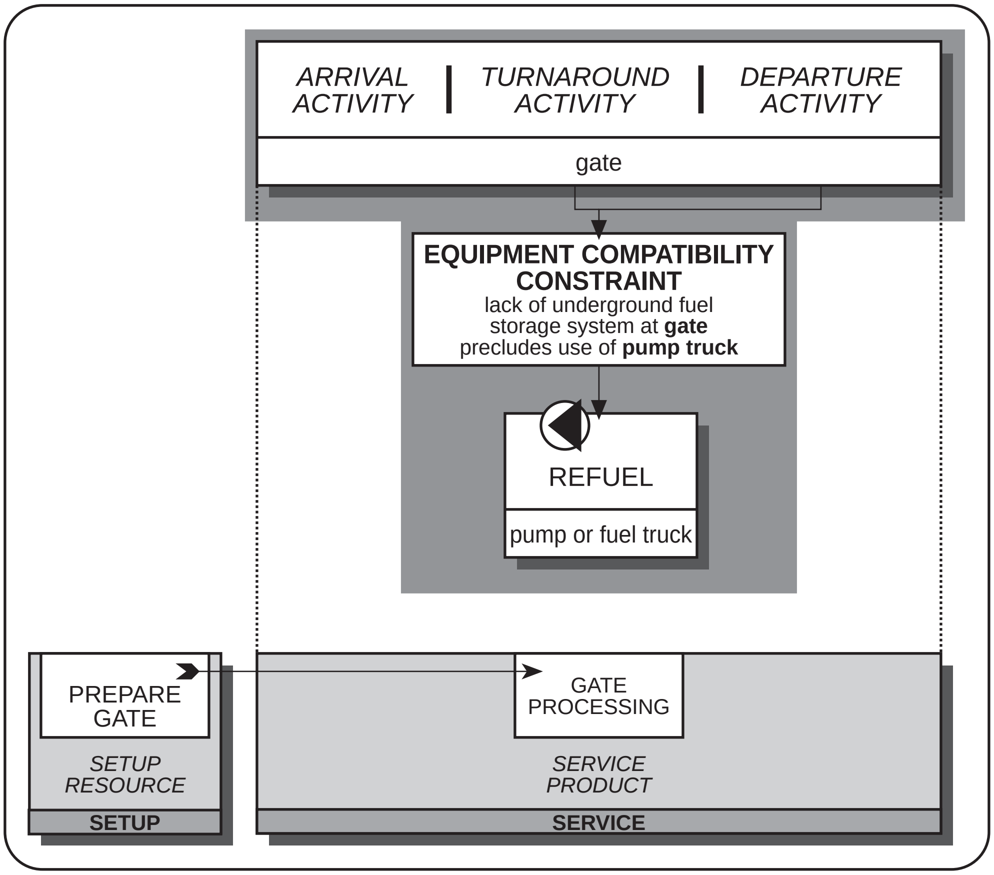


Figure A.2. Common Resource Plan for the ARRIVAL ACTIVITY, DEPARTURE ACTIVITY and TURNAROUND ACTIVITY Product Tasks.

The ARRIVAL ACTIVITY, DEPARTURE ACTIVITY and TURNAROUND ACTIVITY tasks are performed by means of a two-step resource plan. The `PREPARE-GATE` resource task prepares the gate for docking, with a constant duration of 5 minutes. The `GATE-PROCESSING` resource task encompasses the entire ground servicing process. These product tasks are aggregates, meaning that their durations are dependent on the desired and secured servicing times of their required child sub-tasks.

Figure A.3 shows the gate-related resource task specifications that are used to define the various flight servicing types described in Section A.1.6.[2] For the DEPARTURE ACTIVITY and

[2] The definitions of the ARRIVAL ACTIVITY, DEPARTURE ACTIVITY and TURNAROUND ACTIVITY aggregate

```
;;; ------------------------------------------------------------------
;;; Gate Activity Resource Task Definitions:

(define-resource-task PREPARE-GATE (setup-resource))

(define-resource-task GATE-PROCESSING (service-product))

;;; ------------------------------------------------------------------
```

Figure A.3. Resource Task Specifications for Gate-Related Operations.

TURNAROUND ACTIVITY product tasks, both of which require the refueling of an aircraft, the lack of an underground fuel storage system precludes the use of a pump truck to perform the refueling task. This is achieved with an EQUIPMENT-COMPATIBILITY-CONSTRAINT (ECC).

A.1.5.2 LOAD BAGGAGE

The LOAD BAGGAGE product task involves the movement of all baggage for the departing leg of a flight from the baggage outlet shop location nearest the gate servicing the flight to the departing aircraft, and the loading of that baggage onto the aircraft. Figure A.4 provides an illustration of the LOAD BAGGAGE task, including its resource plan and associated constraints. This task may only be performed by a BAGGAGE TRUCK.

The LOAD BAGGAGE product task is performed by means of a four-step resource plan. The GOTO-BAGGAGE-OUTLET resource task moves the baggage truck to the baggage outlet nearest the assigned gate, as specified by the BEGINNING-LOCATION-CONSTRAINT (BLC). If the truck is already at the baggage outlet, this resource task is not performed. At this point, all checked baggage is loaded onto the truck. The duration of the FILL-BAGGAGE-TRUCK resource task ranges from 6 to 18 minutes, depending on the capacity of the aircraft. The GOTO-GATE resource task moves the baggage truck to the assigned gate, at which point the baggage is loaded onto the aircraft. The duration of this FILL-AIRCRAFT resource task ranges from 6 to 12 minutes, again depending on the capacity of the aircraft. The ENDING-LOCATION-CONSTRAINT (ELC) indicates that the baggage truck remains at the gate upon completion of the LOAD BAGGAGE task.

The ANCHORED-DISTANCE-CONSTRAINT (ADC) ensures that the assigned gate is within the allocated travel distance from the nearest baggage outlet. The INITIAL-TRAVEL-DISTANCE-CONSTRAINT (ITDC) indirectly constrains the choice of a gate to service the flight, by limiting the set of usable baggage outlets (and hence, gates) to those within the allocated travel distance of the baggage truck's previous location.[3] The EST constraint indicates that the main activity of the LOAD BAGGAGE product task (LOAD-AIRCRAFT), may not begin any earlier than a time 30 minutes prior to the due date of the order (the expected departure time for the flight).

product tasks are provided in Section A.1.6, along with their corresponding service type descriptions.

[3] Note that the ITDCs, ADCs, DISTANCE-CONSTRAINTs, ECCs, BLCs, and ELCs are defined as part of the duration calculation methods for the resource task nodes. These constraints are instantiated and returned with the processing time durations calculated by these methods.

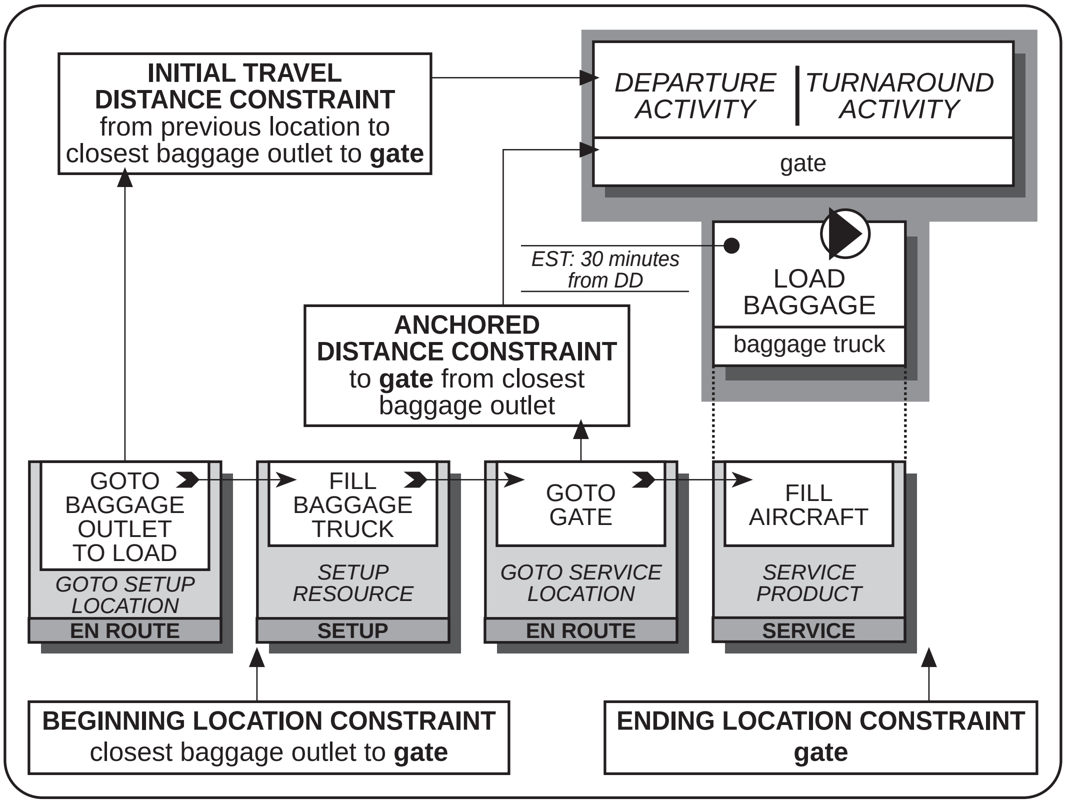

INITIAL TRAVEL DISTANCE CONSTRAINT
from previous location to closest baggage outlet to gate
DEPARTURE ACTIVITY
TURNAROUND ACTIVITY
gate
EST: 30 minutes from DD
LOAD BAGGAGE
baggage truck
ANCHORED DISTANCE CONSTRAINT
to gate from closest baggage outlet
GOTO BAGGAGE OUTLET TO LOAD
GOTO SETUP LOCATION
EN ROUTE
FILL BAGGAGE TRUCK
SETUP RESOURCE
SETUP
GOTO GATE
GOTO SERVICE LOCATION
EN ROUTE
FILL AIRCRAFT
SERVICE PRODUCT
SERVICE
BEGINNING LOCATION CONSTRAINT
closest baggage outlet to gate
ENDING LOCATION CONSTRAINT
gate

Figure A.4. Resource Plan for the LOAD BAGGAGE Product Task.

Figure A.5 shows the task specifications that define the LOAD BAGGAGE operation.[4]

A.1.5.3 UNLOAD BAGGAGE

The UNLOAD BAGGAGE product task involves the removal from the aircraft of all baggage destined for the local airport, and the movement of that baggage from the aircraft to the baggage outlet shop location nearest the gate servicing the flight. Figure A.6 provides an illustration of the UNLOAD BAGGAGE task, including its resource plan and associated constraints. This task may only be performed by a BAGGAGE TRUCK.

The UNLOAD BAGGAGE product task is performed by means of a four-step resource plan. The GOTO-GATE resource task moves the baggage truck to the assigned gate, as specified by the BLC. If the truck is already at the gate, this resource task is not performed. At this point, all baggage destined for the local airport is unloaded from the aircraft, and loaded onto the truck. The duration of the EMPTY-AIRCRAFT resource task ranges from 6 to 10 minutes, depending on the capacity of the aircraft. The GOTO-BAGGAGE-OUTLET-TO-UNLOAD resource task moves the baggage truck to the baggage outlet nearest the assigned gate, at which point the baggage is unloaded from the truck. The duration of the EMPTY-BAGGAGE-TRUCK resource

[4] For the purpose of completeness in these task specification figures, we have redundantly presented a number of generic task definitions that are shared by several of the ARM operations, and which actually appear only once in the actual specification files. (These shared operations are GOTO-GATE, REFUEL-AIRCRAFT, PREPARE-GATE, and GATE-PROCESSING).

```
;;; ------------------------------------------------------------------
;;; Load-Baggage Resource Task Definitions:

(define-resource-task GOTO-BAGGAGE-OUTLET-TO-LOAD (goto-setup-location)
   :NEXT-LOCATION
      (:CLOSEST-SHOP-LOCATION
         (:RESOURCE (turnaround-activity departure-activity) gate)
         baggage-outlet))

(define-resource-task FILL-BAGGAGE-TRUCK (setup-resource))

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity departure-activity) gate))

(define-resource-task LOAD-AIRCRAFT (service-product))

;;; ------------------------------------------------------------------
;;; Load-Baggage Product Task Definition:

(define-product-task LOAD-BAGGAGE
   :SHIFT-PREFERENCE :LATE
   :EARLIEST-START-TIME (:FROM-DUE-DATE 30)
   :RESOURCE-REQUIREMENTS
      ((baggage-truck
          :RESOURCE-PLAN
             (goto-baggage-outlet-to-load fill-baggage-truck
              goto-gate load-aircraft)
          :MAIN-ACTIVITY load-aircraft)))

;;; ------------------------------------------------------------------
```

Figure A.5. Task Specifications for the LOAD BAGGAGE Operation.

task ranges from 6 to 18 minutes, again depending on the capacity of the aircraft. The ELC indicates that the baggage truck remains at the baggage outlet upon completion of the UNLOAD BAGGAGE task. The ADC ensures that the assigned gate is within the allocated travel distance from the nearest baggage outlet. The ITDC indirectly constrains the choice of a gate to service the flight, by limiting the set of usable baggage outlets (and hence, gates) to those within the allocated travel distance from the baggage truck's previous location.

Figure A.7 shows the task specifications that define the UNLOAD BAGGAGE operation.

A.1.5.4 BAGGAGE TRANSFER

The BAGGAGE TRANSFER product task is an *inter-order* task that involves the transfer of baggage between the aircraft for two connected flights. It is an aggregate task with sub-tasks for unloading from the first aircraft all baggage destined for the departing leg of a specific connecting flight (UNLOAD X BAGGAGE), the transfer of that baggage between the gates servicing the two aircraft (TRANSFER X BAGGAGE), and the loading of the baggage onto the connecting flight aircraft (LOAD X BAGGAGE). Figure A.8 provides an illustration of the BAGGAGE TRANSFER task, including its resource plan and associated constraints. This task may only be performed by a BAGGAGE TRUCK.

The BAGGAGE TRANSFER task is performed by means of a two-step resource plan. The `GOTO-GATE` resource task moves the baggage truck to the assigned gate, as specified by the BLC. If the truck is already at the gate, this resource task is not performed. The

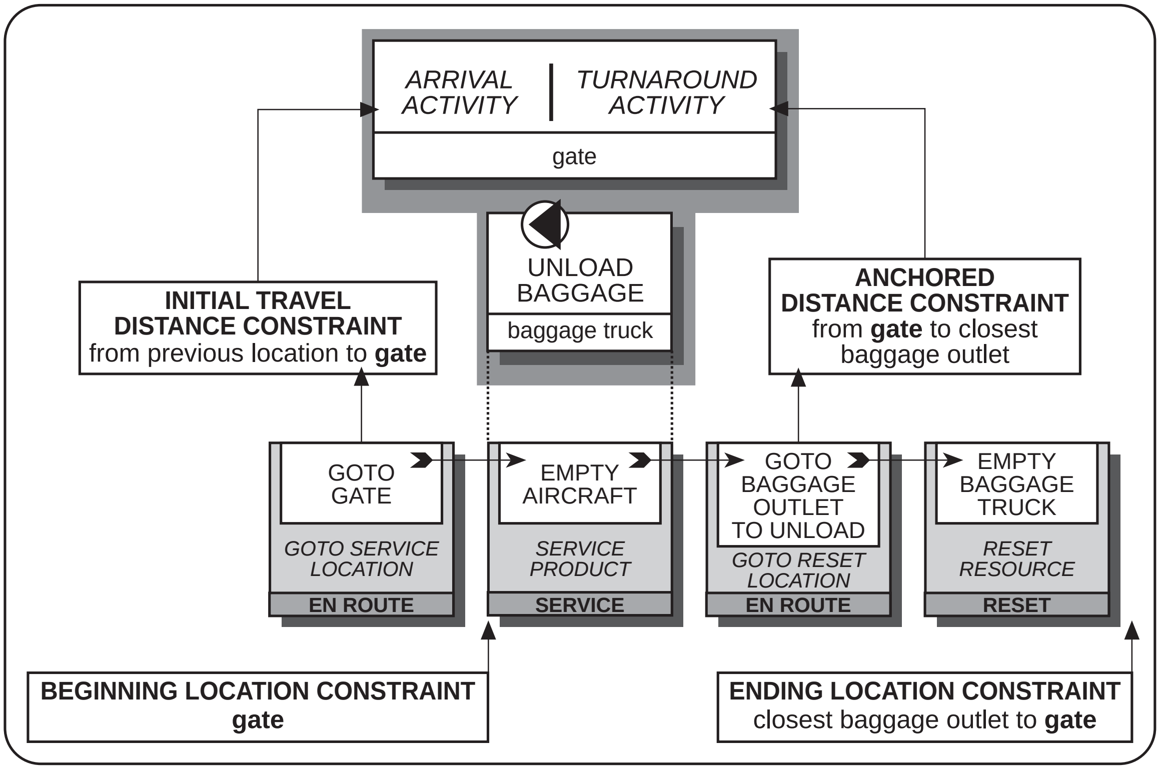

ARRIVAL ACTIVITY
TURNAROUND ACTIVITY
gate
UNLOAD BAGGAGE
baggage truck
INITIAL TRAVEL DISTANCE CONSTRAINT
from previous location to gate
ANCHORED DISTANCE CONSTRAINT
from gate to closest baggage outlet
GOTO GATE
GOTO SERVICE LOCATION
EN ROUTE
EMPTY AIRCRAFT
SERVICE PRODUCT
SERVICE
GOTO BAGGAGE OUTLET TO UNLOAD
GOTO RESET LOCATION
EN ROUTE
EMPTY BAGGAGE TRUCK
RESET RESOURCE
RESET
BEGINNING LOCATION CONSTRAINT
gate
ENDING LOCATION CONSTRAINT
closest baggage outlet to gate

Figure A.6. Resource Plan for the UNLOAD BAGGAGE Product Task.

BAGGAGE-TRANSFER-PROCESSING resource task encompasses the entire baggage transfer process. The ELC indicates that the baggage truck remains at the gate assigned to service the connecting flight aircraft upon completion of the BAGGAGE TRANSFER task. This product task is an aggregate, meaning that its duration is dependent on the desired servicing times of its required sub-tasks.

