

**AFRL-IF-RS-TR-2004-142**  
**Final Technical Report**  
**June 2004**



## **SCHEDULING AND VISUALIZATION**

**Carnegie Mellon University**

**Sponsored by**  
**Defense Advanced Research Projects Agency**  
**DARPA Order No. F186**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

**The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.**

**AIR FORCE RESEARCH LABORATORY**  
**INFORMATION DIRECTORATE**  
**ROME RESEARCH SITE**  
**ROME, NEW YORK**

## **STINFO FINAL REPORT**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-142 has been reviewed and is approved for publication

APPROVED:

/s/  
WAYNE A. BOSCO  
Project Engineer

FOR THE DIRECTOR:

/s/  
JAMES A. COLLINS, Acting Chief  
Information Technology Division  
Information Directorate



# Table of Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>The AMC Allocator: Continuous Management of Airlift and Tanker Resources</b>	<b>6</b>
2.1	Introduction . . . . .	7
2.2	The AMC Barrel Master Allocation Problem . . . . .	9
2.3	Incremental Constraint-based Search Models . . . . .	13
2.4	The AMC Allocator . . . . .	15
2.4.1	Representational Assumptions . . . . .	16
2.4.2	Allocation of Airlift Assets . . . . .	20
2.4.3	Allocation of Air Refueling Missions . . . . .	24
2.4.4	Mission Combination . . . . .	26
2.4.5	Incremental Optimization . . . . .	27
2.5	Project History and Status . . . . .	31
2.6	Summary and Current Directions . . . . .	32
2.7	Acknowledgements . . . . .	33
<b>3</b>	<b>An Ontology for Constructing Scheduling Systems</b>	<b>34</b>
3.1	Ontologies and Model Building . . . . .	35
3.2	The Structure of Domain Models in OZONE . . . . .	36
3.2.1	Basic components of scheduling models and their relationships . . .	37
3.2.2	Associated Capabilities . . . . .	38

3.3	The OZONE Scheduling Ontology . . . . .	40
3.3.1	DEMANDS . . . . .	40
3.3.2	PRODUCTS . . . . .	41
3.3.3	RESOURCES . . . . .	42
3.3.4	ACTIVITIES . . . . .	47
3.3.5	CONSTRAINTS . . . . .	49
3.4	Concluding Remarks . . . . .	52
3.5	Acknowledgements . . . . .	53
<b>4</b>	<b>Interactive Visualizations for Requirements Analysis</b>	<b>54</b>
4.1	Introduction . . . . .	55
4.2	Visualization and Requirements Analysis . . . . .	57
4.3	Visual Interface . . . . .	61
4.4	Example Analysis . . . . .	69
4.4.1	Shortfall Analysis . . . . .	71
4.4.2	Excess Analysis . . . . .	73
4.5	Integration with Information Analysis . . . . .	75
4.6	Other Resource Models . . . . .	77
4.7	User-Guided Scheduling . . . . .	79
4.8	Related Work . . . . .	80
4.9	Summary . . . . .	81
<b>5</b>	<b>Incremental Resource Allocation and Scheduling for Effects-Based Operations</b>	<b>83</b>
5.1	Introduction . . . . .	84
5.2	Air Campaign Scheduler (ACS) . . . . .	85
5.3	The EBO Jump-Start System . . . . .	87
5.4	Results . . . . .	89

<b>6 Other Research Publications</b>	<b>91</b>
6.1 Configurable, Mixed-Initiative Planning and Scheduling Systems . . . . .	91
6.2 Temporally Flexible Scheduling Algorithms . . . . .	93
<b>Bibliography</b>	<b>97</b>

# List of Figures

2.1	The AMC planning and execution environment . . . . .	10
2.2	Constraint-Based Scheduling Models . . . . .	14
2.3	Basic search procedure for resource allocation . . . . .	21
2.4	Overall Mission Scheduling Procedure . . . . .	24
2.5	Generating Options for an Unassignable Mission . . . . .	25
2.6	Air Refueling AssignRequest Procedure . . . . .	26
2.7	Mission Combination Procedure . . . . .	28
2.8	Mission Swap Procedure . . . . .	30
3.1	Abstract Domain Model . . . . .	38
3.2	Layered models of scheduling subdomains . . . . .	39
4.1	A Gantt Chart . . . . .	57
4.2	A Closure Graph . . . . .	59
4.3	A Generalized Gantt Chart . . . . .	60
4.4	Capacity-based Closure Graph . . . . .	63
4.5	Visual computation of demand curve . . . . .	64
4.6	Subset of move requirements with $EAD \geq C12$ . . . . .	65
4.7	Interactive visualization using a sliding window . . . . .	66
4.8	3D Visualization of demand and capacity over time . . . . .	68
4.9	Port capacity shortfalls . . . . .	70

4.10	After re-routing, Phase 2 shortfalls are largely eliminated. . . . .	72
4.11	Identifying Excess Capacity . . . . .	74
4.12	The Visage Outliner drill-down table . . . . .	76
4.13	Thes Visage Interval chart . . . . .	77
5.1	ACS Functional Scope . . . . .	86
5.2	ACS Graphical User Interface . . . . .	87
5.3	Information flows between CAT, JTT and ACS in the EBO Jump Start System.	89

# Chapter 1

## Overview

In this final report, we summarize research performed under the Department of Defense Advanced Research Projects Agency (DARPA) Contract #F30602-97-2-0227. Broadly speaking this research has focused on the development of architectures and reusable tools for mixed-initiative scheduling in complex domains such as continuous transportation, logistics and air operations planning and scheduling. Building from previous research at Carnegie Mellon University in the areas of reactive, mixed-initiative scheduling and with collaborative support from Maya Design Group in the area of advanced data visualization and user interface development, our broad goal has been the development of core component capabilities for

- plan/schedule visualization, comparison and assessment,
- generation of robust schedules that anticipate unpredictability in execution,
- timely and minimally disruptive replanning and rescheduling in response to unexpected execution circumstances, and
- mixed-initiative management of the ongoing planning, scheduling and execution process.

Methodologically, our approach has been to use real applications to drive our research and technology development, and a major portion of our effort to demonstrate and validate core techniques has involved development of specific application systems. Following from our

previous work in the DARPA/Rome Laboratory (ARPI) Planning Initiative, we have continued to draw on military transportation and logistics planning and scheduling problems as a principal application focus. We have also built on previous work carried out within DARPA's "JFACC After Next" program, and utilized the domain of air campaign scheduling as an additional application focus. Finally, in validating the performance of core algorithms for building robust schedules we have also drawn on standard benchmark problems in the domain of Resource-Constrained Project Scheduling.