The ITDC constrains the choice of a gate to service the arriving flight, by limiting the set of usable gates to those within the allocated travel distance from the baggage truck's previous location. The DISTANCE-CONSTRAINTs imposed on each of the gates constrain the choice of gates for servicing the flights to those within the allocated travel distance of each other, as specified by the duration of the TRANSFER X BAGGAGE sub-task. The LFT constraint on the UNLOAD X BAGGAGE sub-task indicates that all of the transfer baggage destined for a connecting flight must be unloaded from the arriving flight aircraft by a time five minutes prior to the due date of the originating order (the time when its aircraft leaves its gate). The EST constraint on the LOAD X BAGGAGE sub-task indicates that all of the transfer baggage originating from a connecting flight may not be loaded onto the departing flight aircraft any earlier than a time five minutes after the ready time of the destination order (the time when its aircraft arrives at its gate).

The duration of the UNLOAD X BAGGAGE task ranges from 4 to 14 minutes, depending on the capacity of the originating aircraft. The duration of the LOAD X BAGGAGE task ranges from 5 to 16 minutes, depending on the capacity of the destination aircraft.

Figure A.9 shows the task specifications that define the BAGGAGE TRANSFER operation. The EXPECTED-DURATION-FUNCTION (including the MINIMUM and MAXIMUM varieties)

```
;;; ------------------------------------------------------------------
;;; Unload-Baggage Resource Task Definitions:

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity arrival-activity) gate))

(define-resource-task UNLOAD-AIRCRAFT (service-product))

(define-resource-task GOTO-BAGGAGE-OUTLET-TO-UNLOAD (goto-reset-location)
   :NEXT-LOCATION
      (:CLOSEST-SHOP-LOCATION
         (:RESOURCE (turnaround-activity arrival-activity) gate)
         baggage-outlet))

(define-resource-task EMPTY-BAGGAGE-TRUCK (reset-resource))

;;; ------------------------------------------------------------------
;;; Unload-Baggage Product Task Definition:

(define-product-task UNLOAD-BAGGAGE
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS
      ((baggage-truck
          :RESOURCE-PLAN
             (goto-gate unload-aircraft
              goto-baggage-outlet-to-unload empty-baggage-truck)
          :MAIN-ACTIVITY unload-aircraft)))

;;; ------------------------------------------------------------------
```

Figure A.7. Task Specifications for the UNLOAD BAGGAGE Operation.

provide DSS with the means to determine the expected, minimum, and maximum processing durations for non-resource-requiring tasks. The NEXT-LOCATION argument to the BAGGAGE-TRANSFER-PROCESSING resource task informs DSS that the baggage truck used to perform the baggage-transfer operation will end up at the gate used by the flight to which the baggage is being transferred (either a *Departure* or *Turnaround* flight). The non-nil value for the SPECIAL-REFINEMENT-CASE? flag indicates that the duration of the TRANSFER X BAGGAGE product task must be recalculated whenever the origin or destination of the transfer is updated (this flag is described in Section 4.2.6).

A.1.5.5 CLEAN

The CLEAN product task involves the removal of waste products and the general cleaning of the aircraft. Figure A.10 provides an illustration of the CLEAN task, including its resource plan and associated constraints. This task is only performed by a CLEANING TRUCK.

The CLEAN product task is performed by means of a two-step resource plan. The GOTO-GATE resource task moves the cleaning truck to the assigned gate, as specified by the BLC. If the truck is already at the gate, this resource task is not performed. Once the truck is at the gate, the cleaning process may begin. The duration of the CLEAN-AIRCRAFT resource task ranges from 5 to 15 minutes, depending on the capacity of the aircraft. The ELC indicates that the cleaning truck remains at the gate upon completion of the CLEAN task. The ITDC constrains the choice of a gate to service the flight, by limiting the set of usable gates to those within the allocated travel distance from the cleaning truck's previous location.

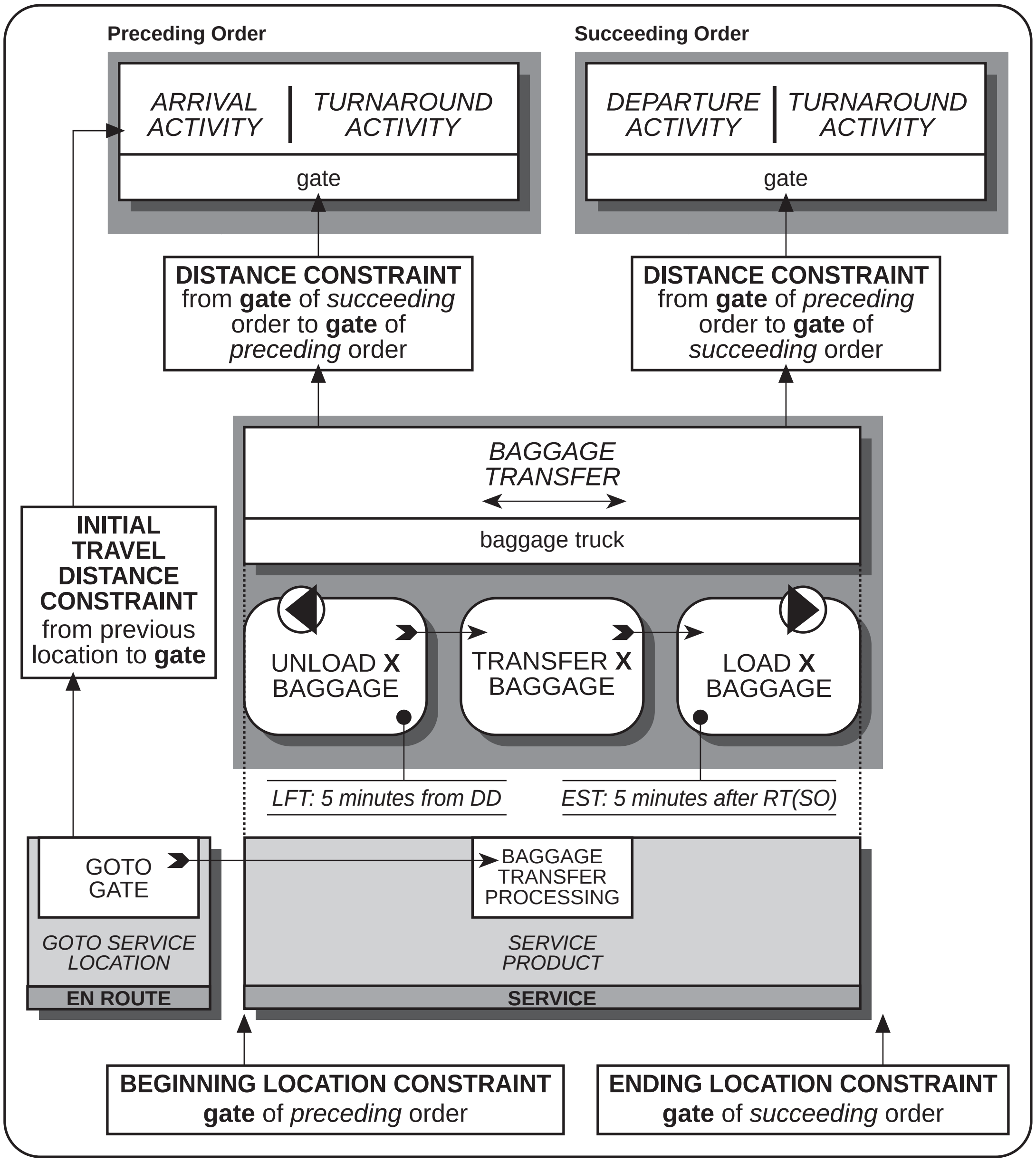

Preceding Order
ARRIVAL ACTIVITY
TURNAROUND ACTIVITY
gate
Succeeding Order
DEPARTURE ACTIVITY
TURNAROUND ACTIVITY
gate
DISTANCE CONSTRAINT from gate of succeeding order to gate of preceding order
DISTANCE CONSTRAINT from gate of preceding order to gate of succeeding order
BAGGAGE TRANSFER
baggage truck
INITIAL TRAVEL DISTANCE CONSTRAINT from previous location to gate
UNLOAD X BAGGAGE
TRANSFER X BAGGAGE
LOAD X BAGGAGE
LFT: 5 minutes from DD
EST: 5 minutes after RT(SO)
GOTO GATE
GOTO SERVICE LOCATION
EN ROUTE
BAGGAGE TRANSFER PROCESSING
SERVICE PRODUCT
SERVICE
BEGINNING LOCATION CONSTRAINT gate of preceding order
ENDING LOCATION CONSTRAINT gate of succeeding order

Figure A.8. Resource Plan for the BAGGAGE TRANSFER Product Task.

```
;;; ----------------------------------------------------------------------
;;; Baggage-Transfer Resource Task Definitions:

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity arrival-activity) gate))

(define-resource-task BAGGAGE-TRANSFER-PROCESSING (service-product)
   :NEXT-LOCATION
      (:RESOURCE
         (:INTER-ORDER (turnaround-activity departure-activity))
         gate))

;;; ----------------------------------------------------------------------
;;; Baggage-Transfer (and Children) Product Task Definitions:

(define-product-task UNLOAD-X-BAGGAGE
   :SHIFT-PREFERENCE :EARLY
   :LATEST-FINISH-TIME (:FROM-DUE-DATE 5)
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-unload-x-baggage)

(define-product-task TRANSFER-X-BAGGAGE
   :SPECIAL-REFINEMENT-CASE? t
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-transfer-x-baggage
   :MINIMUM-EXPECTED-DURATION-FUNCTION
      arm::minimum-time-required-to-transfer-x-baggage
   :MAXIMUM-EXPECTED-DURATION-FUNCTION
      arm::maximum-time-required-to-transfer-x-baggage)

(define-product-task LOAD-X-BAGGAGE
   :SHIFT-PREFERENCE :LATE
   :EARLIEST-START-TIME (:FROM-READY-TIME 5 :SUCCEEDING-ORDER)
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-load-x-baggage)

;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

(define-product-task BAGGAGE-TRANSFER
   :RESOURCE-REQUIREMENTS
      ((baggage-truck
          :RESOURCE-PLAN (goto-gate baggage-transfer-processing)
          :MAIN-ACTIVITY baggage-transfer-processing))
   :AGGREGATE-SUB-TASKS
      (:SEQUENCE unload-x-baggage transfer-x-baggage load-x-baggage))

;;; ----------------------------------------------------------------------
```

Figure A.9. Task Specifications for the BAGGAGE TRANSFER Operation.

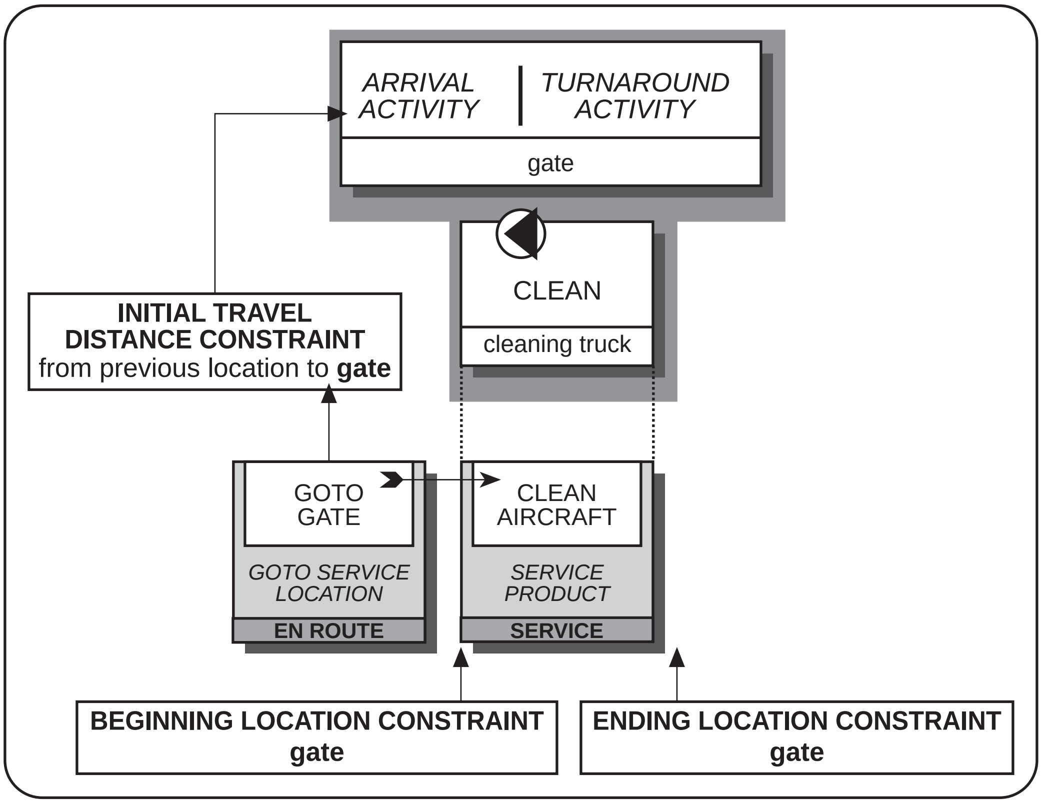


Figure A.10. Resource Plan for the CLEAN Product Task.

Figure A.11 shows the task specifications that define the CLEAN operation.

A.1.5.6 SERVICE

The SERVICE product task involves the general mechanical servicing of the aircraft. Figure A.12 provides an illustration of the SERVICE task, including its resource plan and associated constraints. This task is only performed by a SERVICE TRUCK.

The SERVICE product task is performed by means of a two-step resource plan. The `GOTO-GATE` resource task moves the service truck to the assigned gate, as specified by the BLC. If the truck is already at the gate, this resource task is not performed. Once the truck is at the gate, the servicing process may begin. The duration of the `SERVICE-AIRCRAFT` resource task ranges from 5 to 10 minutes, depending on the capacity of the aircraft. The ELC indicates that the service truck remains at the gate upon completion of the SERVICE task. The ITDC constrains the choice of a gate to service the flight, by limiting the set of usable gates to those within the allocated travel distance from the service truck's previous location.

Figure A.13 shows the task specifications that define the SERVICE operation.

A.1.5.7 RESTOCK

The RESTOCK product task involves the restocking of the necessary food and supplies for the aircraft. Figure A.14 provides an illustration of the RESTOCK task, including its resource plan and associated constraints. This task is only performed by a CATERING TRUCK.

```
;;; ----------------------------------------------------------------------
;;; Clean Resource Task Definitions:

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity arrival-activity) gate))

(define-resource-task CLEAN-AIRCRAFT (service-product))

;;; ----------------------------------------------------------------------
;;; Clean Product Task Definition:

(define-product-task CLEAN
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS
      ((cleaning-truck :RESOURCE-PLAN (goto-gate clean-aircraft)
                       :MAIN-ACTIVITY clean-aircraft)))

;;; ----------------------------------------------------------------------
```

Figure A.11. Task Specifications for the CLEAN Operation.

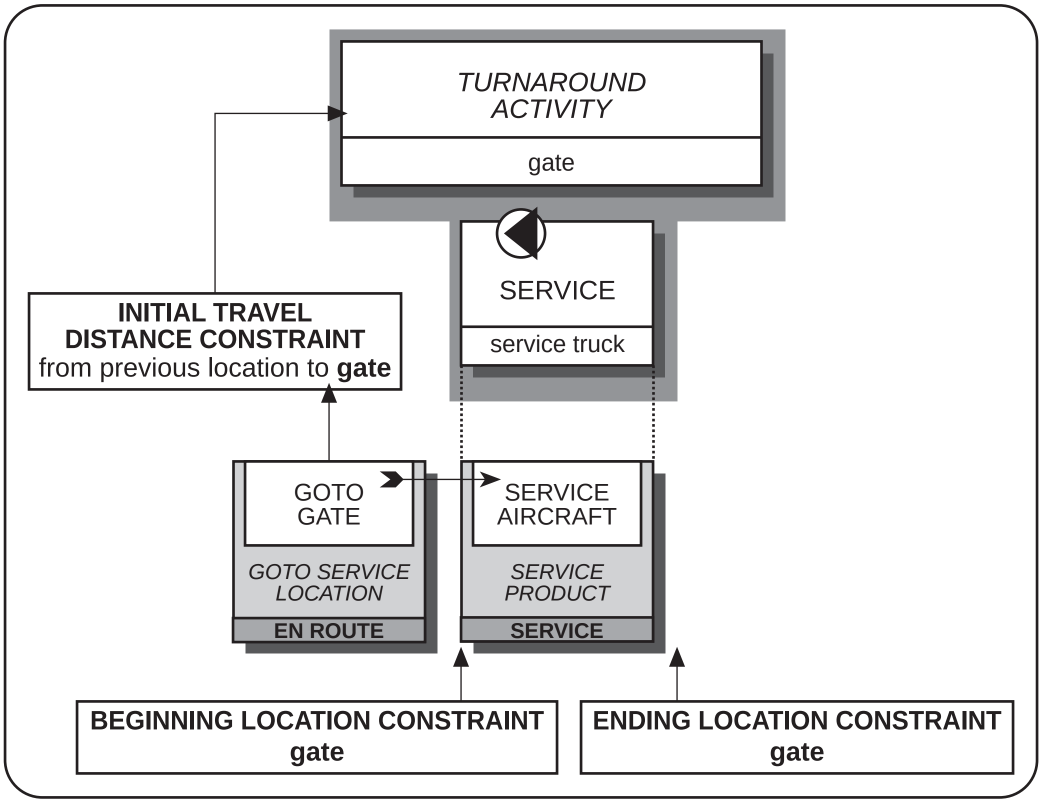

TURNAROUND ACTIVITY
gate
SERVICE
service truck
INITIAL TRAVEL DISTANCE CONSTRAINT from previous location to **gate**
GOTO GATE
GOTO SERVICE LOCATION
EN ROUTE
SERVICE AIRCRAFT
SERVICE PRODUCT
SERVICE
BEGINNING LOCATION CONSTRAINT gate
ENDING LOCATION CONSTRAINT gate

Figure A.12. Resource Plan for the SERVICE Product Task.

```
;;; -----------------------------------------------------------------------
;;; Service Resource Task Definitions:

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION (:RESOURCE turnaround-activity gate))

(define-resource-task SERVICE-AIRCRAFT (service-product))

;;; -----------------------------------------------------------------------
;;; Service Product Task Definition:

(define-product-task SERVICE
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS
      ((service-truck :RESOURCE-PLAN (goto-gate service-aircraft)
                      :MAIN-ACTIVITY service-aircraft)))

;;; -----------------------------------------------------------------------
```

Figure A.13. Task Specifications for the SERVICE Operation.

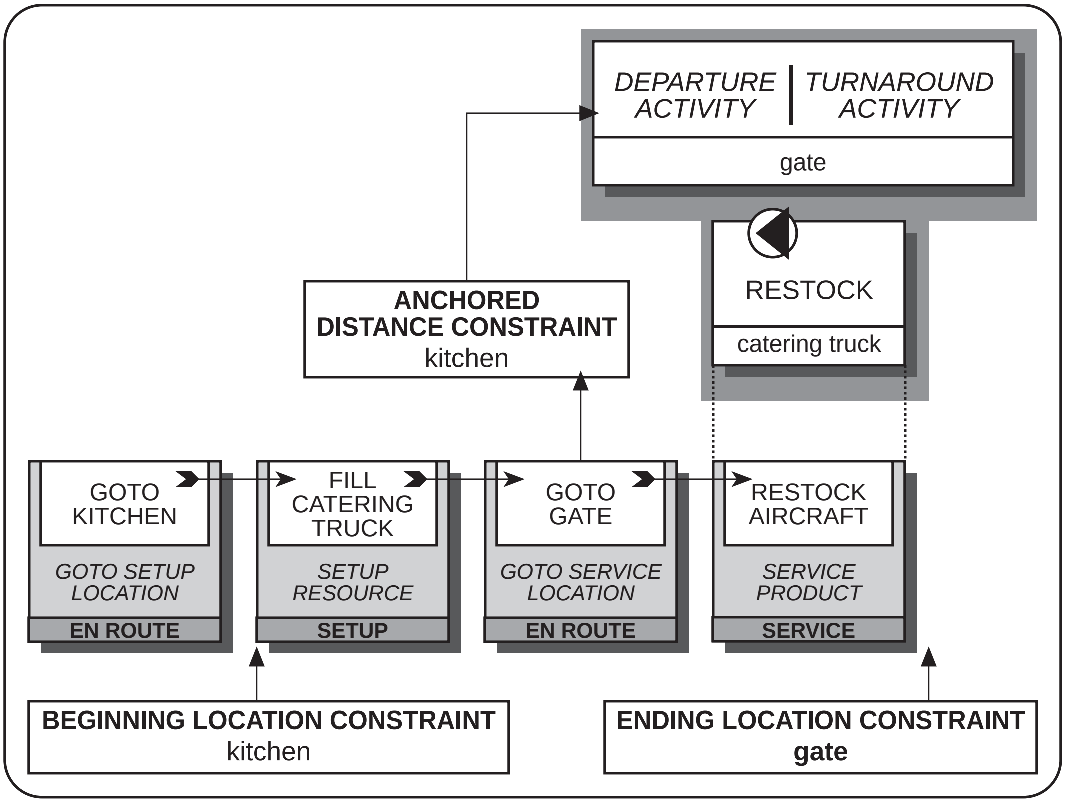


Figure A.14. Resource Plan for the RESTOCK Product Task.

The RESTOCK product task is performed by means of a four-step resource plan. The GOTO-KITCHEN resource task moves the catering truck to the kitchen shop location, as specified by the BLC. If the truck is already at the kitchen, this resource task is not performed. At this point, all necessary catering supplies are loaded onto the truck. The duration of the FILL-CATERING-TRUCK resource task ranges from 6 to 14 minutes when full meal service is provided with the departing flight, from 5 to 14 minutes when only snack service is provided, and from 4 to 10 minutes when no food service is provided. These times are all dependent on the capacity of the aircraft. The GOTO-GATE resource task moves the catering truck to the assigned gate, at which point the supplies are loaded onto the aircraft. The durations for the RESTOCK-AIRCRAFT resource task are identical to the times for the FILL-CATERING-TRUCK resource task. The ELC indicates that the catering truck remains at the gate upon completion of the RESTOCK task. The ADC constrains the choice of a gate to service the flight, by limiting the set of usable gates to those within the allocated travel distance from the kitchen shop location.

Figure A.15 shows the task specifications that define the RESTOCK operation.

```
;;; ------------------------------------------------------------------
;;; Restock Resource Task Definitions:

(define-resource-task GOTO-KITCHEN (goto-setup-location)
   :NEXT-LOCATION kitchen)

(define-resource-task FILL-CATERING-TRUCK (setup-resource))

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity departure-activity) gate))

(define-resource-task RESTOCK-AIRCRAFT (service-product))

;;; ------------------------------------------------------------------
;;; Restock Product Task Definition:

(define-product-task RESTOCK
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS
      ((catering-truck
          :RESOURCE-PLAN (goto-kitchen fill-catering-truck
                          goto-gate restock-aircraft)
          :MAIN-ACTIVITY restock-aircraft)))

;;; ------------------------------------------------------------------
```

Figure A.15. Task Specifications for the RESTOCK Operation.

A.1.5.8 REFUEL

The REFUEL product task involves the refueling of the aircraft. Figure A.16 provides an illustration of the REFUEL task, including its two possible resource plans and their associated constraints. This task is performed by *either* a FUEL TRUCK or a PUMP TRUCK.

While many real-world aircraft actually do allow for servicing by multiple refueling trucks due to their being equipped with multiple fuel connection points (one under each wing, for

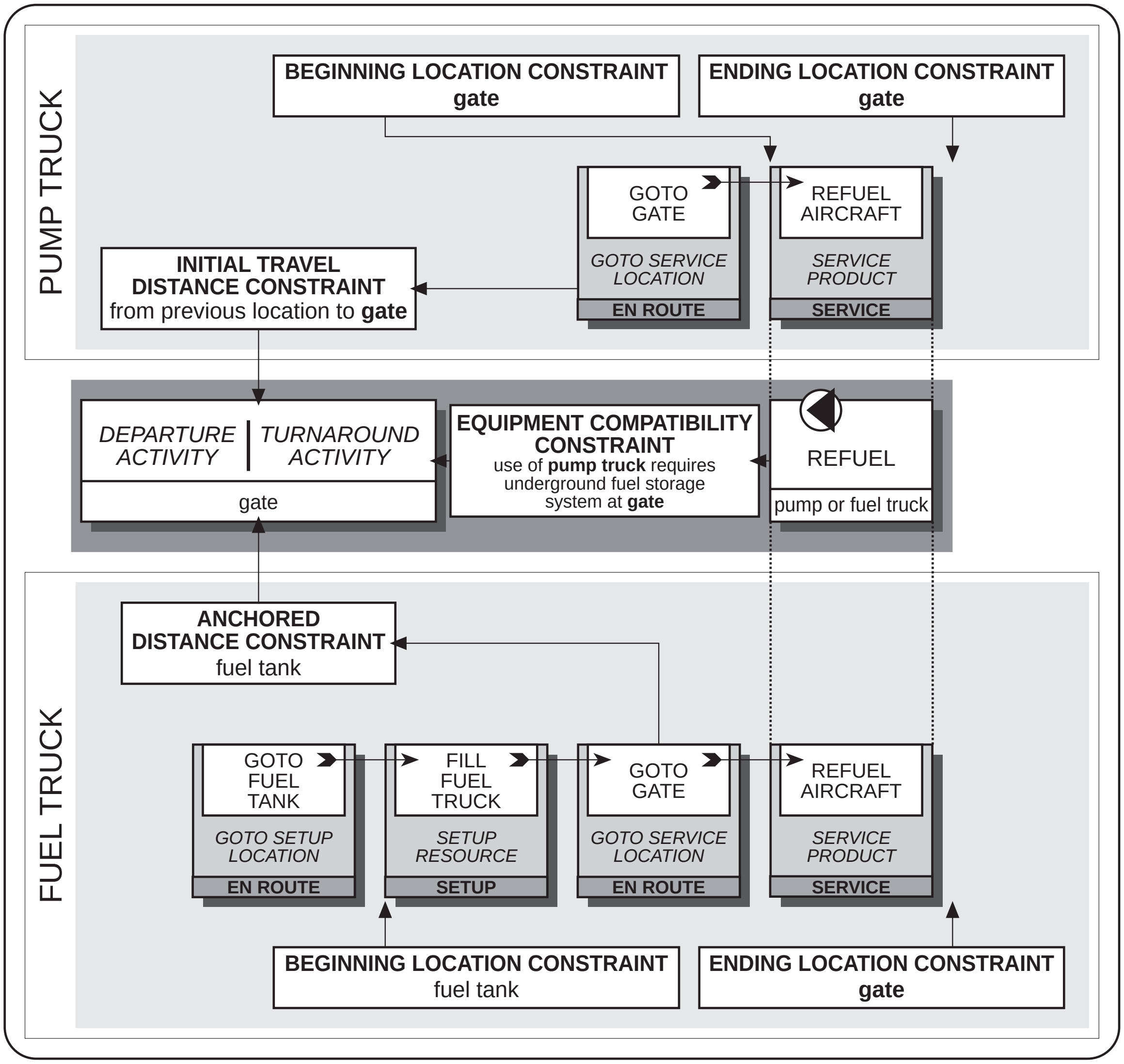


Figure A.16. Both Resource Plans for the REFUEL Product Task.

example), our implementation of the AGSS domain requires the refueling task to be performed by a single truck only, be it a fuel truck or a pump truck.

When the gate being used for staging the overall flight servicing activity is equipped with a hydrant system for dispensing fuel from an underground storage facility, a pump truck may be used to perform the refueling task. In this case, the REFUEL task is performed by means of a two-step resource plan. The GOTO-GATE resource task moves the pump truck to the assigned gate, as specified by the BLC. If the truck is already at the gate, this resource task is not performed. Once the truck is at the gate, it hooks up to both the hydrant system and the aircraft, and the refueling process may begin. The duration of the REFUEL-AIRCRAFT resource task is dependent on the distance from the local airport to the immediate destination city of the departing leg of the flight. The ELC indicates that the pump truck remains at the gate upon completion of the REFUEL task. The ITDC constrains the choice of a gate to service the flight, by limiting the set of usable gates to those within the allocated travel distance from the pump truck's previous location. The ECC indicates that the use of a pump truck for performing the REFUEL task constrains the choice of a gate by forcing it to be equipped with an underground fuel storage system.

When a hydrant system is *not* available at a gate, a fuel truck must be used. In this case, the REFUEL task is performed by means of a four-step resource plan. The GOTO-FUEL-TANK resource task moves the fuel truck to the fuel tank shop location, as specified by the BLC. If the truck is already at the fuel tank, this resource task is not performed. At this point, the required amount of fuel is loaded onto the truck. The duration of the FILL-FUEL-TRUCK depends on the distance from the local airport to the immediate destination city of the departing leg of the flight. The GOTO-GATE resource task moves the fuel truck to the assigned gate, at which point the fuel is loaded onto the aircraft. The durations for the REFUEL-AIRCRAFT resource task are identical to the times for the FILL-FUEL-TRUCK resource task. The ELC indicates that the fuel truck remains at the gate upon completion of the REFUEL task. The ADC constrains the choice of a gate to service the flight, by limiting the set of usable gates to those within the allocated travel distance from the fuel tank.

Figure A.17 shows the task specifications that define the REFUEL operation.

A.1.5.9 PASSENGER TRANSFER

The PASSENGER TRANSFER product task is an *inter-order* task that involves the transfer of passengers between the aircraft for two connected flights. No resource is required for this operation. Figure A.18 illustrates the constraints that are imposed by the PASSENGER TRANSFER task.

The DISTANCE-CONSTRAINTs imposed on each of the gates constrain the choice of gates to service the flights to those within the allocated travel distance of each other, as indicated by the duration of the PASSENGER TRANSFER task.

Figure A.19 shows the task specification that defines the PASSENGER TRANSFER operation. The DISTANCE-CONSTRAINTSs imposed on each of the gates are produced by the arm::passenger-transfer-constraints function, which is linked to the PASSENGER TRANSFER product task by the CONSTRAINT-FUNCTION argument.

A.1.5.10 POWER IN, SHUTDOWN, DEPLANE, ENPLANE, POWER OUT, *and* STARTUP

These remaining product tasks do not require any resources.

```
;;; ---------------------------------------------------------------------
;;; Refuel Resource Task Definitions:

;;; Fuel-Truck:

(define-resource-task GOTO-FUEL-TANK (goto-setup-location)
   :NEXT-LOCATION fuel-tank)

(define-resource-task FILL-FUEL-TRUCK (setup-resource))

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity departure-activity) gate))

(define-resource-task REFUEL-AIRCRAFT (service-product))

;;; - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

;;; Pump-Truck:

(define-resource-task GOTO-GATE (goto-service-location)
   :NEXT-LOCATION
      (:RESOURCE (turnaround-activity departure-activity) gate))

(define-resource-task REFUEL-AIRCRAFT (service-product))

;;; ---------------------------------------------------------------------
;;; Refuel Product Task Definition:

(define-product-task REFUEL
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS
      ((fuel-truck
          :RESOURCE-PLAN
             (goto-fuel-tank fill-fuel-truck goto-gate refuel-aircraft)
          :MAIN-ACTIVITY refuel-aircraft)
       (pump-truck
          :RESOURCE-PLAN (goto-gate refuel-aircraft)
          :MAIN-ACTIVITY refuel-aircraft)))

;;; ---------------------------------------------------------------------
```

Figure A.17. Task Specifications for the REFUEL Operation.

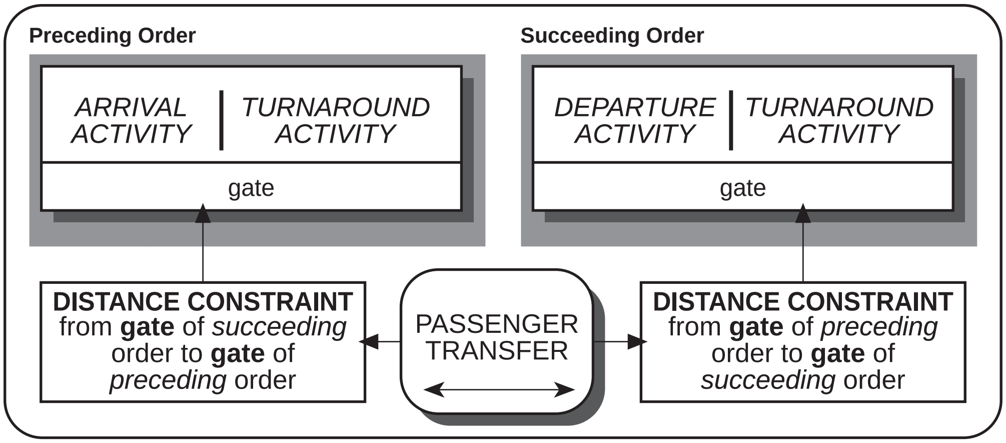

Preceding Order
Succeeding Order
ARRIVAL ACTIVITY
TURNAROUND ACTIVITY
gate
DEPARTURE ACTIVITY
TURNAROUND ACTIVITY
gate
DISTANCE CONSTRAINT from gate of succeeding order to gate of preceding order
PASSENGER TRANSFER
DISTANCE CONSTRAINT from gate of preceding order to gate of succeeding order

Figure A.18. Constraints Imposed by the PASSENGER TRANSFER Product Task.

```
;;; -----------------------------------------------------------------
;;; Passenger-Transfer Product Task Definition:

(define-product-task PASSENGER-TRANSFER
   :SPECIAL-REFINEMENT-CASE? t
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-transfer-passengers
   :MINIMUM-EXPECTED-DURATION-FUNCTION
      arm::minimum-time-required-to-transfer-passengers
   :MAXIMUM-EXPECTED-DURATION-FUNCTION
      arm::maximum-time-required-to-transfer-passengers
   :CONSTRAINT-FUNCTION arm::passenger-transfer-constraints)

;;; -----------------------------------------------------------------
```

Figure A.19. Task Specification for the PASSENGER TRANSFER Operation.

- POWER IN

  The POWER IN product task involves the process of maneuvering the aircraft into the proper docking location at a gate and connecting it to the jetway. The amount of processing time required for this operation ranges from 1 to 2 minutes, depending on the size of the aircraft. The POWER IN task has an early shift preference. No resource is required for this operation.[5]

- SHUTDOWN

  The SHUTDOWN product task involves the process of mechanically shutting down the aircraft and connecting it to the jetway upon its arrival at a gate. For *Arrival* and *Turnaround* flights, this activity encompasses the POWER IN operation. The amount of processing time required for this operation ranges from 3 to 4 minutes, depending on the size of the aircraft. The SHUTDOWN task has an early shift preference. No resource is required for this operation.

- DEPLANE

  The DEPLANE product task involves the process of passenger disembarkation. The amount of processing time required for this operation ranges from 4 to 7 minutes, depending on the seating capacity of the aircraft. The DEPLANE task has an early shift preference. No resource is required for this operation.

- ENPLANE

  The ENPLANE product task involves the process of passenger embarkation (boarding). The amount of processing time required for this operation ranges from 6 to 9 minutes, depending on the seating capacity of the aircraft. The ENPLANE task has a late shift preference. No resource is required for this operation. The EST constraint indicates that the ENPLANE task may not begin any earlier than a time 15 minutes prior to the due date of the order (the expected departure time for the flight).

[5] Note that an appropriate (and realistic) extension to both the POWER IN and POWER OUT task definitions would be to add the requirement of a mobile TUG resource to help with the maneuvering of certain larger types of aircraft.

- POWER OUT

  The POWER OUT product task involves the process of disconnecting the aircraft from the jetway and maneuvering it out from its docking location at a gate. The amount of processing time required for this operation ranges from 1 to 2 minutes, depending on the size of the aircraft. The POWER OUT task has a late shift preference. No resource is required for this operation.

- STARTUP

  The STARTUP product task involves the process of disconnecting the aircraft from the jetway and mechanically starting it up prior to its departure from a gate. For *Turnaround* and *Departure* flights, this activity encompasses the POWER OUT operation. The amount of processing time required for this operation ranges from 3 to 4 minutes, depending on the size of the aircraft. The STARTUP task has a late shift preference. No resource is required for this operation.

Figure A.20 shows the task specifications that define the POWER IN, SHUTDOWN, DEPLANE, ENPLANE, POWER OUT, and STARTUP operations.

```
;;; -------------------------------------------------------------------
;;; Non Resource-Requiring (Intra-Order) Product Task Definitions:

(define-product-task POWER-IN
   :SHIFT-PREFERENCE :EARLY
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-power-in)

(define-product-task SHUTDOWN
   :SHIFT-PREFERENCE :EARLY
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-shutdown)

(define-product-task DEPLANE
   :SHIFT-PREFERENCE :EARLY
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-deplane)

(define-product-task ENPLANE
   :SHIFT-PREFERENCE :LATE
   :EARLIEST-START-TIME (:FROM-DUE-DATE 15)
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-enplane)

(define-product-task STARTUP
   :SHIFT-PREFERENCE :LATE
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-startup)

(define-product-task POWER-OUT
   :SHIFT-PREFERENCE :LATE
   :EXPECTED-DURATION-FUNCTION arm::time-required-to-power-out)

;;; -------------------------------------------------------------------
```

Figure A.20. Task Specifications for the POWER IN, SHUTDOWN, DEPLANE, ENPLANE, POWER OUT, and STARTUP Operations.