Our research has produced the following technical accomplishments:

- *The AMC Barrel Allocator* - One major accomplishment has been the development of the AMC Barrel Allocator, a system for day-to-day allocation of aircraft and aircrews to airlift and tanker missions at the USAF Air Mobility Command (AMC). Conceived initially as part of a Technology Integration Experiment (TIE) aimed at demonstrating the potential of advanced automated scheduling tools to AMC operations, a decision was made in January 1999 to transition the system into operations in the the Tanker/Airlift Control Center (TACC) at AMC, and the system has subsequently been transitioned into AMC's operational Consolidated Air Mobility Planning System (CAMPS). Although it is still too early to have "return on investment" data, estimates based on even a small reduction in either AMC's current outsourcing levels to the commercial airlines or AMC's fuel usage indicate projected savings of many millions of dollars.

The AMC Allocator supports solution of AMC's short-term airlift/tanker allocation problem. The system is designed to accept new mission requests generated by different planning offices at AMC as input, and based on current availability of contracted aircraft and aircrews, it is used to determine which missions can be supported and to generate wing assignments. It utilizes incremental, constraint-based search techniques to selectively re-optimize allocation decisions to accommodate new higher-priority missions while minimizing disruption to previous assignments. As resource assignments are made to a given mission, any necessary auxiliary tasks (e.g., positioning/depositioning flights, crew rest periods, etc.) are generated and inserted into the mission plan. In the simplest case, all missions are planned and scheduled as round trips. Various missions will be sequenced when necessary to satisfy overall resource capacity constraints (and in some cases rejected as unsupportable). It is also possible to direct the system to consider mission "merging" possibilities (e.g., "recycling" the aircraft

to support a second mission instead of returning directly back to home station), which provides another means for optimizing resource usage. The AMC Allocator's Mission scheduling and resource allocation capabilities can be invoked in various automated or semi-automated modes. In the latter case, the system generates different options that might be taken to support a given mission, and various visual displays are provided for evaluating and comparing generated alternatives. This mixed-initiative scheduling model and the techniques used to realize it are described in Chapter 2.

- *Configurable Scheduling Systems* - Underlying the construction of the AMC Barrel Allocator, our research has also made progress in the development of techniques and tools for rapid configuration of planning and scheduling applications. These results have been consolidated in OZONE, a tool-kit for constructing mixed-initiative planning and scheduling application systems that was initiated under prior research within the ARPI Planning Initiative and has been under continued development for several years [Smith *et al.*, 1996]. One principal contribution of our research under the current contract has been elaboration of the OZONE scheduling ontology. The OZONE ontology provides a conceptual framework for mapping a high-level domain analysis into an executable scheduling model. At its core, the OZONE scheduling framework provides a class library of components for configuring mixed-initiative planning and scheduling applications, and the OZONE ontology (or more precisely the *abstract domain model* that the ontology defines) dictates how these components fit together. Through specification of a *concrete domain model* in terms of the ontology for a given application, corresponding *capabilities* in the class library are identified and instantiated within the system's basic search architecture. Thus it is possible to very quickly arrive at an executable scheduling model. At this point effort can be focused directly on developing whatever problem specific heuristics or constraint models are required to achieve high performance in the target scheduling domain. The OZONE Ontology is described more fully in Chapter 3.

A complementary research accomplishment in the area of configurable scheduling systems has been the development of an interactive tool for application model construction in OZONE. This tool provides a basis for using the ontology to navigate (and if necessary extend) the underlying class library, and for subsequently compiling various components into an application system. This work is summarized in Chapter 6 and described more fully in [Becker, 1998].

- *Interactive Visualization for Requirements Analysis* - Another technical accomplishment of our research has been the development of interactive visualizations of resource capacity constraints for use in analyzing requirements in large-scale planning and scheduling domains. In domains as diverse as strategic deployment planning and manufacturing production management, detailed planning and scheduling is generally preceded by higher-level analysis of requirements in relation to available (or apportioned) resource capacity. The goal of this analysis is early detection of resource capacity shortfalls (as well as excesses), and subsequent adjustment of the constraints associated with infeasible requirements (if possible) to bring them more into line with capacity. In support of this goal, we have defined novel 2D and 3D visualizations that generalize simple Gantt chart and closure graph representations of resource usage over time to provide a high-level basis for identification of capacity shortfalls and overflows *in advance of* actual scheduling. Once requirements have been adjusted to match available resources, these same visualizations also appear useful as a tool for interactive plan and schedule generation. In Chapter 4, we describe these visualizations and their use in the context of strategic deployment, specifically supporting analysis of a set of movement requirements from the standpoint of throughput capacity at various ports.
- *Incremental Scheduling for Effects-Based Operations* - A second major application that has been built using the OZONE framework is ACS, an Air Campaign Scheduling system. ACS utilizes the same incremental scheduling technology that underlies the AMC Allocator and provides analogous capabilities for incrementally generating and maintaining air campaign schedules. Another accomplishment of our research has been to demonstrate the relevance of the ACS technology to the concept of “Effects-Based Operations”. A Technology Integration Experiment (TIE) aimed at coupling an effects-driven planning process (embodied in a Bayesian reasoning tool developed at the Air Force Research Laboratory (AFRL) called CAT - Causal Analysis Tool) with a dynamic scheduling capability (embodied by ACS) was successfully carried out to produce a “jump start” demonstration for AFRL’s Effects-Based Operations Advance Technology Demonstration Program. As one measure of success, the initial integration was accomplished within a three month period following an unsuccessful, one-year attempt by AFRL to integrate with another scheduling technology. This TIE is summarized in Chapter 5.

- *Generating Robust Schedules* - A final area of accomplishment of this research effort has been the development of core algorithms for generating temporally flexible schedules. A temporally flexible schedule is produced by adopting a disjunctive graph formulation of the scheduling problem, wherein resource conflicts are resolved by posting ordering constraints between competing activities (thus sequentializing usage of shared resources). A principal property of a schedule that is constructed in this way is that activity start and end times are not fixed to specific time points, but instead delineate a range (or interval) of feasible solutions. As such, these schedules provide a measure of *robustness* in an unpredictable execution environment.

Our research has produced a family of algorithms for generating temporally flexible schedules, and has demonstrated their effectiveness across a range of benchmark problems. One thread of work has produced scalable algorithms for solving Multi-Capacitated Job Shop Scheduling Problems (MCJSSP) [Cesta *et al.*, 1998a, Cesta *et al.*, 1998b, Cesta *et al.*, 1999a, Cesta *et al.*, 2000]. Another thread has extended this class of techniques to more general Resource-Constrained Project Scheduling Problems with Maximum Time Lags (RCPSP/Max)[Cesta *et al.*, 1999b, Cesta *et al.*, 2002]. For both problem classes, our techniques have been shown to outperform the current best known approximate solution techniques. These results are summarized in Chapter 6.