A.1.6 Orders and Process Plans

In this section, we describe the different types of orders described in ARM, and present their corresponding process plans. The orders in our implementation of the AGSS domain represent regularly scheduled passenger flights. There are no cargo, charter or private flights. ARM understands three types of flights, referred to as *Turnaround*, *Arrival* and *Departure* flights, each distinguished by the specific kind of ground service required by the aircraft involved. The aircraft for a *Turnaround* flight originates elsewhere, arrives at a gate at the local airport, and then departs for another destination following a short (roughly one hour) layover period. The aircraft for an *Arrival* flight originates elsewhere, arrives at a gate at the local airport, and is then moved to the local hangar following a short servicing period. The aircraft for a *Departure* flight is moved from the local hangar to a gate, and then departs for another destination following a short preparation period.

Each of these types of flights has a desired ready time and due date respectively corresponding to their expected arrival and departure times. In the AGSS domain, one of the primary scheduling goals is to produce a schedule that exactly meets the ready times and due dates of all flights so that each is serviced immediately upon arrival and does not leave any earlier or later than its expected departure time. This goal as achieved by specifying late shift preferences on some of the tasks in the process plan template so that the task network may not be collapsed during the scheduling process.

Each flight that requires scheduling in ARM is instantiated from a particular *flight specifier*. The flight specifier provides all of the information that is needed by the domain code that helps DSS produce a schedule. This code calculates the expected and required durations of the various ground servicing tasks, and identifies pairs of connecting flights. We provide below a description of the various kinds of information that help describe each individual flight.

- SERVICE TYPE: Indicates the specific kind of ground service required by the flight.

- REMOTE DEPARTURE TIME: This time serves as the point when DSS is alerted that a flight will require scheduling. For *Arrival* and *Turnaround* flights, this is the time when the flight is expected to begin its flight to the local airport.[6] For *Departure* flights, this time is set according to the default amount of time the user has indicated the associated ground servicing activities will require, which is subtracted from the expected departure time, and an additional user-supplied duration indicating how much advance warning is to be given, which is subtracted from the previous result.

- ARRIVAL TIME: Indicates the expected arrival time for a flight at a gate. For *Departure* flights, this time is set according to the default amount of time the user has indicated the associated ground servicing activities will require, which is subtracted from the expected departure time.

- DEPARTURE TIME: Indicates the expected departure time for a flight from a gate. For *Arrival* flights, this time is set according to the default amount of time the user has indicated the associated ground servicing activities will require, which is added to the expected arrival time.

[6] Note that this value is subject to override by the user.

- SERVICE [ORIGIN]: The level of meal service provided on the arriving leg of a flight. This value is undefined for *Departure* flights.
- SERVICE [DESTINATION]: The level of meal service provided on the departing leg of a flight. This value is undefined for *Arrival* flights.
- CONNECTING FLIGHTS [DESTINATION]: Indicates any remote flights to which the departing leg of flight is connected. This value is undefined for *Arrival* flights.
- AIRLINE: The airline responsible for a flight.
- PRECEDING FLIGHT SPECIFIERS: A link used to identify all preceding inter-connected flights.
- SUCCEEDING FLIGHT SPECIFIERS: A link used to identify all succeeding inter-connected flights.
- ORIGIN: The origin city of an *Arrival* or *Turnaround* flight. This value is undefined for *Departure* flights.
- DESTINATION: The destination city of a *Departure* or *Turnaround* flight. This value is undefined for *Arrival* flights.

We now describe the process plans for the three types of flights defined in ARM.

A.1.6.1 Arrival ***Flights***

An *Arrival* flight aircraft arrives at the airport, reports to its assigned gate for servicing, and is then moved to the local hangar to await later usage by a *Departure* flight.[7] The duration of this activity defaults to 40 minutes, and is specified by the user. The required activities involve emptying and cleaning the aircraft. Baggage is unloaded and transferred to any connecting flights. Passengers are deplaned and transferred to any connecting flights. The aircraft must be properly shut down upon arrival and then powered out of the gate prior to its being moved to the local hangar. The process plan template for this particular service is illustrated in Figure A.21.

The READY TIME label in the three following process plan figures refers to the flight arrival time for *Arrival* and *Turnaround* flights, or the time at which a departing aircraft arrives at its gate (from the hangar) prior to *Departure* flight processing. The DUE DATE label refers to the flight departure time for *Departure* and *Turnaround* flights, or or the time at which an arriving aircraft leaves its gate (for the hangar) following *Arrival* flight processing.

Figure A.22 shows the specification of the process plan template and the top-level product task (ARRIVAL ACTIVITY) that describe the required ground servicing activities comprising the ARRIVAL FLIGHT PROCESSING service type. The `SUCCEEDING-INTER-ORDER-TASK` entry indicates the origin of a particular inter-order task. The second argument in the specification

[7] Remember, however, that in our implementation of the AGSS domain, an *Arrival* flight aircraft will report to the hangar, but will not be reused by any other flight. Instead, a new aircraft is created for each *Departure* flight and given the hangar as its initial location.

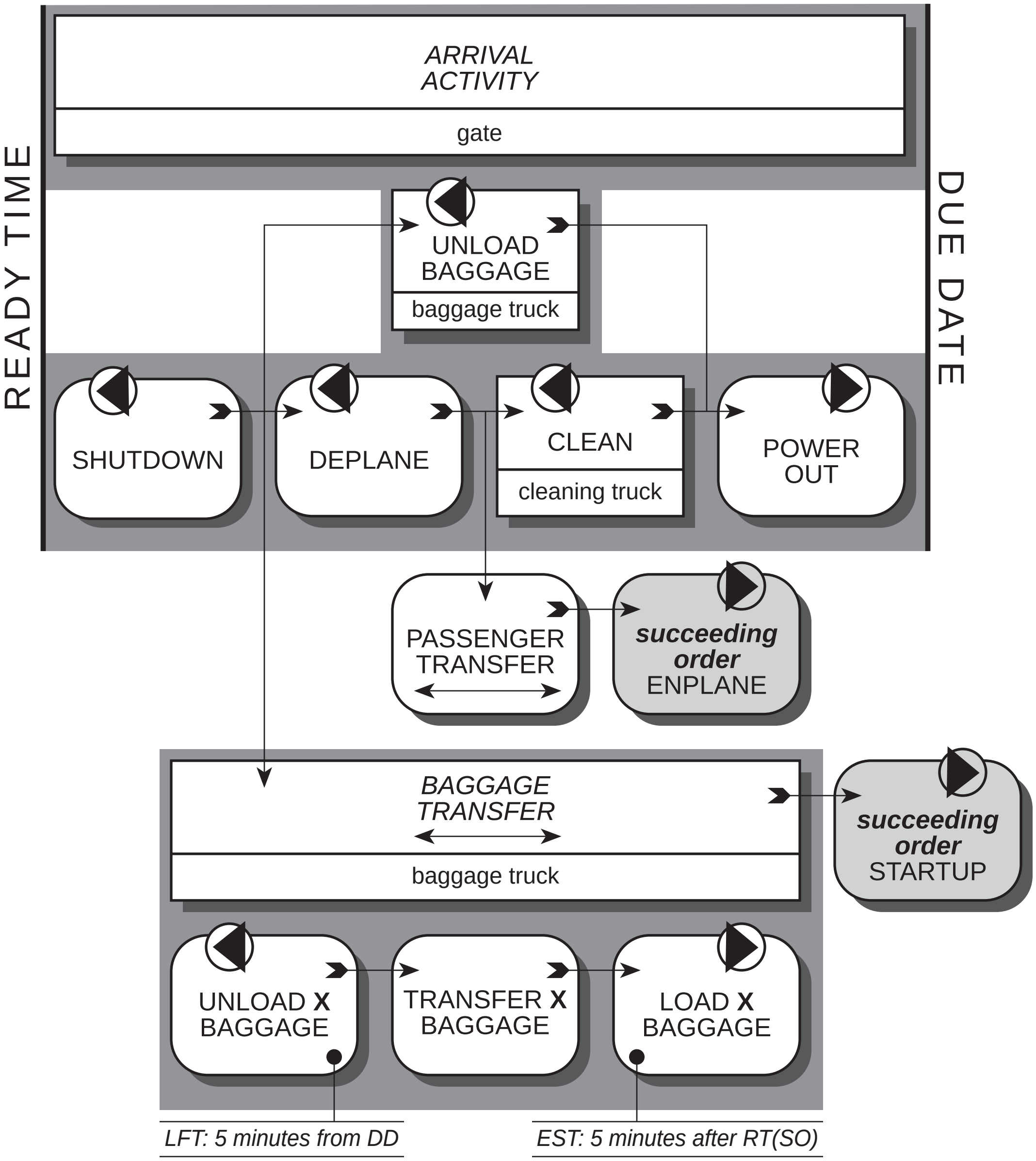


Figure A.21. Process Plan for the ARRIVAL FLIGHT PROCESSING Service Type.

```
;;; -----------------------------------------------------------------
;;; Arrival-Activity Product Task Definition:

(define-product-task ARRIVAL-ACTIVITY
   :RESOURCE-REQUIREMENTS
      ((gate :RESOURCE-PLAN (prepare-gate gate-processing)
             :MAIN-ACTIVITY gate-processing))
   :AGGREGATE-SUB-TASKS
      (:SEQUENCE
         shutdown
         (:PARALLEL
            (:SEQUENCE
               deplane
               (:PARALLEL
                  (:SUCCEEDING-INTER-ORDER-TASK
                     arm::flight-spec$succeeding-flight-specs
                     passenger-transfer)
                  clean))
            (:SUCCEEDING-INTER-ORDER-TASK
               arm::flight-spec$succeeding-flight-specs
               baggage-transfer)
            unload-baggage)
         power-out))

;;; -----------------------------------------------------------------
;;; Arrival Flight Process Plan Definition:

(define-process-plan ARRIVAL-FLIGHT-PROCESSING
   :SERVICES arrival-activity)

;;; -----------------------------------------------------------------
```

Figure A.22. Specifications for the ARRIVAL FLIGHT PROCESSING Service Type.

is the accessor for the SUCCEEDING FLIGHT SPECIFIERS slot of the order's corresponding flight specifier. It returns the set of all connected flight specifiers for which an instance of this inter-order task must be instantiated. The third argument is the name of the inter-order task.

A.1.6.2 Departure *Flights*

A *Departure* flight aircraft waits at the local hangar before reporting to its assigned gate for servicing, after which it departs from the airport. The duration of the service activity defaults to 40 minutes, and is specified by the user. The required activities involve the loading of the aircraft. Baggage, including quantities possibly transferred from connecting flights, is loaded. The aircraft is restocked and refueled. Passengers, possibly including some from connecting flights, are enplaned. The aircraft is powered into the gate upon arrival from the hangar, and must be started up prior its departure. The process plan template for this particular service is illustrated in Figure A.23.

Figure A.24 shows the specification of the process plan template and the top-level product task (DEPARTURE ACTIVITY) that describe the required ground servicing activities comprising the DEPARTURE FLIGHT PROCESSING service type. The `PRECEDING-INTER-ORDER-TASK` entry indicates the destination of a particular inter-order task. The second argument in the specification is the accessor for the PRECEDING FLIGHT SPECIFIERS slot of the order's corresponding flight specifier. It returns the set of all connected flight specifiers for which an

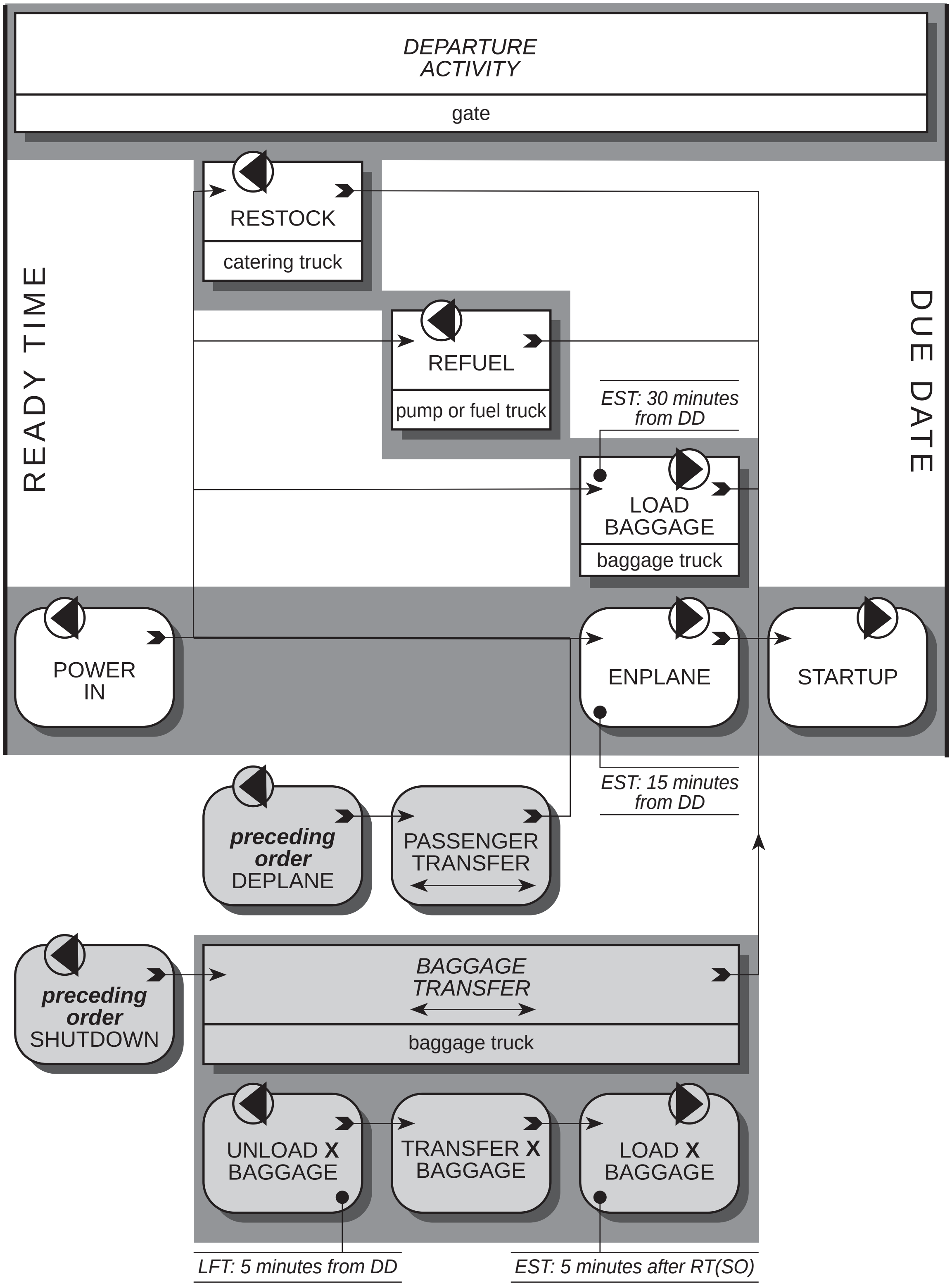


Figure A.23. Process Plan for the DEPARTURE FLIGHT PROCESSING Service Type.

```
;;; ----------------------------------------------------------------------
;;; Departure-Activity Product Task Definition:

(define-product-task DEPARTURE-ACTIVITY
   :RESOURCE-REQUIREMENTS
      ((gate :RESOURCE-PLAN (prepare-gate gate-processing)
             :MAIN-ACTIVITY gate-processing))
   :AGGREGATE-SUB-TASKS
      (:SEQUENCE
         power-in
         (:PARALLEL
            (:SEQUENCE
               (:PRECEDING-INTER-ORDER-TASK
                  arm::flight-spec$preceding-flight-specs
                  passenger-transfer)
               enplane)
            (:PRECEDING-INTER-ORDER-TASK
               arm::flight-spec$preceding-flight-specs
               baggage-transfer)
            load-baggage
            refuel
            restock)
         startup))

;;; ----------------------------------------------------------------------
;;; Departure Flight Process Plan Definition:

(define-process-plan DEPARTURE-FLIGHT-PROCESSING
   :SERVICES departure-activity)

;;; ----------------------------------------------------------------------
```

Figure A.24. Specifications for the DEPARTURE FLIGHT PROCESSING Service Type.

instance of this inter-order task must be instantiated. The third argument is the name of the inter-order task.

A.1.6.3 Turnaround *Flights*

Turnaround flights represent the majority of flights. They are comprised of two separate segments: an arriving leg and a departing leg. The basic servicing of a *Turnaround* flight aircraft generally takes place within a one hour period. Baggage must be unloaded, loaded and possibly transferred to one or more connecting flights.[8] The aircraft must be cleaned, serviced, refueled and restocked for the departing flight. Passengers must be deplaned, possibly transferred to any connecting flights, and enplaned. The aircraft itself requires specific periods for shutting down upon arriving at the gate, and starting up prior to its departure. The process plan template for this particular service is presented in Figure A.25. The gradient-filled inter-order tasks in Figure A.25 represent the preceding *and* succeeding instantiations of the task nodes for the TURNAROUND FLIGHT PROCESSING order class.

Figure A.26 shows the specification of the process plan template and the top-level product task (TURNAROUND ACTIVITY) that describe the required ground servicing activities comprising the TURNAROUND FLIGHT PROCESSING service type.

[8] The unloading, transfer, and loading of the baggage *from* a connecting flight is the responsibility of the originating flight.