## Chapter 2

# The AMC Allocator: Continuous Management of Airlift and Tanker Resources

**Summary:** Efficient allocation of aircraft and aircrews to transportation missions is an important priority at the USAF Air Mobility Command (AMC), where airlift demand must increasingly be met with less capacity and at lower cost. In addition to presenting a formidable optimization problem, the AMC resource management problem is complicated by the fact that it is situated in a continuously executing environment. Mission requests are received (and must be acted upon) incrementally, and, once allocation decisions have been communicated to the executing agents, subsequent opportunities for optimizing resource usage must be balanced against the cost of solution change. In this paper, we describe the technical approach taken to this problem in the AMC Barrel Allocator, a scheduling tool developed to address this problem and provide support for day-to-day allocation and management of AMC resources. The system utilizes incremental and configurable constraint-based search procedures to provide a range of automated and semi-automated scheduling capabilities. Most basically, the system provides an efficient solution to the fleet scheduling problem. More importantly to continuous operations, it also provides techniques for selectively re-optimizing to accommodate higher priority missions while minimizing disruption to most previously scheduled missions, and for selectively “merging” previously planned missions to minimize non-productive flying time. In situations where all mission requirements cannot be met, the system can generate and compare alternative constraint relaxation options. The Barrel Allocator technology is currently transitioning into operational use within AMC’s Tanker/Airlift Control Center (TACC). A version of the Barrel Allocator supporting airlift allocation was first incorporated as an experimental module of the AMC’s Consolidated Air Mobility Planning System (CAMPS) in September 2000. In June 2003, a new tanker allocation module was delivered for initial operational release to users as part of CAMPS Release 5.4.<sup>1</sup>

---

<sup>1</sup>This chapter has been accepted for publication and will appear as: Stephen F. Smith, Marcel A. Becker, and Laurence A. Kramer, “Continuous Management of Airlift and Tanker Resources: A Constraint-Based Ap-

## 2.1 Introduction

Efficient utilization of transportation resources is central to the effectiveness of operations at the USAF Air Mobility Command (AMC). In normal day-to-day operations, airlift demand must increasingly be met with less capacity and at lower cost. In crisis situations, rapid, large-scale deployment is crucial to overall military success. The allocation of aircraft and aircrews to airlift and tanker missions is a challenging problem. Several thousand missions are typically flown worldwide on a weekly basis, involving several hundreds of aircraft and comparable numbers of air crews. Individual missions impose a myriad of temporal constraints on their execution, and these constraints must be reconciled with resource availability and usage constraints (e.g., crew duty day restrictions and scheduled return dates, aircraft speed, range, and capacity) to find feasible assignments. Generally there are many more mission requests than can be accommodated by available assets in any given time frame, and hence an ability to optimize resource usage contributes directly to reduced reliance on higher-cost, commercial transportation assets.

Like many practical planning and scheduling problems, resource management at AMC is further complicated by the fact that it is situated in a continuous planning and execution environment. New mission requests of varying priority enter the system every day, and the current schedule is constantly evolving. Since taskings are incrementally communicated to the executing air wings, new requirements must be integrated into the current schedule without wholesale disruption to previous assignments. Likewise, the dynamics of execution regularly force changes to planned activities. Aircraft break down, airports become unavailable due to weather, missions become delayed due to diplomatic clearance problems, etc., and all such events can warrant reassessment of previous allocation decisions. In such execution-driven rescheduling contexts, it is similarly important to remain sensitive to solution stability concerns.

A third distinguishing aspect of the AMC allocation problem is the need for flexible accommodation of and integration with human decision-making. Given the scale and complexity of the AMC environment, there will always be user knowledge that is outside of system models and hence it must always be possible for the user to over-ride, constrain, bias, or otherwise direct system problem solving processes. While problem scale necessitates

---

proach”, *Journal of Computer and Mathematical Modeling - Special Issue on Defense Transportation: Algorithms, Models and Applications for the 21st Century*, 39(6-8), 2004.

automation, effective problem solving also requires flexible user intervention.

These problem requirements are at direct odds with the design of most current transportation planning and scheduling tools. Current tools tend to be organized as black-box, batch-oriented solution generators which, when invoked, re-solve the problem from scratch. This presents three fundamental problems. First, there is no memory of solutions from one run to the next, and small changes in inputs can lead to large changes in outputs. Hence, they are difficult to control and use in circumstances where localized change is necessary or advantageous. Second, the computational cost of batch-oriented solution procedures makes it difficult for planning to keep pace with execution in dynamic (or higher tempo) circumstances, forcing the traditional disconnect between these two processes. Finally, in user-driven (and mixed-initiative) problem solving contexts, the “specify and solve” pattern of user-interaction promoted by batch-oriented solvers is inherently inefficient.

In this paper, we describe an alternative approach implemented in the AMC Allocator, a tool for day-to-day allocation of aircraft and aircrews to airlift and tanker missions. In contrast to the above mentioned tools, the AMC Allocator is designed specifically for continuous operations, and provides a range of automated and semi-automated capabilities for allocating resources to missions against the backdrop of a pre-existing airlift schedule. Underlying the AMC Allocator is a novel constraint-based search approach to transportation scheduling. This approach is *incremental* by nature, and this characteristic is key to the Allocator’s effectiveness as a continuous planning tool. The incrementality of the approach provides a direct basis for localizing and controlling solution change. It also enables real-time response to user rescheduling directives. The AMC Allocator has been positively evaluated by AMC personnel and is currently transitioning into operational use within the Tanker Airlift Control Center (TACC) at AMC. It was first embedded as an experimental component of AMC’s Consolidated Air Mobility Planning System (CAMPS) in 2000, and an enhanced tanker allocation module is scheduled for operational release in May 2003.

The remainder of the paper is organized as follows. In section 2.2 we summarize the allocation problem faced by AMC. In section 2.3 we contrast identified problem requirements with traditional solution approaches, and consider the advantages of incremental, constraint-based search models as a basis for a more suitable approach. The solution framework developed within the AMC Allocator is then described in Section 2.4. We first review basic representational assumptions and solution generation procedures, along with the functional capabilities they give rise to. We then discuss some layered solution optimization capabilities

ties. In Section 2.5, we summarize the history of the AMC Allocator Project and indicate the current status of the work. Finally, in Section 2.6, we indicate the current directions of our research and development efforts.

## 2.2 The AMC Barrel Master Allocation Problem

The overall mission planning, scheduling and execution process in the TACC at AMC is depicted in Figure 2.1. Customer requirements flow into several distinct planning offices which respond by generating *missions*. A mission typically involves the movement of cargo and/or personnel, and each planning office generates air missions of a specific type. The channel planning office, for example, is responsible for planning channel missions, which correspond closely to the types of routes flown by commercial airlines; these missions are established and revised periodically, and then flown repeatedly at regular (e.g., daily or weekly) intervals. Special Assignment Airlift Missions (SAAMs) planned by the SAAM planning office, alternatively, correspond more to chartered air flights. In this case, custom itineraries are generated to satisfy the customer's movement requirements, and the customer's requirements constitute the sole focus of the mission. The missions associated with Presidential travel are SAAM missions. Other offices are concerned with generating missions that address other types of requirements, such as contingency operations, exercises, and training. Different types of missions create different types of resource requirements, but all planned missions specify an itinerary, a priority, a particular type of aircraft (technically referred to as the Model Development Series or MDS type) to be used, a preferred USAF unit (or *wing*) assignment and a time period, represented as a pair of dates, in which the mission should be executed.

All generated missions flow into the "Barrel Master" office (or Barrel for short) for allocation of necessary resources. The Barrel is the office within the TACC that has total visibility of AMC assets. Based on mission specific constraints and current resource availability, the Barrel determines which missions will be supportable and for those that are, which air wing(s) will be tasked to fly them. Within the Barrel Master office, the resource allocation and management problem is distributed among multiple Barrel Masters. Each Barrel manages the aircraft and crews of a particular set of air wings, determined by aircraft type and geographic location. An air *wing* consists of a set of aircraft (and corresponding crews) of the same type stationed at a particular USAF base. For example, McGuire Air Force base

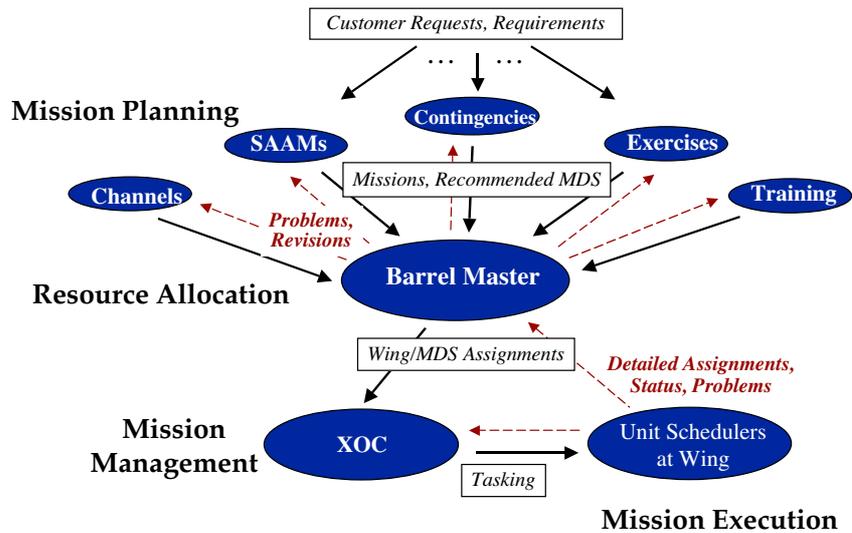


Figure 2.1: The AMC planning and execution environment

has a wing of C141s and a wing of KC135s. Currently, the “West Strat” Barrel is responsible for all west coast C141, C5 and C17 wings, the “East Strat” Barrel is responsible for all east coast C141, C5 and C17 wings, the “Tanker Barrel” is responsible for all tanker wings (i.e., KC135s and KC10s) and so on.

At present, all Barrels track available aircraft and aircrew capacity at various wings, and taskings of missions to specific wings are made on this notional basis. The individual air wing has responsibility for determining the actual aircraft tail and air crew(s) that will fly a given assigned mission.<sup>2</sup> The resource assignment for any planned mission is managed by the Barrel until the 24 hours before execution point, at which time the mission is “pushed” to the execution office (XOC) for plan review and execution management. As problems arise during the execution of missions, this information flows back to the Barrel, and may necessitate either reassignment of specific assets or revision to previously planned missions.

In allocating resources to a given mission, the Barrel must confirm that all constraints associated with its planned itinerary can be satisfied. A mission’s itinerary is the sequence of stops or airports the aircraft should visit during the execution of the mission. We refer to the

<sup>2</sup>Plans call for this assumption to change and for the Barrel to begin to track and allocate actual tails, as AMC’s “state of the world” initiative increases the visibility of execution data to planning processes.

flight between two successive stops as a *mission leg* (or *leg* for short). Each leg has an origin airport, the Point of Embarkation (POE), and a destination airport, the Point of Debarkation (POD). Each leg is followed (or preceded) by a certain ground time. During the period the aircraft is on the ground, a number of activities, or *ground events*, can occur: for example, loading and offloading of cargo, refueling, crew rest, crew change. The time period specified in the mission request should be at least as large as the time required by the aircraft to fly between all intermediate stops in the itinerary plus the required ground time at each airport. To feasibly support the mission, a sufficient number of aircraft and aircrews of the specified MDS type must be available during this entire period. The earliest date the mission can start is called the *Available to Load Date* (ALD) and the latest date the mission should finish is called the *Latest Arrival Date* (LAD). The length of this interval should be at least as large as the total duration of the mission.

Aircraft availability and aircrew availability are defined for each wing on a daily basis. Each wing has a total number of aircraft of a particular type and a corresponding total number of aircrews. Considering that some planes are undergoing maintenance and the wing also has some need for training and local missions, the wing will make a subset of its total aircraft and crews available for tasking to AMC missions. Each day, each wing will provide a certain number of *contract* aircraft and crews that can be allocated by the Barrel. The remaining wing assets, designated as *fenced* aircraft and aircrews, are reserved for local wing use and are beyond the jurisdiction of the Barrel.

The allocation problem just outlined is similar to the classically defined *Fleet Assignment Problem*: Given a schedule of flights defining the departure and arrival times for each flight leg, the *Fleet Assignment Problem* is the problem of deciding which flight equipment, or fleet, should be assigned to each flight segment [Clarke *et al.*, 1996, Rushmeier and Knottgiorgis, 1997]. For commercial airlines, the standard objective is to maximize revenues minus operating costs. The Barrel Master, alternatively, generally tries to maximize the total sum of priorities: s/he will try to assign the maximum number of high priority missions, and will generally only consider assigning lower priority missions after all higher priority ones have been assigned.

Most traditional solutions to the fleet assignment problem assign fleets to individual flight segments. The Barrel Master, alternatively, is concerned with assigning fleets to a sequence of segments or *strings*. A *string* is a sequence of connected flight segments that begins and ends at possibly different maintenance stations [Barnhart *et al.*, 1998]. By default, an AMC

mission itinerary is planned as a string that starts and ends at the same location (i.e., a round trip). Strings that start and end at the same station are usually referred as *aircraft rotations*. If possible, the Barrel would consider, and sometimes even prefer, using the same rotation for more than one mission. The difficulty in combining missions is in identifying and sorting out the opportunities for potential combinations among thousands of missions in the database.

The Barrel Master responsible for refueling assets, the Tanker Barrel, has a somewhat different allocation problem. In this case, the starting point is an *air refueling event*, which identifies one or more receiver missions, a cumulative fuel requirement, a refueling location (or track), and a rendezvous time. Considering the availability of tanker assets at various wings within range of the target refueling constraint, together with the fuel carrying capacities and burn rates of available aircraft types, one or more tanker missions are generated, sourced to a specific wing (or wings), and linked to the originating air refueling event.