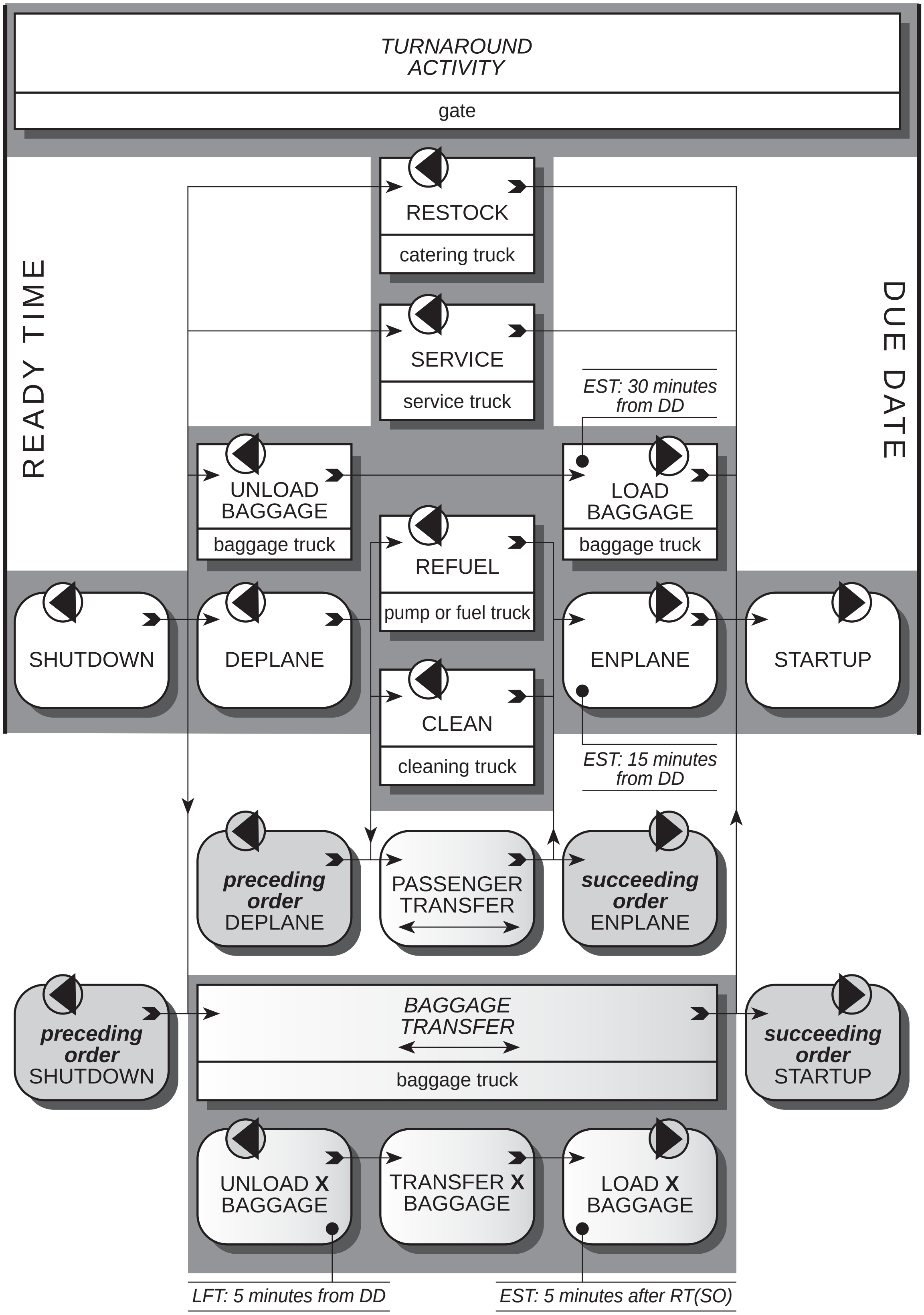


Figure A.25. Process Plan for the TURNAROUND FLIGHT PROCESSING Service Type.

```
;;; -----------------------------------------------------------------------
;;; Turnaround-Activity Product Task Definition:

(define-product-task TURNAROUND-ACTIVITY
   :RESOURCE-REQUIREMENTS
      ((gate :RESOURCE-PLAN (prepare-gate gate-processing)
            :MAIN-ACTIVITY gate-processing))
   :AGGREGATE-SUB-TASKS
      (:SEQUENCE
         shutdown
         (:PARALLEL
            (:SEQUENCE
               deplane
               (:PARALLEL
                  refuel
                  (:SUCCEEDING-INTER-ORDER-TASK
                     arm::flight-spec$succeeding-flight-specs
                     passenger-transfer)
                  (:PRECEDING-INTER-ORDER-TASK
                     arm::flight-spec$preceding-flight-specs
                     passenger-transfer)
                  clean)
               enplane)
            (:SUCCEEDING-INTER-ORDER-TASK
               arm::flight-spec$succeeding-flight-specs
               baggage-transfer)
            (:PRECEDING-INTER-ORDER-TASK
               arm::flight-spec$preceding-flight-specs
               baggage-transfer)
            (:SEQUENCE unload-baggage load-baggage)
            restock
            service)
         startup))

;;; -----------------------------------------------------------------------
;;; Turnaround Flight Process Plan Definition:

(define-process-plan TURNAROUND-FLIGHT-PROCESSING
   :SERVICES turnaround-activity)

;;; -----------------------------------------------------------------------
```

Figure A.26. Specifications for the TURNAROUND FLIGHT PROCESSING Service Type.

A.2 TCP*: The Turbine Component Plant Job-Shop Scheduling System*

The TCP application system implements the job-shop scheduling domain used in the evaluation of the initial OPIS system [Ow, 1986, Ow and Smith, 1988]. The factory model for this benchmark system is described in [Chiang *et al.*, 1990]. Compared to the AGSS domain as implemented in ARM, this domain is quite straightforward.

A.2.1 The Turbine Component Plant Job-Shop Scheduling Domain

TCP implements a standard job-shop for producing turbine blades. Figure A.27 presents the job-shop model for the domain. Three classes (families) of turbine blades are produced by the factory, with a different process routing for each family.

A.2.2 Domain Assumptions

The factory controlled by TCP only operates from 8:00am Monday morning until 8:00am Saturday morning, at which point it is shut down for the weekend (a two-day period). There is no penalty for suspending the processing of a job over the weekend. Suspended work is immediately resumed on the following Monday morning.

A.2.3 Resources

The TCP factory layout consists of a set of 33 stationary machine resources assigned to eleven work areas. Each machine requires a setup operation prior to its main servicing activity, unless the immediately preceding operation of the resource involved a product of the same family. The setup operations for each resource class have constant durations.

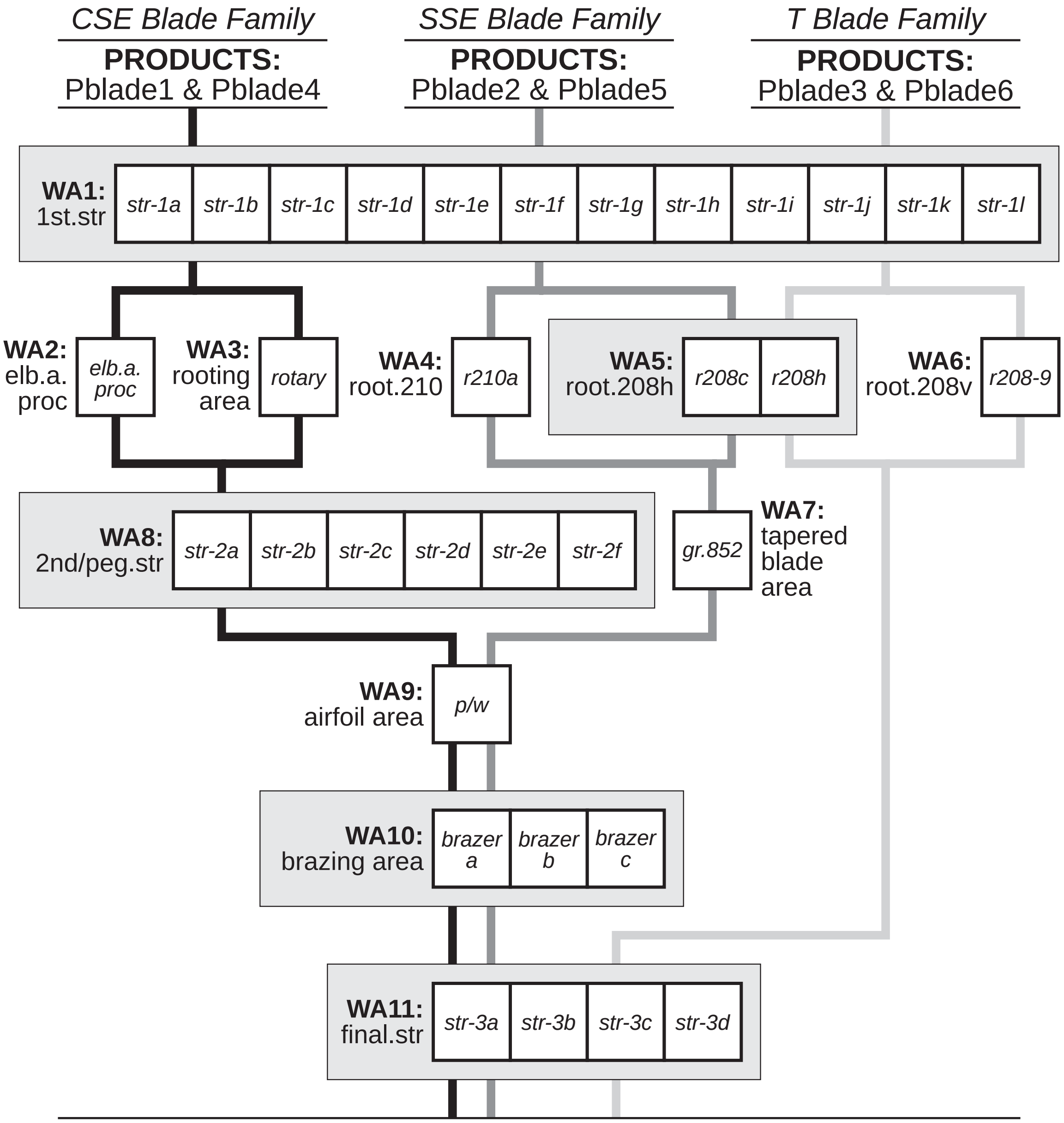
A.2.4 Products

There are three different turbine blade families, with two products per family. The *CSE* blade family includes products PBLADE1 and PBLADE4. The *SSE* blade family includes products PBLADE2 and PBLADE5. The *T* blade family includes products PBLADE3 and PBLDAE6.

A.2.5 Operations

Figure A.28 provides an illustration of the resource plan for the typical TCP product task. Each product task has an early shift preference, and consists of a `SERVICE-PRODUCT` resource task preceded by an optional `SETUP-RESOURCE` resource task. The processing durations for the product tasks are based on the operation being performed, and the lot-size of the order.

Figure A.29 shows the task specifications that define the typical TCP product task. The non-nil value for the `VARIABLE-SETUP-DURATION` flag indicates that the `SETUP-RESOURCE` resource task is optional. Note that the specification does not include the definition for the three product tasks that allow for a choice of machine types.



This figure is based on Figure 2 (page 3) from CMU-RI-TR-90-05,
***Factory Model and Test Data Descriptions: OPIS Experiments**,*
by Whay-Yu Chiang, Mark S. Fox, and Peng Si Ow.

Figure A.27. TCP Job-Shop Model

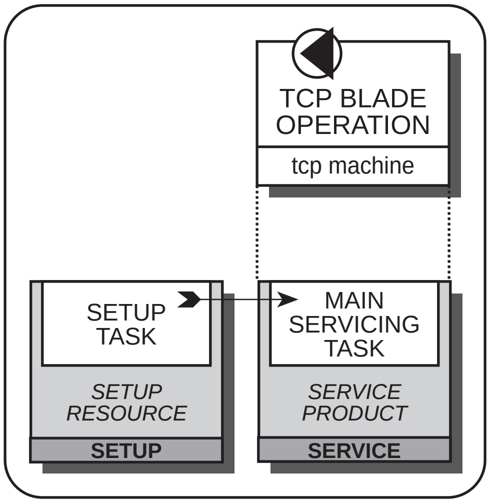


Figure A.28. Basic Resource Plan for all TCP Product Tasks.

```
;;; -----------------------------------------------------------------------
;;; General TCP Operation Resource Task Definitions:

(define-resource-task SETUP-TCP-RESOURCE (setup-resource)
   :VARIABLE-SETUP-DURATION? t)

(define-resource-task MAIN-TCP-OPERATION-ACTIVITY (service-product))

;;; -----------------------------------------------------------------------
;;; General TCP Operation Product Task Definition:

(define-product-task TCP-OPERATION
   :SHIFT-PREFERENCE :EARLY
   :RESOURCE-REQUIREMENTS*
   ((tcp-resource
       :RESOURCE-PLAN (setup-tcp-resource main-tcp-operation-activity)
       :MAIN-ACTIVITY main-tcp-operation-activity)))

;;; -----------------------------------------------------------------------
```

* The second operation in each of the three TCP service types provides a choice between two resource types.

Figure A.29. Basic Task Specifications for all TCP Operations.

A.2.6 Orders

Orders in TCP have a lot-size indicating the quantity of a certain (single) type of turbine blade to produce. In addition, each order is assigned to one of six priority classes, indicating its relative urgency. The priority class for an order is combined with its scheduled tardiness (in terms of days tardy) to determine its tardiness cost.

A.2.7 Process Plans

We now describe the process plans used for producing the three families of turbine blades.

A.2.7.1 CSE *Blade Family*

The process plan templates for the PRODUCE PBLADE1 and PRODUCE PBLADE4 service types are presented in Figure A.30. Both products (PBLADE1 and PBLADE4) are produced in a six-step serial process, with a choice of machine types to perform the second operation.

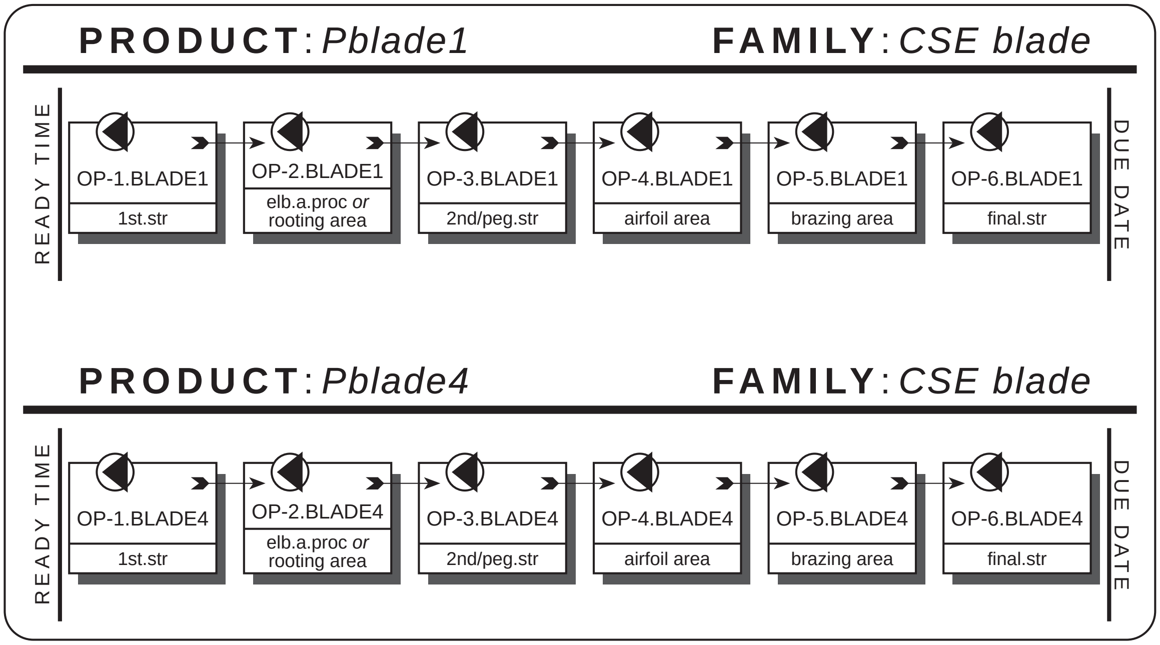


Figure A.30. Process Plans for the PRODUCE PBLADE1 & PRODUCE PBLADE4 Service Types.

Figure A.31 shows the specification of the process plan template that describes the PRODUCE PBLADE1 and PRODUCE PBLADE4 service types.

A.2.7.2 SSE *Blade Family*

The process plan templates for the PRODUCE PBLADE2 and PRODUCE PBLADE5 service types are presented in Figure A.32. Both products (PBLADE2 and PBLADE5) are produced in a six-step serial process, with a choice of machine types to perform the second operation.

Figure A.33 shows the specification of the process plan template that describes the PRODUCE PBLADE2 and PRODUCE PBLADE5 service types.

```
;;; ----------------------------------------------------------------------
;;; CSE Blade Family Process Plan Definitions:

;;; PBlade1:

(define-process-plan PRODUCE_PBLADE1
   :SERVICES (:SEQUENCE op-1.blade1 op-2.blade1 op-3.blade1
                        op-4.blade1 op-5.blade1 op-6.blade1))

;;; PBlade4:

(define-process-plan PRODUCE_PBLADE4
   :SERVICES (:SEQUENCE op-1.blade4 op-2.blade4 op-3.blade4
                        op-4.blade4 op-5.blade4 op-6.blade4))

;;; ----------------------------------------------------------------------
```

Figure A.31. Specifications for the PRODUCE PBLADE1 & PRODUCE PBLADE4 Service Types.

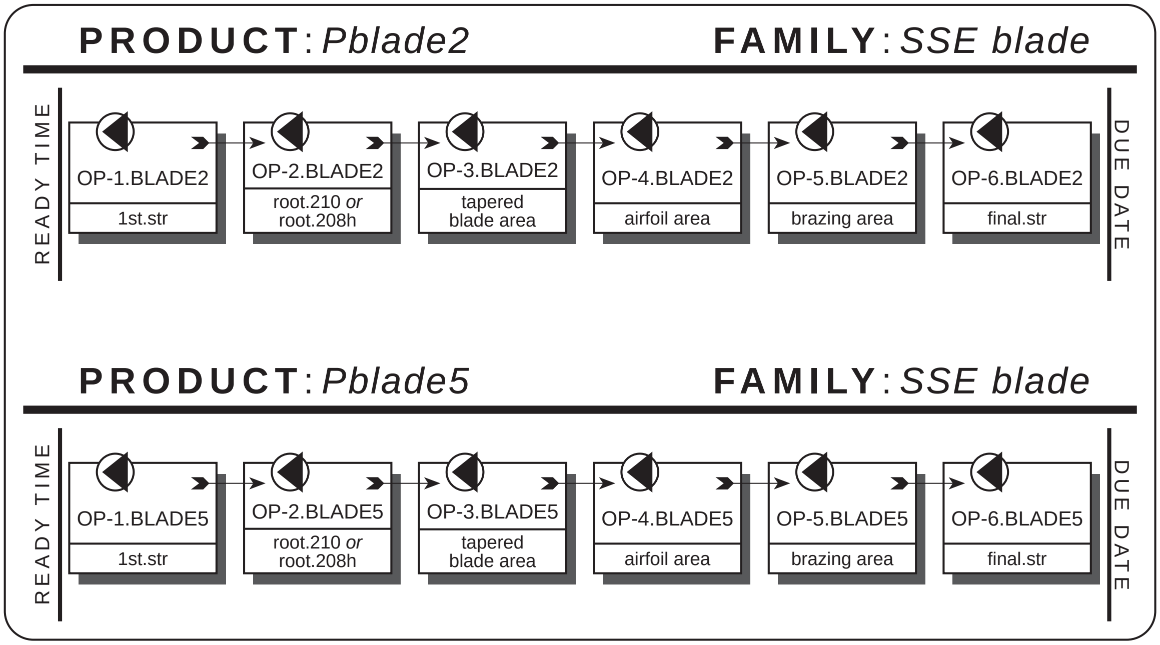


Figure A.32. Process Plans for the PRODUCE PBLADE2 & PRODUCE PBLADE5 Service Types.

```
;;; ----------------------------------------------------------------------
;;; SSE Blade Family Process Plan Definitions:

;;; PBlade2:

(define-process-plan PRODUCE_PBLADE2
   :SERVICES (:SEQUENCE op-1.blade2 op-2.blade2 op-3.blade2
                        op-4.blade2 op-5.blade2 op-6.blade2))

;;; PBlade5:

(define-process-plan PRODUCE_PBLADE5
   :SERVICES (:SEQUENCE op-1.blade5 op-2.blade5 op-3.blade5
                        op-4.blade5 op-5.blade5 op-6.blade5))

;;; ----------------------------------------------------------------------
```

Figure A.33. Specifications for the PRODUCE PBLADE2 & PRODUCE PBLADE5 Service Types.