In current practice, the allocation process carried out within the Barrel Master office is a mostly manual process. In her/his daily activities, each Barrel Master utilizes an electronic *commitment matrix*, which tracks available aircraft and aircrew capacity of different wings over time and records those missions already allocated. As new missions are received from various planning offices, the Barrel consults this matrix and tries to allocate resources which satisfy mission requirements. If all requirements can be satisfied, the wing assignment is made and this commitment is communicated back to the mission planner. In those cases where there are insufficient resources available to support a particular set of missions, the Barrel will consider more disruptive allocation alternatives. For example, s/he may consider using resources already allocated to lower priority missions, s/he may consider using resources provided by a wing other than the planner's preference, s/he may consider delaying the mission, and so on. Once one or more acceptable options are found, the Barrel communicates these possibilities back to the relevant planner and a solution that would best satisfy all sides is negotiated. A mission may also be determined to be unsupported, in which case this information is communicated back to the planner. In general, resource assignment is performed one mission at a time with little automation. Capabilities for generating more sophisticated allocation alternatives (e.g., involving mission combination) are quite limited, and generally such optimizations are not considered. In crisis situations where time pressure is greater, the lack of automated decision aids has increasing ramifications; high priority missions get accomplished, but at the expense of inordinate numbers of routine missions and at very high cost.

## 2.3 Incremental Constraint-based Search Models

The continuous, dynamic nature of the Barrel Master's allocation problem restricts the appropriateness of various automated planning and scheduling techniques. For example, classical fleet assignment solution procedures present several practical difficulties. First, they operate as black-box solution generators, and, as such, provide no ability to control solution change over time. Second, by virtue of their "re-solve from scratch" design, it can be difficult for these techniques to keep pace with execution events as tempo increases. In commercial airline settings, these shortcomings are not that important, since the set of routes to be flown tends to be rather stable (e.g., changing only month to month), and scheduling is more of a periodic activity aimed at establishing assignments for the next period. However, in the AMC environment, short notice (SAAM) missions mix with longer term (Channel) mission sets to create dynamically changing demand patterns over time. This requires more attention to advance commitment, and since new orders must be "re-cut" each time a mission is retasked there is pragmatic utility to minimizing disruption to scheduled missions when responding to new mission requests. Moreover, the higher time pressure associated with crisis situations puts increased emphasis on computationally efficient scheduling and allocation processes. In these contexts, automated techniques that integrate flexibly and gracefully with user decision processes is an important additional necessity.

Constraint-based search models provide a general approach to scheduling that is well matched to the requirements of the Barrel Allocator's allocation problem. Constraint-based scheduling models combine principles of constraint programming and heuristic search in the formulation and solution of scheduling problems (e.g., [Smith *et al.*, 1996, Cheng and Smith, 1997, Baptiste *et al.*, 2001]). Figure 2.2 shows the basic framework, which consists of three main components. In the center is an *active data base*, which contains a representation of the current solution. As scheduling decisions are posted to or retracted from this data base, a set of *constraint propagation* routines are triggered. Propagation computes the consequences of any change for related scheduling decisions, possibly winnowing (or enlarging) the set of feasible values for other decision variables or detecting a constraint conflict (which signifies an infeasible state). The other two principal components of a constraint-based scheduling model are commitment and retraction strategies/heuristics, which respectively guide the search process in moving forward (e.g., allocating resources or assigning start and end times to an as yet unassigned mission) or moving backward (e.g., unassigning a previously assigned mission) in the underlying search space. By configuring commitment and retraction

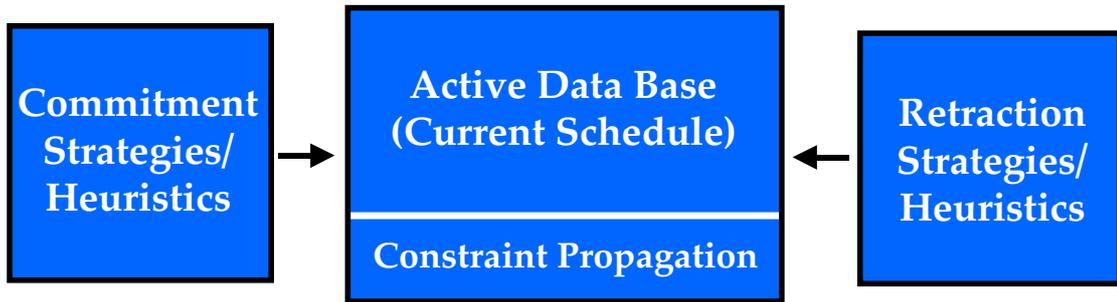


Figure 2.2: Constraint-Based Scheduling Models

heuristics with different search control mechanisms and search space assumptions, it is possible to define a range of both constructive (i.e., global search) and repair-based (i.e., local search) scheduling methods.

Constraint-based scheduling models are attractive in the current context for several basic reasons:

- *Incrementality* - By definition, constraint-based scheduling models are incremental decision-making procedures. As such, they provide a direct basis for managing solution change over time, and for minimizing undue disruption to the current schedule as new demands (missions) are accommodated.
- *Generality* - Constraint-based models have been shown to operate efficiently under quite general representational assumptions. Simple Temporal Problem (STP) constraint networks [Dechter *et al.*, 1991], for example, allow expression of a diverse range of relative and metric temporal constraints, while providing efficient incremental propagation algorithms. Similarly rich representations of resource constraints are also possible.
- *Configurability* - Constraint-based search procedures are also quite compositional by nature and these models tend to be easily reconfigured to accommodate new constraints and objectives [Smith *et al.*, 1996]. In the case of the AMC Allocator described below, this property is used to provide selective constraint relaxation capabilities in situations where all constraints cannot be simultaneously satisfied.

## 2.4 The AMC Allocator

As just suggested, an incremental, constraint-based scheduling model provides the core basis for the functionality provided by the AMC Allocator, a software tool designed to address the AMC Barrel Master allocation problem. Most basically, this model is used to define an efficient solution to a dynamic, string-based variant of the fleet assignment problem. More importantly from the standpoint of continuous operations, it is also used to define techniques for selectively re-optimizing to accommodate higher priority missions while minimizing disruption to most previously scheduled missions, and for selectively “merging” previously planned missions to minimize non-productive flying time. Finally, it is used to generate and compare alternative constraint relaxation options.