A.2.7.3 T *Blade Family*

The process plan templates for the PRODUCE PBLADE3 and PRODUCE PBLADE6 service types are presented in Figure A.34. Both products (PBLADE3 and PBLADE6) are produced in a three-step serial process, with a choice of machine types to perform the second operation.

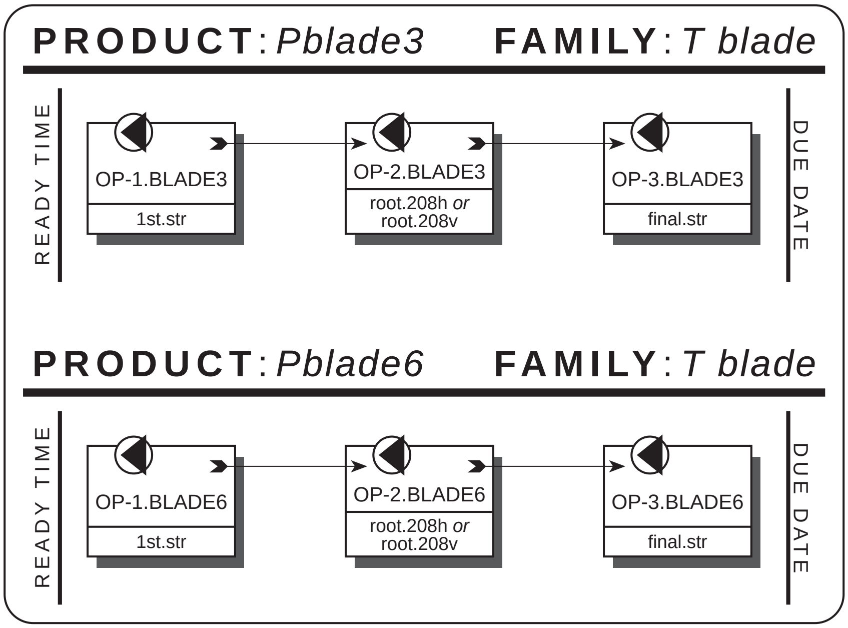


Figure A.34. Process Plans for the PRODUCE PBLADE3 & PRODUCE PBLADE6 Service Types.

Figure A.35 shows the specification of the process plan template that describes the PRODUCE PBLADE3 and PRODUCE PBLADE6 service types.

```
;;; ---------------------------------------------------------------------
;;; T Blade Family Process Plan Definitions:

;;; PBlade3:

(define-process-plan PRODUCE_PBLADE3
   :SERVICES (:SEQUENCE op-1.blade3 op-2.blade3 op-3.blade3))

;;; PBlade6:

(define-process-plan PRODUCE_PBLADE6
   :SERVICES (:SEQUENCE op-1.blade6 op-2.blade6 op-3.blade6))

;;; ---------------------------------------------------------------------
```

Figure A.35. Specifications for the PRODUCE PBLADE3 & PRODUCE PBLADE6 Service Types.

APPENDIX B

ANNOTATED DSS EXECUTION TRACE

This appendix presents a sequence of annotated execution trace segments from the output of DSS (via the ARM application system). In the trace segments, we have chosen to focus on one particular flight, and the satisfaction of some of its service goals. The system is run in dispatch mode, for a simulated time period of roughly 4 hours. One resource failure occurs, approximately 3 hours into the scheduling process. A timetable consisting of ten flights (with four inter-order connections), is used in conjunction with a sufficiently-stocked airport layout.

Each line in the trace corresponds to a particular problem-solving event. DSS starts by executing the ARM frontend function to initialize the APPLICATION DATA blackboard by loading the airport layout and the timetable. The `Queuing Flight Activity` event signals the insertion of the ten *order notification* events, one for each flight, onto the pending world event queue. The scheduling process now commences with the processing of the earliest *order notification* event, which corresponds to `<[Flight #235/235(Mon)D]-#0>`. The *Instantiate Task Network* knowledge source is immediately triggered, and then activated for execution.

```
=============================================================================
Initialization --------> GBB
KS Activation ---------> FRONTEND NIL
--------------------------------- KSA 1 ---------------------------------
KS Invocation ---------> FRONTEND (SYSTEM-INITIALIZATION-EVENT)
Calling Application Frontend Function --> FRONTEND-KS-FUNCTION.
DSS Application Message --> Using AIRPORT: '(DTW) Standard Allocation'.
DSS Application Message --> Using TIMETABLE: '10 flights (4 connections)'.
Queuing Flight Activity --> 10 flights (interval: [Mon 12:00am] to [Sun 11:59pm]).
Order Received --------> [Mon 1:36pm] <[Flight #235/235(Mon)D]-#0>
                         (Release Time: [Mon 3:36pm] - Due Date: [Mon 4:20pm])
                         TURNAROUND <Application priority: 1>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
--------------------------------- KSA 2 ---------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #235/235(Mon)D]-#0>)
```

⋮

The actions of the *Instantiate Task Network* knowledge source trigger the creation and activation of all required service goals for an order. Since no other orders have been received at this time, DSS proceeds immediately with the process of satisfying all existing service goals to produce a schedule for `<[Flight #235/235(Mon)D]-#0>`.

After a schedule has been completed for `<[Flight #235/235(Mon)D]-#0>`, the pending world event queue is processed, and the *order notification* event corresponding to `<[Flight #516/516(Mon)D]-#0>` is received. The *Instantiate Task Network* knowledge source is immediately triggered, and then activated for execution.

```
                                                            ⋮

Order Received --------> [Mon 1:39pm] <[Flight #516/516(Mon)D]-#0>
                         (Release Time: [Mon 3:39pm] - Due Date: [Mon 4:25pm])
                         TURNAROUND <Application priority: 1>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #516/516(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #516/516(Mon)D]-#0>)
-------------------------------- KSA 37 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #516/516(Mon)D]-#0>)

                                                            ⋮
```

Again, since no other orders have been received at this time, DSS proceeds immediately with the process of satisfying all existing service goals to produce a schedule for `<[Flight #516/516(Mon)D]-#0>`. After a schedule has been completed for `<[Flight #516/516(Mon)D]-#0>`, the pending world event queue is processed, and the *order notification* events corresponding to `<[Flight #1412/1412(Mon)D]-#0>`, `<[Flight #416/416(Mon)I]-#0>`, and `<[Flight #3039/3052(Mon)D]-#0>` are received. The *Instantiate Task Network* knowledge source is immediately triggered (once for each order), and three activations are then instantiated for execution.

```
                                     ⋮

Order Received --------> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         (Release Time: [Mon 3:40pm] - Due Date: [Mon 4:25pm])
                         TURNAROUND  <Application priority: 1>.
Order Received --------> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         (Release Time: [Mon 3:40pm] - Due Date: [Mon 4:25pm])
                         TURNAROUND  <Application priority: 1>.
Order Received --------> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         (Release Time: [Mon 3:40pm] - Due Date: [Mon 4:30pm])
                         TURNAROUND  <Application priority: 1>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #1412/1412(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #1412/1412(Mon)D]-#0>)
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #416/416(Mon)I]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #416/416(Mon)I]-#0>)
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #3039/3052(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #3039/3052(Mon)D]-#0>)
-------------------------------- KSA 53 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #1412/1412(Mon)D]-#0>)

                                     ⋮

-------------------------------- KSA 54 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #416/416(Mon)I]-#0>)

                                     ⋮
```

After task networks have been instantiated for <[Flight #1412/1412(Mon)D]-#0> and <[Flight #416/416(Mon)D]-#0>, DSS begins performing the same activities for <[Flight #3039/3052(Mon)D]-#0>. Since <[Flight #3039/3052(Mon)D]-#0> is connected to <[Flight #370/370(Mon)D]-#0>, the *Instantiate Task Network* knowledge source automatically arranges for the scheduling of both orders.

```
-------------------------------- KSA 55 --------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #3039/3052(Mon)D]-#0>)
Created Service Goal --> [Mon 1:40pm] [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#53 (GATE)
                         ([Mon 3:40pm] to [Mon 4:29pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [SERVICE]-SERVICE-GOAL-#54 (SERVICE-TRUCK)
                         ([Mon 3:43pm] to [Mon 4:26pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [RESTOCK]-SERVICE-GOAL-#55 (CATERING-TRUCK)
                         ([Mon 3:43pm] to [Mon 4:26pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#56 (BAGGAGE-TRUCK)
                         ([Mon 3:59pm] to [Mon 4:26pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [UNLOAD-BAGGAGE]-SERVICE-GOAL-#57 (BAGGAGE-TRUCK)
                         ([Mon 3:43pm] to [Mon 4:20pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [CLEAN]-SERVICE-GOAL-#58 (CLEANING-TRUCK)
                         ([Mon 3:47pm] to [Mon 4:20pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [REFUEL]-SERVICE-GOAL-#59 (FUEL-TRUCK PUMP-TRUCK)
                         ([Mon 3:47pm] to [Mon 4:20pm]) <[Flight #3039/3052(Mon)D]-#0>.
```

```
Connected Order -------> [Mon 1:40pm] Scheduling for <[Flight #370/370(Mon)D]-#0>
                         (owing to its connection with <[Flight #3039/3052(Mon)D]-#0>).
Created Service Goal --> [Mon 1:40pm] [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#60 (GATE)
                         ([Mon 4:40pm] to [Mon 5:24pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [SERVICE]-SERVICE-GOAL-#61 (SERVICE-TRUCK)
                         ([Mon 4:44pm] to [Mon 5:20pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [RESTOCK]-SERVICE-GOAL-#62 (CATERING-TRUCK)
                         ([Mon 4:44pm] to [Mon 5:20pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         ([Mon 4:54pm] to [Mon 5:20pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [UNLOAD-BAGGAGE]-SERVICE-GOAL-#64 (BAGGAGE-TRUCK)
                         ([Mon 4:44pm] to [Mon 5:10pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 (BAGGAGE-TRUCK)
                         ([Mon 3:43pm] to [Mon 5:20pm]) <[Flight #3039/3052(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [CLEAN]-SERVICE-GOAL-#66 (CLEANING-TRUCK)
                         ([Mon 4:50pm] to [Mon 5:12pm]) <[Flight #370/370(Mon)D]-#0>.
Created Service Goal --> [Mon 1:40pm] [REFUEL]-SERVICE-GOAL-#67 (FUEL-TRUCK PUMP-TRUCK)
                         ([Mon 4:50pm] to [Mon 5:12pm]) <[Flight #370/370(Mon)D]-#0>.
```

The service goal creation process causes the new goals, and any existing service goals whose urgency is affected by the increased contention resulting from the new goals, to be placed onto a list of service goals to be rated (and re-rated). The (re-)rating process commences after the *Instantiate Task Network* knowledge source has finished executing.

```
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#67 (FUEL-TRUCK PUMP-TRUCK) {487}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#45 (FUEL-TRUCK PUMP-TRUCK)
                         from {182} to {186}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#37 (FUEL-TRUCK PUMP-TRUCK)
                         from {1992} to {2728}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#66 (CLEANING-TRUCK) {172}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#36 (CLEANING-TRUCK)
                         from {1798} to {3401}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 (BAGGAGE-TRUCK) {9680}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#64 (BAGGAGE-TRUCK) {2880}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK) {1457}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#42 (BAGGAGE-TRUCK)
                         from {1396} to {1333}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#41 (BAGGAGE-TRUCK)
                         from {604} to {616}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#34 (BAGGAGE-TRUCK)
                         from {4906} to {4480}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#33 (BAGGAGE-TRUCK)
                         from {4243} to {4264}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [RESTOCK]-SERVICE-GOAL-#62 (CATERING-TRUCK) {115}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [RESTOCK]-SERVICE-GOAL-#32 (CATERING-TRUCK)
                         from {351} to {498}.
```

```
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#61 (SERVICE-TRUCK) {115}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#31 (SERVICE-TRUCK)
                         from {532} to {755}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#60 (GATE) {3772}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #381/381(Mon)D]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#30 (GATE)
                         from {3539} to {4645}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#38 (GATE)
                         from {2635} to {2489}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#59 (FUEL-TRUCK PUMP-TRUCK) {1907}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [REFUEL]-SERVICE-GOAL-#52 (FUEL-TRUCK PUMP-TRUCK)
                         from {3424} to {2638}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [REFUEL]-SERVICE-GOAL-#29 (FUEL-TRUCK PUMP-TRUCK)
                         from {3141} to {2910}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#58 (CLEANING-TRUCK) {2897}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [CLEAN]-SERVICE-GOAL-#51 (CLEANING-TRUCK)
                         from {6104} to {4084}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [CLEAN]-SERVICE-GOAL-#28 (CLEANING-TRUCK)
                         from {4580} to {4244}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#57 (BAGGAGE-TRUCK) {5947}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#56 (BAGGAGE-TRUCK) {5714}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#50 (BAGGAGE-TRUCK)
                         from {8894} to {7550}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#49 (BAGGAGE-TRUCK)
                         from {8593} to {7337}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#43 (BAGGAGE-TRUCK)
                         from {14733} to {14091}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#35 (BAGGAGE-TRUCK)
                         from {16797} to {15374}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#27 (BAGGAGE-TRUCK)
                         from {16096} to {14063}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#26 (BAGGAGE-TRUCK)
                         from {14362} to {12647}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [RESTOCK]-SERVICE-GOAL-#55 (CATERING-TRUCK) {459}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#54 (SERVICE-TRUCK) {879}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #416/416(Mon)I]-#0>
                         [SERVICE]-SERVICE-GOAL-#47 (SERVICE-TRUCK)
                         from {1635} to {1161}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [SERVICE]-SERVICE-GOAL-#24 (SERVICE-TRUCK)
                         from {1531} to {1444}.
Rated Service Goal ----> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#53 (GATE) {9275}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1412/1412(Mon)D]-#0>
                         [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#23 (GATE)
                         from {767} to {12595}.

                                           ⋮
```

The `SELECT-RESERVATION-SECURING-METHOD` knowledge source precondition is now triggered for each new service goal, resulting in a single activation of the knowledge source for each. This particular sequence of events for two of the service goals associated with `<[Flight #3039/3052(Mon)D]-#0>` and `<[Flight #370/370(Mon)D]-#0>` is presented below.

```
                                         ⋮

Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)

                                         ⋮

Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)

                                         ⋮
```

Each `SELECT-RESERVATION-SECURING-METHOD` knowledge source activation is immediately executed, resulting in the activation of a reservation-securing knowledge source(s) for each. (Multiple reservation-securing knowledge sources are activated when there is a choice of resource classes to perform the task involved.) This particular sequence of events for the same two service goals associated with `<[Flight #3039/3052(Mon)D]-#0>` and `<[Flight #370/370(Mon)D]-#0>` is presented below.

```
--------------------------------- KSA 90 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)
KS Activation ---------> STANDARD-ASSIGNMENT ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)

                                         ⋮

--------------------------------- KSA 96 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)

                                         ⋮
```

By now, the construction of the schedules for <[Flight #3039/3052(Mon)D]-#0> and <[Flight #370/370(Mon)D]-#0> has begun. We now follow the process in which the urgencies of [LOAD-BAGGAGE]-SERVICE-GOAL-#63 and [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 are re-rated as a result of other related scheduling decisions. Whenever a service goal is re-rated, all of its triggered knowledge source activations must also be re-rated.

```
--------------------------------- KSA 101 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#35)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-D1 assigned to
                         [Task Node: <Task: BAGGAGE-TRANSFER>
                                     <[Flight #1412/1412(Mon)D]-#0>]
                         ([Mon 3:36pm] - [Mon 5:05pm] ([Mon 3:44pm] - [Mon 5:05pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {1457} to {1864}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {1457} to {1864}.

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-142 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {9680} to {13762}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 (BAGGAGE-TRUCK)
                         from {9680} to {13762}.

                                             ⋮

--------------------------------- KSA 102 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#43)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-C3 assigned to
                         [Task Node: <Task: BAGGAGE-TRANSFER>
                                     <[Flight #1412/1412(Mon)D]-#0>]
                         ([Mon 3:36pm] - [Mon 5:21pm] ([Mon 3:44pm] - [Mon 5:21pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {1864} to {2856}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {1864} to {2856}.

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-142 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {13762} to {18762}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 (BAGGAGE-TRUCK)
                         from {13762} to {18762}.
```

```
                                             ⋮

-------------------------------- KSA 103 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([UNLOAD-BAGGAGE]-SERVICE-GOAL-#27)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO4 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE>
                                     <[Flight #1412/1412(Mon)D]-#0>]
                         ([Mon 3:36pm] - [Mon 4:07pm] ([Mon 3:44pm] - [Mon 3:52pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-142 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {18762} to {19735}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #3039/3052(Mon)D]-#0>
                         [BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 (BAGGAGE-TRUCK)
                         from {18762} to {19735}.

                                             ⋮
```