The AMC Allocator has been developed using the Ozone scheduling framework [Smith *et al.*, 1996, Becker, 1998], a shell for constructing constraint-based scheduling applications. Ozone consists of three major components: (1) a library of modeling primitives for constructing scheduling domain models, (2) an underlying constraint-based search architecture (an elaborated version of the basic model in Figure 2.2), and (3) a library of scheduling and rescheduling methods (encoding various commitment and retraction strategies/heuristics). Through instantiation of an *abstract domain model* with modeling primitives that match the characteristics of a given application domain, and selection of one or more basic scheduling methods, it is possible to quickly obtain an executable model, and allow development effort to quickly focus on the customizations of the model and the scheduling methods necessary to obtain high performance. The Ozone application framework consolidates the results of application building experiences in a wide range of complex scheduling domains.

The AMC Allocator itself currently provides two core sets of functionality to the AMC Barrel Master: *resource allocation* and *mission combination*. Separate modules are provided for airlift allocation and tanker allocation respectively, owing to differences in the nature of these two allocation problems as currently structured within the Barrel Master office. In all cases, functionality may be utilized in a more or less automated fashion, ranging from a fully manual mode where the system does little more than decision bookkeeping, to a semi-automatic mode, where the system generates alternative options and previews their impact, to a completely automatic mode, where the system determines the best decisions based on user-specified preferences. The system may also be invoked in an extended optimize mode, in which case the underlying incremental search process that is performed is broadened.

All of these capabilities are provided within a graphical, spreadsheet-like user interface, designed to emphasize the real-time responsiveness of the system’s scheduling and allocation procedures.

In the following subsections, we describe the representations and search procedures underlying the AMC Allocator in more detail. The configurability of these search procedures is considered in further depth in [Becker and Smith, 2000]. Other mixed-initiative aspects of the approach are emphasized in [Kramer and Smith, 2002].

### 2.4.1 Representational Assumptions

Within the AMC Allocator, missions are represented as hierarchical task (or activity) networks. A mission expands into a network of more detailed flight segments and ground activities. Flight segments correspond to individual legs of the mission’s itinerary; ground activities model such activities as crew rest periods and aircraft preparation tasks. Each constituent activity has a *start time* and an *end time*, and is linked via precedence constraints to other mission activities. The duration of an activity is a function of its type. For example, the duration of a flight activity depends on resource speed and the distance traveled from its *origin* to its *destination*.<sup>3</sup> For many ground events, durations are specified as fixed constants. The duration of an aggregate activity (e.g., a mission) is the duration of its sub-network.

A mission specifies requirements for resource capacity for its entire duration. In the case of airlift missions, some number of aircraft and aircrews of a particular type are specified. In the case of air refueling (or tanker) missions, there is often flexibility with respect to type of refueling aircraft, and the required numbers of aircraft and aircrews are derived from fuel offload requirements and fuel carrying capacity limits. A mission also has associated temporal restrictions on when it can execute. In the case of an airlift mission, execution is constrained by its “Available to Load date” (ALD) and its “Latest Arrival Date” (LAD). Tanker missions, alternatively, must synchronize to arrive at the refueling point at the designated “Air Refueling Control Time” (ARCT).

Aircraft and aircrews are modeled at an aggregate wing level. A wing designates a co-located set of aircraft and aircrews of a particular type. It specifies a particular geographic *location*, a pool (or fleet) of aircraft and a pool of aircrews. A given pool of aircraft or aircrews has a *total capacity*, with each unit of capacity representing one aircraft or aircrew.

---

<sup>3</sup>Currently, distances are computed based on great circle route.

The total capacity of a resource pool is partitioned into two subsets: *contract* capacity, representing the capacity that has been budgeted to AMC and is available for allocation, and *fenced* capacity, representing the aircraft that have been held back for the wing's local use. The contract and fenced capacity of a wing can vary over time, and resource availability will become further restricted as capacity is assigned to specific missions over time. The available capacity of a resource pool is thus represented as a sequence of *capacity intervals*, each recording the total, contract, fenced and available capacity over the interval spanned by its start and end time. As missions are assigned to wings (or de-assigned) during the allocation process, the affected portion (along the time line) of their available capacity representations is updated to reflect the reserving (or freeing up) of required aircraft and aircrew capacity. Thus resource capacity is represented and allocated over continuous time, at some level of temporal granularity.

The output of the AMC Allocator (i.e., the current schedule) is a set of assignments of the form (*Mission, Wing, start-time, end-time*), indicating that *Wing* will perform *Mission* over the interval from *start-time* to *end-time*. To be feasible, such an assignment must satisfy the following constraints:

- **Wing capacity constraints** - The cumulative resource requirements of all missions assigned to *Wing* over the interval from *start-time* to *end-time* must not exceed the number of contract aircraft or aircrews available at *Wing* for any  $t$ ,  $start-time \leq t \leq end-time$ .
- **Resource ground time constraints** - The assigned interval from *start-time* to *end-time* must account for aircraft onload, offload and minimum time-on-ground requirements. Each of these constraints is specified as a function of aircraft type.
- **Flight duration constraints** - The assigned interval from *start-time* to *end-time* must account for cumulative flight time requirements, including necessary positioning and depositioning flight segments.
- **Aircraft range constraints** - Any given flight segment inserted into *Mission's* itinerary as a consequence of assigning *Wing* must cover a distance less than the flying range of the assigned aircraft type. In general, the creation of positioning flights, depositioning flights and/or bridging legs (from offload of one mission to onload of the next) may be implied by a given wing assignment.

- **Crew duty day constraints** - Depending on positioning/depositioning considerations and the crew type required by *Mission* - basic or augmented - it may be necessary to insert crew rest periods at appropriate intermediate points of the mission to enforce crew duty day restrictions. The assigned interval from *start-time* to *end-time* must account for this additional constraint as well.
- **Mission timing constraints** - Finally, *start-time* and *end-time* must be consistent with all timing constraints on the execution of *Mission*. In the case of an airlift mission, the start of the first cargo carrying flight segment must be  $\geq$  *Mission's* ALD, and the end of the last cargo carrying leg must be  $\leq$  *Mission's* LAD. For tanker missions, the end time of the positioning flight must = *Mission's* ARCT.

Given the oversubscribed nature of the Barrel Master allocation problem, it is not always possible to satisfy mission time constraints with existing available lift capacity, and those missions that cannot be feasibly scheduled at any point are designated as unassignable. In such situations, the user may choose to analyze and compare options that involve relaxation of various constraints. Currently, the following relaxed formulations may be solved to generate such options:

- **Priority-based pre-emption** - Every mission has a pre-defined priority that establishes its intrinsic importance. Under a priority-based pre-emption formulation, an unassignable mission may obtain its required resource capacity by pre-empting (or bumping) one or more previously assigned, lower-priority missions. If this occurs, the set of pre-empted missions is then rescheduled in succession and may, in turn, find alternative resource assignments at the expense of still lower priority missions. At quiescence, any remaining unassigned missions become unsupported and are added to the current set of unassignable missions. When operating under this formulation, additional constraints can be added by the user to exert greater control of the extent of solution change. A *mission locking* mechanism allows any specific mission assignment to be designated as unalterable. A *freeze interval*, specifying a period of time in advance of execution within which pre-emption is not allowed, can also be imposed.
- **Resource Over-allocation** - Under this formulation, the aircraft and aircrew contract capacity constraints are considered relaxable. Since the numbers of contract aircraft and aircrews are typically smaller than the total number of assets possessed by the

wing, the user may choose to go over the published contract levels of a given wing. This happens with a fair amount of frequency. It generally reflects private knowledge that the Barrel Master may have about wing assets or agreement on the part of the wing to use fenced aircraft.

- **Mission Delay** - Under this formulation, the mission's LAD constraint (in the case of airlift) or ARCT constraint (in the case of refueling) is considered relaxable. The user may consider the option of delaying the current mission until necessary resources are available. If delay seems like a potentially viable alternative then this information can be suggested to the mission planner.
- **Alternative MDS** - Under this formulation, the requirement that the mission use the airframe (or MDS) type designed by the mission planner is relaxed, and alternative aircraft types can be considered as a means of accommodating the mission. Due to differences in carrying capacities of different aircraft, the numbers of aircraft and crews required to support a mission may vary across aircraft type as well. As with mission delay, potentially viable options can be communicated back to the mission planner as suggestions.
- **Composite mission constraint relaxations** - Formulations that permit simultaneous relaxation along multiple dimensions (e.g., Mission Delay and Priority-based Pre-emption, Mission Delay and Over-Allocation) are also possible.
- **Mission Combination** - By default, missions are flown as round trips from a particular home base. If the origin and/or destination do not coincide with this base, then positioning and/or de-positioning flight segments are added into the mission itinerary. In mission combination mode, the requirement that each mission be flown as a round trip is relaxed, and opportunities to exploit non-productive flying time by "recycling" an aircraft directly from the destination (offload) of one mission to the origin (onload) of the next are sought. Mission combination can provide a direct option for supporting an otherwise unassignable mission. It can also provide a basis for compressing the resource requirements of a set of previously assigned missions and reclaiming resource capacity for other use.

Below, we outline solution procedures for these various problem formulations.

## 2.4.2 Allocation of Airlift Assets

The central function supported by the Allocator is that of assigning planned missions to wings over time. In typical mode of operation, there is an existing set of assignments (i.e., the current schedule) in place, and the problem is one of determining assignments for sets of newly planned missions in the presence of this backdrop. Figure 2.3 graphically depicts the basic search procedure used to make the wing/time assignment for a single mission.

The search procedure proceeds in 3 basic steps:

1. a set of candidate resources (wings) is generated,
2. for each candidate wing, a set of possible allocation intervals is generated, and
3. each  $\langle \text{wing}, \text{allocation interval} \rangle$  pair is evaluated and the highest ranked candidate is selected.

As implied by Figure 2.3, a full instantiation of the `AssignMission` search procedure is obtained by specifying three components: a search operator for generating candidate resources (referred to generically as  $Gen_{Resources}$ ), a search operator for generating candidate allocation intervals (generically called  $Gen_{Intervals}$ ), and an evaluation metric ( $Eval_{Criterion}$ ) for ranking alternatives. By parameterizing the procedure to operate with different sets of operators and evaluation criteria, resource assignments can be generated and evaluated under a range of different constraint relaxation assumptions.

In the most basic case, `AssignMission` is configured to search only for feasible assignments, i.e.,  $\langle \text{wing}, \text{allocation interval} \rangle$  pairs that are consistent with the time and resource requirements specified by the mission and are also compatible with the assignments of previously scheduled missions. This *feasible* configuration of `AssignMission` is obtained by incorporation of the triple  $\langle Gen_{RequestedRes}, Gen_{FeasibleInts}, Eval_{MinFlyingTime} \rangle$ . Here,  $Gen_{RequestedRes}$  generates candidate wings consistent with aircraft type requested by the mission. Likewise,  $Gen_{FeasibleInts}$  scans a candidate wing's aircraft and aircrew capacity profiles for allocation intervals (1) with sufficient amounts of available capacity to support mission requirements, (2) with a duration greater than or equal to the time required to accomplish the mission (a function of mission itinerary, aircraft speed, wing's home base location, crew rest requirements, and other constraints on duration identified in Section 2.4.1), and

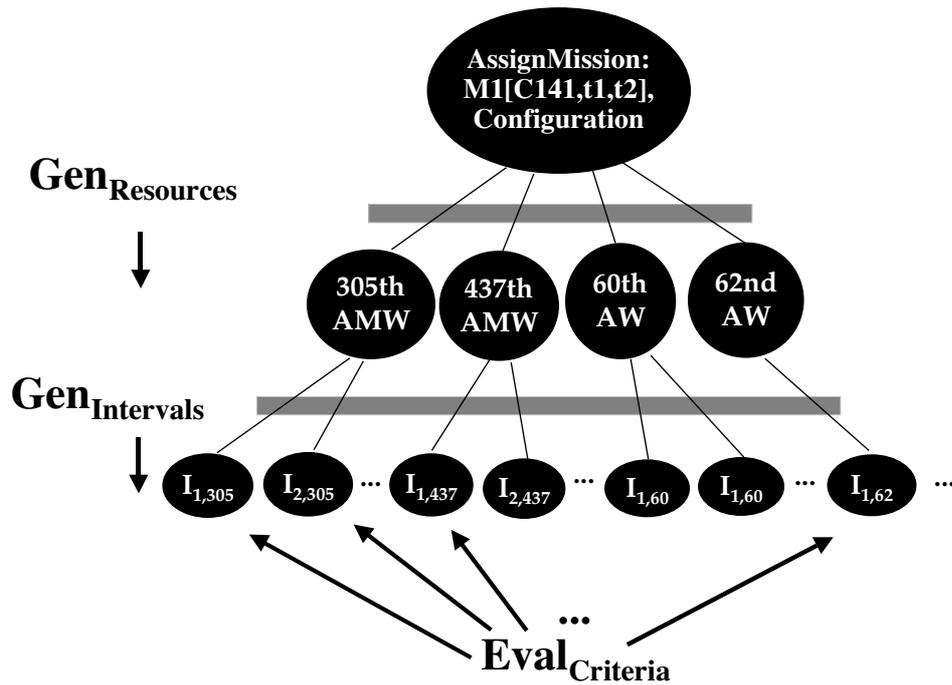


Figure 2.3: Basic search procedure for resource allocation. This example assumes that mission  $M1$  requires  $C141$  aircraft with earliest pickup at  $t1$  and latest delivery by  $t2$ , and that  $C141$ s can be provided by either the 305th Air Mobility Wing (AMW), the 437th AMW, the 60th Air Wing (AW) or the 62nd AW.

(3) with start and end times that satisfies the mission’s earliest on-load time and latest off-load time constraints. Candidates are differentiated on the basis of total flying time and the candidate assignment that minimizes this metric is selected.