[BAGGAGE-TRANSFER]-SERVICE-GOAL-#65 is now ready to be satisfied using the standard *Assignment* reservation-securing knowledge source. The process of re-rating [LOAD-BAGGAGE]-SERVICE-GOAL-#63 and its triggered knowledge source activation continues.

```
-------------------------------- KSA 104 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([BAGGAGE-TRANSFER]-SERVICE-GOAL-#65)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO3B assigned to
                         [Task Node: <Task: BAGGAGE-TRANSFER>
                                     <[Flight #3039/3052(Mon)D]-#0>]
                         ([Mon 3:35pm] - [Mon 5:20pm] ([Mon 3:43pm] - [Mon 5:20pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {2856} to {4104}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {2856} to {4104}.

                                             ⋮

-------------------------------- KSA 109 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#33)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO4 assigned to
                         [Task Node: <Task: LOAD-BAGGAGE>
                                     <[Flight #381/381(Mon)D]-#0>]
                         ([Mon 4:36pm] - [Mon 5:05pm] ([Mon 4:56pm] - [Mon 5:05pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4104} to {3784}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {4104} to {3784}.
```

```
                                             ⋮

-------------------------------- KSA 115 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([UNLOAD-BAGGAGE]-SERVICE-GOAL-#34)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-E4 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE>
                                     <[Flight #381/381(Mon)D]-#0>]
                         ([Mon 4:26pm] - [Mon 4:57pm] ([Mon 4:34pm] - [Mon 4:42pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {3784} to {4716}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {3784} to {4716}.

                                             ⋮

-------------------------------- KSA 116 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([UNLOAD-BAGGAGE]-SERVICE-GOAL-#64)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO2 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE>
                                     <[Flight #370/370(Mon)D]-#0>]
                         ([Mon 4:36pm] - [Mon 5:07pm] ([Mon 4:44pm] - [Mon 4:52pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4716} to {4451}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {4716} to {4451}.

                                             ⋮
```

The following re-rating of `[LOAD-BAGGAGE]-SERVICE-GOAL-#63` and its triggered knowledge source activation comes about from the refinement of the reservation for the UNLOAD BAGGAGE task in `<[Flight #381/381(Mon)D]-#0>`, which frees up nine minutes of time in the resource schedule for `Baggage-Truck-#BT-E4`.

```
-------------------------------- KSA 119 --------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#86)
Refined Reservation ---> [Mon 1:40pm] Baggage-Truck-#BT-E4 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE> <[Flight #381/381(Mon)D]-#0>]
                         ([Mon 4:34pm] - [Mon 4:56pm] ([Mon 4:34pm] - [Mon 4:42pm]));
                         was ([Mon 4:26pm] - [Mon 4:57pm] ([Mon 4:34pm] - [Mon 4:42pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4451} to {4418}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {4451} to {4418}.
```

⋮

The satisfaction of [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#60 triggers the refinement process for all of its satisfied subgoals. The refinement of the reservation for the UNLOAD BAGGAGE task in <[Flight #370/370(Mon)D]-#0> frees up one minute of time in the resource schedule for Baggage-Truck-#BT-BO2. The refinement of the reservation for the BAGGAGE TRANSFER task in <[Flight #3039/3052(Mon)D]-#0> (connecting with <[Flight #370/370(Mon)D]-#0>) frees up no time, because a gate for <[Flight #3039/3052(Mon)D]-#0> has not yet been secured. The existing reservation must therefore be left unrefined, still containing enough time for any gate to be selected.

```
-------------------------------- KSA 121 --------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#60)
Resource Assignment ---> [Mon 1:40pm] Gate-#F5 assigned to
                         [Task Node: <Task: TURNAROUND-ACTIVITY>
                                     <[Flight #370/370(Mon)D]-#0>]
                         ([Mon 4:35pm] - [Mon 5:24pm] ([Mon 4:40pm] - [Mon 5:24pm])).

                                         ⋮

Invoked Precondition --> KS RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#115)
KS Activation ---------> RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#115)
Invoked Precondition --> KS RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)
KS Activation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)
-------------------------------- KSA 122 --------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([UNLOAD-BAGGAGE]-REFINE-GOAL-#115)
Refined Reservation ---> [Mon 1:40pm] Baggage-Truck-#BT-BO2 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE> <[Flight #370/370(Mon)D]-#0>]
                         ([Mon 4:36pm] - [Mon 5:06pm] ([Mon 4:44pm] - [Mon 4:52pm]));
                         was ([Mon 4:36pm] - [Mon 5:07pm] ([Mon 4:44pm] - [Mon 4:52pm])).
Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4418} to {4384}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {4418} to {4384}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-124 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#41
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {2329} to {2312}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#41 (BAGGAGE-TRUCK)
                         from {2329} to {2312}.
Rerated KSA -----------> [Mon 1:40pm] <KSA-125 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [UNLOAD-BAGGAGE]-SERVICE-GOAL-#42
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4527} to {4500}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #1044/1044(Mon)D]-#0>
                         [UNLOAD-BAGGAGE]-SERVICE-GOAL-#42 (BAGGAGE-TRUCK)
                         from {4527} to {4500}.
-------------------------------- KSA 123 --------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)
```

A final re-rating of `[LOAD-BAGGAGE]-SERVICE-GOAL-#63` and its triggered knowledge source activation precedes its solution by the standard *Assignment* knowledge source.

```
--------------------------------- KSA 124 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([UNLOAD-BAGGAGE]-SERVICE-GOAL-#42)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-E4 assigned to
                         [Task Node: <Task: UNLOAD-BAGGAGE>
                                     <[Flight #1044/1044(Mon)D]-#0>]
                         ([Mon 4:57pm] - [Mon 5:18pm] ([Mon 5:04pm] - [Mon 5:09pm])).

                                             ⋮

Rerated KSA -----------> [Mon 1:40pm] <KSA-149 STANDARD-ASSIGNMENT>
                         (TRIGGER-STANDARD-ASSIGNMENT-KS-EVENT
                          [LOAD-BAGGAGE]-SERVICE-GOAL-#63
                          BAGGAGE-TRUCK STANDARD-ASSIGNMENT)
                         from {4384} to {11960}.
Rerated Service Goal --> [Mon 1:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {4384} to {11960}.

                                             ⋮

--------------------------------- KSA 125 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO1 assigned to
                         [Task Node: <Task: LOAD-BAGGAGE>
                                     <[Flight #370/370(Mon)D]-#0>]
                         ([Mon 4:56pm] - [Mon 5:20pm] ([Mon 5:11pm] - [Mon 5:20pm])).

                                             ⋮
```

When `[TURNAROUND-ACTIVITY]-SERVICE-GOAL-#53` is finally satisfied, thereby producing a gate assignment for `<[Flight #3039/3052(Mon)D]-#0>`, the refinement of the reservation for the UNLOAD BAGGAGE task in `<[Flight #3039/3052(Mon)D]-#0>` (connecting with `<[Flight #370/370(Mon)D]-#0>`) can finally occur. Six minutes of time are freed up in the resource schedule for `Baggage-Truck-#BT-BO1`.

The satisfaction of `[TURNAROUND-ACTIVITY]-SERVICE-GOAL-#53` also removes the need for the `GOTO-BAGGAGE-OUTLET-TO-LOAD` segment of the LOAD BAGGAGE task in `<[Flight #381/381(Mon)D]-#0>`, because `Baggage-Truck-#BT-BO4`, which performs the UNLOAD BAGGAGE operation for `<[Flight #3039/3052(Mon)D]-#0>`, will now be left at the exact BAGGAGE OUTLET location where it can begin performing its previously-assigned, LOAD BAGGAGE task for `<[Flight #381/381(Mon)D]-#0>`.

```
--------------------------------- KSA 141 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([TURNAROUND-ACTIVITY]-SERVICE-GOAL-#53)
Succeeding Reservation Refined --> [Mon 1:40pm] Resized GOTO-BAGGAGE-OUTLET-TO-LOAD in
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #381/381(Mon)D]-#0>]
                         on Baggage-Truck-#BT-BO4 (CLEARED).
Resource Assignment ---> [Mon 1:40pm] Gate-#F3 assigned to
                         [Task Node: <Task: TURNAROUND-ACTIVITY>
                                     <[Flight #3039/3052(Mon)D]-#0>]
                         ([Mon 3:35pm] - [Mon 4:29pm] ([Mon 3:40pm] - [Mon 4:29pm])).

                                             ⋮
```

```
Invoked Precondition --> KS RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)
KS Activation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)

                                        ⋮

--------------------------------- KSA 143 ---------------------------------
KS Invocation ---------> RESERVATION-REFINEMENT ([BAGGAGE-TRANSFER]-REFINE-GOAL-#107)
Refined Reservation ---> [Mon 1:40pm] Baggage-Truck-#BT-BO3B assigned to
                         [Task Node: <Task: BAGGAGE-TRANSFER>
                                     <[Flight #3039/3052(Mon)D]-#0>]
                         ([Mon 3:41pm] - [Mon 5:20pm] ([Mon 3:43pm] - [Mon 5:20pm]));
                         was ([Mon 3:35pm] - [Mon 5:20pm] ([Mon 3:43pm] - [Mon 5:20pm])).

                                        ⋮
```

As a result of the following assignment, the `GOTO-BAGGAGE-OUTLET-TO-LOAD` segment of the recently-solved LOAD BAGGAGE task in `<[Flight #370/370(Mon)D]-#0>` must be increased in size by one minute, to account for a change in the expected positioning of `Baggage-Truck-#BT-BO1`.

```
--------------------------------- KSA 155 ---------------------------------
KS Invocation ---------> STANDARD-RIGHT-SHIFT ([LOAD-BAGGAGE]-SERVICE-GOAL-#56)
Succeeding Reservation Refined --> [Mon 1:40pm] Resized GOTO-BAGGAGE-OUTLET-TO-LOAD in
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #370/370(Mon)D]-#0>]
                         on Baggage-Truck-#BT-BO1 changed to
                         ([Mon 4:55pm] [Mon 4:56pm]) from ([Mon 4:56pm] [Mon 4:56pm]).
Resource Assignment ---> [Mon 1:40pm] Baggage-Truck-#BT-BO1 assigned to
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #3039/3052(Mon)D]-#0>]
                         ([Mon 4:16pm] - [Mon 4:30pm] ([Mon 4:25pm] - [Mon 4:30pm])).

                                        ⋮
```

After the schedules for the first nine flights have been completed, the pending world event queue is processed, and the *order notification* event corresponding to `<[Flight #243/243(Mon)D]-#0>` is received. The *Instantiate Task Network* knowledge source is immediately triggered, and then activated for execution. Notice also that immediately following the notification of DSS, `<[Flight #243/243(Mon)D]-#0>` departs from Fort Lauderdale (on time).

```
                                        ⋮

Order Received --------> [Mon 1:45pm] <[Flight #243/243(Mon)D]-#0>
                         (Release Time: [Mon 4:47pm] - Due Date: [Mon 5:30pm])
                         TURNAROUND  <Application priority: 1>.
Approaching Takeoff ---> [Mon 1:45pm] [Flight #243/243(Mon)D]
                         <[FLL] Fort Lauderdale, FL> DC-10-#10 <ON TIME>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #243/243(Mon)D]-#0>)
KS Activation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #243/243(Mon)D]-#0>)
--------------------------------- KSA 168 ---------------------------------
KS Invocation ---------> INSTANTIATE-TASK-NETWORK (<[Flight #243/243(Mon)D]-#0>)
```

⋮

After a schedule has been completed for `<[Flight #243/243(Mon)D]-#0>`, the schedule for all ten flights is finished. The remaining orders received by DSS have already been processed, owing to their previously-encountered connections to other orders. The precondition for the *Instantiate Task Network* knowledge source is immediately executed for each order, but the knowledge source is not activated.

⋮

```
Approaching Takeoff ---> [Mon 1:59pm] [Flight #235/235(Mon)D]
                         <[DCA] Washington, DC/Baltimore, MD [National]> A300-#1
                         <ON TIME>.
Approaching Takeoff ---> [Mon 2:00pm] [Flight #416/416(Mon)I]
                         <[MSP] Minneapolis/St. Paul, MN> DC-9-Series-20-#6 <ON TIME>.
Order Received --------> [Mon 2:30pm] <[Flight #381/381(Mon)D]-#0>
                         (Release Time: [Mon 4:30pm] - Due Date: [Mon 5:10pm])
                         TURNAROUND  <Application priority: 1>.
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #381/381(Mon)D]-#0>)
Quiescence ------------> Pending KSA Queue
Order Received --------> [Mon 2:35pm] <[Flight #1447/1447(Mon)D]-#0>
                         (Release Time: [Mon 4:37pm] - Due Date: [Mon 5:20pm])
                         TURNAROUND  <Application priority: 1>.
Approaching Takeoff ---> [Mon 2:35pm] [Flight #516/516(Mon)D]
                         <[CVG] Cincinnati, OH> F-28-Friendship-#7 <ON TIME>.
Approaching Takeoff ---> [Mon 2:35pm] [Flight #1447/1447(Mon)D]
                         <[PVD] Providence, RI> DC-9-Series-20-#2 <ON TIME>.
Quiescence ------------> Pending KSA Queue
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #1447/1447(Mon)D]-#0>)
Order Received --------> [Mon 2:40pm] <[Flight #370/370(Mon)D]-#0>
                         (Release Time: [Mon 4:40pm] - Due Date: [Mon 5:25pm])
                         TURNAROUND  <Application priority: 1>.
Approaching Takeoff ---> [Mon 2:40pm] [Flight #1412/1412(Mon)D]
                         <[MDW] Chicago, IL [Midway]> A300-#3 <ON TIME>.
Approaching Takeoff ---> [Mon 2:40pm] [Flight #3039/3052(Mon)D]
                         <[CAK] Akron/Canton, OH> SA-226TC-Metro-#8 <ON TIME>.
Quiescence ------------> Pending KSA Queue
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #370/370(Mon)D]-#0>)
Order Received --------> [Mon 2:45pm] <[Flight #1044/1044(Mon)D]-#0>
                         (Release Time: [Mon 4:45pm] - Due Date: [Mon 5:25pm])
                         TURNAROUND  <Application priority: 1>.
Quiescence ------------> Pending KSA Queue
Invoked Precondition --> KS INSTANTIATE-TASK-NETWORK (<[Flight #1044/1044(Mon)D]-#0>)
```

⋮

The schedule now begins executing. The following trace segment includes the remaining Approaching Takeoff events, and the execution of the scheduled ground servicing activities for <[Flight #370/370(Mon)D]-#0>.

```
                                       ⋮

Approaching Takeoff ---> [Mon 3:25pm] [Flight #381/381(Mon)D]
                         <[GRB] Green Bay, WI> A300-#4 <ON TIME>.
Approaching Takeoff ---> [Mon 3:25pm] [Flight #370/370(Mon)D]
                         <[ORD] Chicago, IL [O'Hare]> A300-#9 <ON TIME>.

Started Unloading Transfer Baggage --> [Mon 3:43pm] Baggage-Truck-#BT-BO3B
                         [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Finished Unloading Transfer Baggage --> [Mon 3:46pm] Baggage-Truck-#BT-BO3B
                         [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Started Passenger Transfer --> [Mon 3:47pm] [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Finished Transferring Baggage --> [Mon 3:47pm] Baggage-Truck-#BT-BO3B
                         [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Finished Passenger Transfer --> [Mon 3:48pm] [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Approaching Takeoff ---> [Mon 3:55pm] [Flight #1044/1044(Mon)D]
                         <[GRR] Grand Rapids, MI> F-28-Friendship-#5 <ON TIME>.

Flight Takeoff --------> [Mon 4:25pm] A300-#1 [Flight #235/235(Mon)D].

Flight Takeoff --------> [Mon 4:30pm] A300-#3 [Flight #1412/1412(Mon)D].
Flight Takeoff --------> [Mon 4:30pm] DC-9-Series-20-#6 [Flight #416/416(Mon)I].
Flight Takeoff --------> [Mon 4:30pm] F-28-Friendship-#7 [Flight #516/516(Mon)D].

Flight Landing --------> [Mon 4:35pm] A300-#9 [Flight #370/370(Mon)D] taxiing to Gate-#F5.

Flight Takeoff --------> [Mon 4:39pm] SA-226TC-Metro-#8 [Flight #3039/3052(Mon)D].

                                       ⋮
```

At 4:40pm Monday, DSS receives notification that Baggage-Truck-#BT-BO1 has failed. The PROCESS-RESOURCE-FAILURE knowledge source is activated and executed immediately. The only reservation affected by the failure is for the LOAD BAGGAGE task in <[Flight #370/370(Mon)D]-#0>. This now-obsolete reservation is officially canceled, whereupon [LOAD-BAGGAGE]-SERVICE-GOAL-#63 is re-activated and re-rated.

A new reservation is finally found for `[LOAD-BAGGAGE]-SERVICE-GOAL-#63` by the standard *Right Shift* knowledge source, following the failure of the standard *Assignment* and standard *Preemption* knowledge sources. As a result of the right shift, the due date (departure time) for `<[Flight #370/370(Mon)D]-#0>` is pushed back twelve minutes, and the reservation of `Gate-#F5` must be extended accordingly.

⋮

```
Resource Breakdown ----> [Mon 4:40pm] Baggage-Truck-#BT-BO1 now unavailable.
Invoked Precondition --> KS PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
KS Activation ---------> PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
--------------------------------- KSA 188 ---------------------------------
KS Invocation ---------> PROCESS-RESOURCE-FAILURE (Baggage-Truck-#BT-BO1)
Canceled Reservation --> [Mon 4:40pm] Baggage-Truck-#BT-BO1 no longer assigned to
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #370/370(Mon)D]-#0>].
Rerated Service Goal --> [Mon 4:40pm] <[Flight #370/370(Mon)D]-#0>
                         [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         from {11960} to {233}.
```

⋮

```
Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
--------------------------------- KSA 189 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Invoked Precondition --> KS STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
--------------------------------- KSA 190 ---------------------------------
KS Invocation ---------> STANDARD-ASSIGNMENT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Allocation Failure ----> [Mon 4:40pm] 'STANDARD-ASSIGNMENT' KS unable to find any
                         'BAGGAGE-TRUCK' resources to satisfy STANDARD-ASSIGNMENT.
Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
--------------------------------- KSA 191 ---------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Invoked Precondition --> KS STANDARD-PREEMPTION ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> STANDARD-PREEMPTION ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
--------------------------------- KSA 192 ---------------------------------
KS Invocation ---------> STANDARD-PREEMPTION ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Allocation Failure ----> [Mon 4:40pm] 'STANDARD-PREEMPTION' KS unable to find any
                         'BAGGAGE-TRUCK' resources to satisfy STANDARD-PREEMPTION.
Invoked Precondition --> KS SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
```

```
--------------------------------- KSA 193 --------------------------------
KS Invocation ---------> SELECT-RESERVATION-SECURING-METHOD
                         ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Invoked Precondition --> KS STANDARD-RIGHT-SHIFT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
KS Activation ---------> STANDARD-RIGHT-SHIFT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
--------------------------------- KSA 194 --------------------------------
KS Invocation ---------> STANDARD-RIGHT-SHIFT ([LOAD-BAGGAGE]-SERVICE-GOAL-#63)
Modified Goal ---------> [Mon 4:40pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         allowance now ([Mon 5:06pm] [Mon 5:20pm]);
                         was ([Mon 4:54pm] [Mon 5:20pm]) <[Flight #370/370(Mon)D]-#0>.
Modified Goal ---------> [Mon 4:40pm] [LOAD-BAGGAGE]-SERVICE-GOAL-#63 (BAGGAGE-TRUCK)
                         allowance now ([Mon 5:06pm] [Mon 5:32pm]);
                         was ([Mon 5:06pm] [Mon 5:20pm]) <[Flight #370/370(Mon)D]-#0>.
Modified Goal ---------> [Mon 4:40pm] [TURNAROUND-ACTIVITY]-SERVICE-GOAL-#60 (GATE)
                         allowance now ([Mon 4:40pm] [Mon 5:36pm]);
                         was ([Mon 4:40pm] [Mon 5:24pm]) <[Flight #370/370(Mon)D]-#0>.
Modified Due Date -----> [Mon 4:40pm] <[Flight #370/370(Mon)D]-#0>
                         due [Mon 5:37pm]; originally [Mon 5:25pm];
                         running [12 minutes] LATE  <Penalty: 12>.
Resource Assignment ---> [Mon 4:40pm] Baggage-Truck-#BT-BO4 assigned to
                         [Task Node: <Task: LOAD-BAGGAGE> <[Flight #370/370(Mon)D]-#0>]
                         ([Mon 5:06pm] - [Mon 5:32pm] ([Mon 5:23pm] - [Mon 5:32pm])).
Modified Reservation --> [Mon 4:40pm] [Task Node: <Task: TURNAROUND-ACTIVITY>
                         <[Flight #370/370(Mon)D]-#0>] Gate-#F5
                         ([Mon 4:35pm] - [Mon 5:36pm] ([Mon 4:40pm] - [Mon 5:36pm]));
                         was ([Mon 4:35pm] - [Mon 5:24pm] ([Mon 4:40pm] - [Mon 5:24pm])).
```

⋮

The execution of the remaining portion of the schedule now continues, finishing with the takeoff of `<[Flight #243/243(Mon)D]-#0>` at 5:47pm Monday.

⋮

```
Finished Airplane Shutdown --> [Mon 4:43pm] A300-#9 [Flight #370/370(Mon)D].

Started Deplaning Airplane --> [Mon 4:44pm] A300-#9 [Flight #370/370(Mon)D].

Started Restocking ----> [Mon 4:44pm] Catering-Truck-#KT-F4 A300-#9
                         [Flight #370/370(Mon)D].

Started Unloading Baggage --> [Mon 4:44pm] Baggage-Truck-#BT-BO2 A300-#9
                         [Flight #370/370(Mon)D].

Finished Deplaning Airplane --> [Mon 4:49pm] A300-#9 [Flight #370/370(Mon)D].

Started Refueling -----> [Mon 4:50pm] Fuel-Truck-#FT-F4 A300-#9 [Flight #370/370(Mon)D].

Started Cleaning ------> [Mon 4:51pm] Cleaning-Truck-#CT-E1 A300-#9
                         [Flight #370/370(Mon)D].

Finished Restocking ---> [Mon 4:53pm] Catering-Truck-#KT-F4 A300-#9
                         [Flight #370/370(Mon)D].

Finished Refueling ----> [Mon 5:00pm] Fuel-Truck-#FT-F4 A300-#9 [Flight #370/370(Mon)D].

Finished Servicing ----> [Mon 5:07pm] Service-Truck-#ST-G1 A300-#9
                         [Flight #370/370(Mon)D].

Flight Takeoff --------> [Mon 5:15pm] A300-#4 [Flight #381/381(Mon)D].
```

```
Finished Loading Transfer Baggage --> [Mon 5:20pm] Baggage-Truck-#BT-BO3B
                         [Flight #3039/3052(Mon)D] (SA-226TC-Metro-#8)
                         to [Flight #370/370(Mon)D] (A300-#9).

Started Loading Baggage --> [Mon 5:23pm] Baggage-Truck-#BT-BO4 A300-#9
                         [Flight #370/370(Mon)D].

Started Enplaning Airplane --> [Mon 5:25pm] A300-#9 [Flight #370/370(Mon)D].
Flight Takeoff --------> [Mon 5:25pm] DC-9-Series-20-#2 [Flight #1447/1447(Mon)D].

Flight Takeoff --------> [Mon 5:30pm] F-28-Friendship-#5 [Flight #1044/1044(Mon)D].

Finished Loading Baggage --> [Mon 5:32pm] Baggage-Truck-#BT-BO4 A300-#9
                         [Flight #370/370(Mon)D].

Started Airplane Prep --> [Mon 5:33pm] A300-#9 [Flight #370/370(Mon)D].
Finished Airplane Prep --> [Mon 5:36pm] A300-#9 [Flight #370/370(Mon)D].

Flight Departure ------> [Mon 5:37pm] A300-#9 [Flight #370/370(Mon)D]
                         from Gate-#F5 <[12 minutes] LATE>.

Flight Takeoff --------> [Mon 5:42pm] A300-#9 [Flight #370/370(Mon)D].

Flight Takeoff --------> [Mon 5:47pm] DC-10-#10 [Flight #243/243(Mon)D].
=============================================================================
***** Explicit :STOP returned by PROCESS-PENDING-SIMULATION-EVENT-QUEUE function *****
```

REFERENCES

[Adams *et al.*, 1988] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, March 1988.

[Babić, 1987] Obrad Babić. Optimization of refuelling truck fleets at an airport. *Transportation Research*, 21B(6):479–487, December 1987.

[Bellman *et al.*, 1982] R. Bellman, A. Esogbue, and I. Nabeshima. *Mathematical Aspects of Scheduling and Applications.* Pergamon Press, Oxford England, 1982.

[Blackboard Technology Group, 1992] Blackboard Technology Group. *GBB Reference: Version 2.1.* Amherst MA, September 1992.

[Boctor, 1990] Fayez F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49(1):3–13, November 1990.

[Brazile and Swigger, 1988] Robert P. Brazile and Kathleen M. Swigger. GATES: An airline gate assignment and tracking expert system. *IEEE Expert*, 3(2):33–39, Summer 1988.

[Carver and Lesser, 1992] Norman Carver and Victor Lesser. The evolution of blackboard control architectures. CMPSCI Technical Report 92-71, Department of Computer Science, University of Massachusetts, Amherst, October 1992.

[Chiang *et al.*, 1990] Whay-Yu Chiang, Mark S. Fox, and Peng Si Ow. Factory model and test data descriptions: OPIS experiments. Technical Report CMU-RI-TR-90-05, Center for Integrated Manufacturing and Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh PA, March 1990.

[Christofides *et al.*, 1987] Nicos Christofides, R. Alvarez-Valdes, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, June 1987.

[Coffman, 1976] E.G. Coffman, Jr., editor. *Computer and Job-Shop Scheduling Theory.* Wiley, New York, 1976.

[Corkill *et al.*, 1982] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlická. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings, Second National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh PA, August 1982. American Association for Artificial Intelligence (AAAI).

[Davis and Heidorn, 1971] Edward W. Davis and George E. Heidorn. An algorithm for optimal project scheduling under multiple resource constraints. *Management Science*, 17(12):B803–B816, August 1971.

[Davis and Patterson, 1975] Edward W. Davis and James H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21(8):944–955, April 1975.

[Davis, 1973] Edward W. Davis. Project scheduling under resource constraints—historical review and categorization of procedures. *AIIE Transactions*, 5(4):297–313, December 1973.

[Dechter and Pearl, 1988] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, January 1988.

[Erman *et al.*, 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[Field and Harrison, 1988] Anthony J. Field and Peter G. Harrison. *Functional Programming*. Addison-Wesley, Reading MA, 1988.

[Fisher, 1988] Marsha J. Fisher. Airport gate system is ready for arrival. *Datamation*, 34(13):21–25, 1 July 1988.

[Fox and Kempf, 1985] B.R. Fox and K.G. Kempf. Complexity, uncertainty and opportunistic scheduling. In *Proceedings, Second Conference on Artificial Intelligence Applications*, pages 487–492, Miami Beach FL, December 1985. Institute of Electrical and Electronics Engineers (IEEE).

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

[Fox *et al.*, 1982] Mark S. Fox, Brad P. Allen, and Gary A. Strohm. Job-shop scheduling: An investigation in constraint-directed reasoning. In *Proceedings, Second National Conference on Artificial Intelligence*, pages 155–158, Pittsburgh PA, August 1982. American Association for Artificial Intelligence (AAAI).

[Fox, 1983] Mark S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh PA, December 1983.

[French, 1982] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, Chichester England, 1982.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.

[Garey *et al.*, 1978] M.R. Garey, R.L. Graham, and D.S. Johnson. Performance guarantees for scheduling algorithms. *Operations Research*, 26(1):3–21, January–February 1978.

[Gosling, 1982] Geoffrey D. Gosling. An aircraft gate assignment computer program user guide. Research Report UCB-ITS-RR-82-8, Institute of Transportation Studies, University of California, Berkeley, June 1982.

[Gosling, 1990] Geoffrey D. Gosling. Design of an expert system for aircraft gate assignment. *Transportation Research*, 24A(1):59–69, January 1990.

[Graves, 1981] Stephen C. Graves. A review of production scheduling. *Operations Research*, 29(4):646–675, July–August 1981.

[Hamzawi, 1986] Salah G Hamzawi. Management and planning of airport gate capacity: A microcomputer-based gate assignment simulation model. *Transportation Planning and Technology*, 11(3):189–202, December 1986.

[Haralick and Elliott, 1980] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, October 1980.

[Held and Karp, 1962] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, March 1962.

[Keene, 1989] Sonya E. Keene. *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS*. Addison-Wesley, Reading MA, 1989.

[Kelley and Walker, 1959] James E. Kelley, Jr. and Morgan R. Walker. Critical-path planning and scheduling. In *Proceedings, Eastern Joint Computer Conference*, pages 160–173, Boston MA, December 1959. Joint Computer Conference.

[Kurtulus and Davis, 1982] I. Kurtulus and E.W. Davis. Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172, February 1982.

[Lawler and Wood, 1966] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, July–August 1966.

[Lawrence and Morton, 1993] Stephen R. Lawrence and Thomas E. Morton. Resource-constrained multi-project scheduling with tardy costs: Comparing myopic, bottleneck, and resource pricing heuristics. *European Journal of Operational Research*, 64(2):168–187, January 1993.

[Mangoubi and Mathaisel, 1985] R.S. Mangoubi and Dennis F.X. Mathaisel. Optimizing gate assignments at airport terminals. *Transportation Science*, 19(2):173–188, May 1985.

[Martin-Martin and Mary, 1986] T. Martin-Martin and D. Mary. CARPPA model: Optimization of aircraft positionning and staff regulation for Orly-West airport. In *Proceedings, 26th Annual AGIFORS Symposium*, pages 7–16, Bowness-on-Windermere England, October 1986. Airline Group of the International Federation of Operational Research Societies (AGIFORS).

[McKay *et al.*, 1988] Kenneth N. McKay, Frank R. Safayeni, and John A. Buzacott. Job-shop scheduling theory: What is relevant? *Interfaces*, 18(4):84–90, July–August 1988.

[Neiman *et al.*, 1994] Daniel E. Neiman, David W. Hildum, Victor R. Lesser, and Tuomas W. Sandholm. Exploiting meta-level information in a distributed scheduling system. In *Proceedings, Twelfth National Conference on Artificial Intelligence*, volume 1, pages 394–400, Seattle WA, August 1994. American Association for Artificial Intelligence (AAAI).

[Niland, 1970] Powell Niland. *Production Planning, Scheduling, and Inventory Control: A Text and Cases.* Macmillan, New York, 1970.

[Norbis and Smith, 1988] Mario I. Norbis and J. MacGregor Smith. A multiobjective, multi-level heuristic for dynamic resource constrained scheduling problems. *European Journal of Operational Research*, 33(1):30–41, January 1988.

[Noronha and Sarma, 1991] S.J. Noronha and V.V.S. Sarma. Knowledge-based approaches for scheduling problems: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):160–171, June 1991.

[Ow and Smith, 1986] Peng Si Ow and Stephen F. Smith. Towards an opportunistic scheduling system. In *Proceedings, Nineteenth Hawaii International Conference on System Sciences*, pages 345–353, Honolulu, January 1986.

[Ow and Smith, 1988] Peng Si Ow and Stephen F. Smith. Viewing scheduling as an opportunistic problem-solving process. *Annals of Operations Research*, 12:85–108, 1988.

[Ow *et al.*, 1988] Peng Si Ow, Stephen F. Smith, and Alfred Thiriez. Reactive plan revision. In *Proceedings, Seventh National Conference on Artificial Intelligence*, volume 1, pages 77–82, Saint Paul MN, August 1988. American Association for Artificial Intelligence (AAAI).

[Ow, 1986] Peng Si Ow. Experiments in knowledge-based scheduling. Technical report, The Robotics Institute, Carnegie Mellon University, Pittsburgh PA, April 1986. Forthcoming.

[Panwalkar and Iskander, 1977] S.S. Panwalkar and Wafik Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, January–February 1977.

[Papadimitriou and Steiglitz, 1982] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs NJ, 1982.

[Patterson, 1984] James H. Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30(7):854–867, July 1984.

[Pease, 1978] Marshall C. Pease, III. ACS.1: An experimental automated command support system. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(10):725–735, October 1978.

[Pritsker *et al.*, 1969] A. Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe. Multi-project scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93–108, September 1969.

[Riccio and Ron, 1985] Lawrence A. Riccio and Nathan Ron. Computer-generated system aids airline's passenger flow and routing of aircraft. *Industrial Engineering*, 17(9):52–56, September 1985.

[Rickel, 1988] Jeff Rickel. Issues in the design of scheduling systems. In Michael D. Oliff, editor, *Expert Systems and Intelligent Manufacturing*, pages 70–89. Elsevier, New York, 1988.

[Sadeh and Fox, 1990] Norman Sadeh and Mark S. Fox. Variable and value ordering heuristics for activity-based job-shop scheduling. In *Proceedings, Fourth International Conference on Expert Systems in Production and Operations Management*, pages 134–144, Hilton Head SC, May 1990.

[Sadeh *et al.*, 1993] Norman Sadeh, Shinichi Otsuka, and Robert Schnelbach. Predictive and reactive scheduling with the Micro-Boss production scheduling and control system. In *Proceedings, IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling and Control*, Chambery France, August 1993. International Joint Conferences on Artificial Intelligence (IJCAI).

[Sadeh, 1991] Norman Sadeh. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, March 1991.

[Schröder, 1972] Helmut Schröder. The assignment of aircraft to gate positions. In *Proceedings, 12th Annual AGIFORS Symposium*, pages 301–318, Nathanya Israel, October 1972. Airline Group of the International Federation of Operational Research Societies (AGIFORS).

[Shifrin, 1988] Carole A. Shifrin. Gate assignment expert system reduces delays at United's hubs. *Aviation Week & Space Technology*, pages 148–149, 25 January 1988.

[Simon, 1981] Herbert A. Simon. *Sciences of the Artificial.* MIT Press, Cambridge MA, second edition, 1981.

[Smith and Ow, 1985] Stephen F. Smith and Peng Si Ow. The use of multiple problem decompositions in time constrained planning tasks. In *Proceedings, Ninth International Joint Conference on Artificial Intelligence*, volume 2, pages 1013–1015, Los Angeles CA, August 1985. International Joint Conferences on Artificial Intelligence (IJCAI).

[Smith *et al.*, 1986a] Stephen F. Smith, Mark S. Fox, and Peng Si Ow. Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems. *AI Magazine*, 7(4):45–61, Fall 1986.

[Smith *et al.*, 1986b] Stephen F. Smith, Peng Si Ow, Claude Le Pape, Bruce McLaren, and Nicola Muscettola. Integrating multiple scheduling perspectives to generate detailed production plans. In *Proceedings, Conference on AI in Manufacturing*, pages 2/123–2/137, Long Beach CA, September 1986. Society of Manufacturing Engineers (SME).

[Smith *et al.*, 1990] Stephen F. Smith, Peng Si Ow, Nicola Muscettola, Jean-Yves Potvin, and Dirk C. Matthys. An integrated framework for generating and revising factory schedules. *Journal of the Operational Research Society*, 41(6):539–552, June 1990.

[Smith, 1987] Stephen F. Smith. A constraint-based framework for reactive management of factory schedules. In *Proceedings, International Conference on Expert Systems and the Leading Edge in Production Planning and Control*, pages 349–366, Charleston SC, May 1987.

[Smith, 1991] Stephen F. Smith. Knowledge-based production management: Approaches, results and prospects. Technical Report CMU-RI-TR-91-21, Center for Integrated Manufacturing Decision Systems, The Robotics Institute, Carnegie Mellon University, Pittsburgh PA, December 1991.

[Speranza and Vercellis, 1993] M. Grazia Speranza and Carlo Vercellis. Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, 64(2):312–325, January 1993.

[Steele, 1990] Guy L. Steele, Jr. *Common Lisp: The Language*. Digital Press, Bedford MA, second edition, 1990.

[Stefik, 1980] Mark J. Stefik. *Planning With Constraints*. PhD thesis, Department of Computer Science, Stanford University, Stanford CA, January 1980.

[Stinson *et al.*, 1978] Joel P. Stinson, Edward W. Davis, and Basheer M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10(3):252–259, September 1978.

[Tsubakitani and Deckro, 1990] Shigeru Tsubakitani and Richard F. Deckro. A heuristic for multi-project scheduling with limited resources in the housing industry. *European Journal of Operational Research*, 49(1):80–91, November 1990.

[Vepsaleinen, 1984] Ari Vepsaleinen. *State Dependent Priority Rules for Scheduling*. PhD thesis, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh PA, April 1984.

[Weiss and Kulikowski, 1984] Sholom M. Weiss and Casimir A. Kulikowski. *A Practical Guide to Designing Expert Systems*. Rowman and Alanheld, Totowa NJ, 1984.

[Westbrook *et al.*, 1992] David L. Westbrook, Scott D. Anderson, David M. Hart, and Paul R. Cohen. Common lisp instrumentation package: User manual. CMPSCI Technical Report 94-26, Department of Computer Science, University of Massachusetts, Amherst, 1992.

[Willis, 1985] R.J. Willis. Critical path analysis and resource constrained project scheduling—theory and practice. *European Journal of Operational Research*, 21(2):149–155, August 1985.