By selectively substituting different search operators and/or evaluation criteria, `AssignMission` can alternatively be used to find assignments under various relaxed problem assumptions identified in Section 2.4.1. For some types of constraints, relaxation simply implies the consideration of a different discrete set of options. For example, substitution of  $Gen_{AlternativeRes}$  for  $Gen_{RequestedRes}$  results in generation of assignments that consider types of aircraft other than the type requested by the mission planner. For other classes of constraints, however, relaxation is more continuous in nature, and in substituting a search operator that assumes constraints can be relaxed, the search must also be biased to promote their satisfaction to the extent possible.

By varying the operator used to generate allocation intervals and the evaluation metric used to prioritize candidate solutions, a number of useful `AssignMission` configurations are defined:

- **Delay** - Incorporation of the triple  $\langle Gen_{RequestedRes}, Gen_{DelayInts}, Eval_{MinTardiness} \rangle$  yields an assignment procedure which assumes that mission deadlines can be relaxed if necessary.  $Gen_{DelayInts}$  uses the same mechanism used by  $Gen_{FeasibleInts}$  but considers a larger portion of the candidate wing’s aircraft and crew capacity profiles, and  $Eval_{MinTardiness}$  ensures that the mission deadline will be relaxed to the minimum extent possible

In this configuration and others, advantage can be taken of the relationship between search operator and evaluation criterion to effectively constrain the number of candidate solutions generated. For example, by scanning forward in time through the capacity profiles of required resources, the first interval with available capacity found will be the one that minimizes delay for that resource. If this approach is taken only one interval need be generated for each candidate resource. In other configurations (e.g., the *pre-emption* case below), where there is no such dominance condition for constrained solution generation, more ad-hoc heuristic cutoffs can be used.

- **Over-allocation** - The triple  $\langle Gen_{RequestedRes}, Gen_{OverInts}, Eval_{MinOverUsage} \rangle$  defines an assignment procedure where capacity constraints are relaxable.  $Gen_{OverInts}$  scans the capacity profile of a candidate wing, but generates allocation intervals that extend above the wing’s “contracted” level (i.e., dipping into its locally reserved or “fenced” pool of aircraft capacity).  $Eval_{MinOverUsage}$  promotes selection of the generated allocation interval that minimizes the level of over-allocation.

In this case, maximal intervals at different levels of over-allocation can be efficiently generated via linear scans of the capacity profiles of the two required resources, and subsequently pruned to minimize the temporal extent of over-allocation.

- **Priority-based Pre-emption** - A configuration which assumes that some number of previously made assignments can be relaxed (or disrupted) is defined by the triple  $\langle Gen_{RequestedRes}, Gen_{BumpInts}, Eval_{MinDisruption} \rangle$ . This configuration implements a form of pre-emption, based on mission priority. In scanning a candidate wing’s capacity profile,  $Gen_{BumpInts}$  considers capacity currently allocated to lower priority

missions as available for assignment, and generates allocation intervals based on this assumption. *Eval<sub>MinDisruption</sub>* promotes allocation intervals that disrupt the fewest missions and those with the lowest priority. This minimizes the cascading effect (since any mission that is pre-empted by a higher priority mission is recursively re-scheduled using the same procedure).

Given the combinatorial number of allocation intervals that can be generated via a complete capacity profile scanning procedure ( $O(c^f)$  where  $c$  is the capacity of the resource and  $f$  is the duration of capacity profile fragments), our current implementation utilizes a linear, heuristic sampling strategy compatible with *Eval<sub>MinDisruption</sub>*. Briefly, allocation intervals are generated by single forward scan through the required resources' capacity profiles over the time interval where capacity is required. At each time point encountered during the scan, the set of pre-emptable missions is computed. If non-empty, the current allocation interval is extended by (1) computing the subset of pre-emptable missions of lowest priority and (2) selecting the mission in this subset that maximally extends the current interval. If there are no pre-emptable missions, the current allocation interval is ended (and retained if of sufficient duration), and the scan continues at the start point of the next pre-emptable mission.<sup>4</sup>

- **Composite Relaxations** - Components of the above base configurations can also be composed to define configurations of `AssignMission` where multiple constraints are simultaneously relaxed.

`AssignMission` (in any of the configurations described above) can be applied to any selected set of missions via the `AssignMissions` procedure given in Figure 2.4. Within this Constraint Satisfaction Problem Solving (CSP) style search procedure, the *sort-order* parameter determines the overall variable-ordering strategy (i.e., the order in which missions are selected for assignment). In the current implementation, mission priority (first) and latest delivery date (second) are used as a heuristic basis for prioritizing unassigned missions. Input missions are sorted on this basis (within `SelectMission`) and `AssignMission` is then sequentially applied to each to allocate required resources (using whichever of the above configurations has been designated).

---

<sup>4</sup>An alternative heuristic strategy, which attempts to find pre-emptable missions that best match target “resource area” requirements, is described in [Zhou and Smith, 2002]. Though somewhat more costly computationally, this approach has been found to be considerably less disruptive than the above strategy, and we plan to incorporate it as an option in a future Allocator release.

```

-----
AssignMissions(UnassignedMissions, configuration, sort-order)
  While UnassignedMissions  $\neq \emptyset$  Do
     $M \leftarrow \text{SelectMission}(\text{UnassignedMissions}, \text{sort-order})$ 
     $\text{UnassignedMissions} \leftarrow \text{UnassignedMissions} - \{M\}$ 
    If AssignMission( $M$ , configuration)
      Then  $\text{Status}_M \leftarrow \text{assigned}$ 
      Else  $\text{Status}_M \leftarrow \text{unassignable}$ 
    EndWhile
End
-----

```

Figure 2.4: Overall Mission Scheduling Procedure

In cases where a given mission has no feasible assignments (i.e., `AssignMission(M, feasible)` is applied and returns no solution), an exploration of possible relaxation options can be conducted through repeated application of `AssignMission` in different configurations. In the current implementation, this procedure is used in what-if mode to generate alternative options, and the user performs the evaluation and selection. Figure 2.5 shows a sample output as displayed within the Allocator’s “Compare Options” interface.

### 2.4.3 Allocation of Air Refueling Missions

The process of allocating wings to refueling or “tanker” missions is similar to that used for assignment of airlift missions, but requires an important extension. Tanker missions are generated and sourced to satisfy an *air refueling (AR) request*, which minimally calls for a specified amount of fuel to be delivered to a particular rendezvous point at a particular time. The number of tanker missions required to satisfy a given AR request will be a function of the fuel required and the fuel capacity of aircraft allocated, and hence may not be specified in advance. The AR request may also specify a particular aircraft type and possibly a preferred wing assignment, which is referred to as a specific tanker request.

Given these distinctions, the tanker allocation process must incorporate an additional mission planning function. The extended `AssignRequest` procedure is given in Figure 2.6,

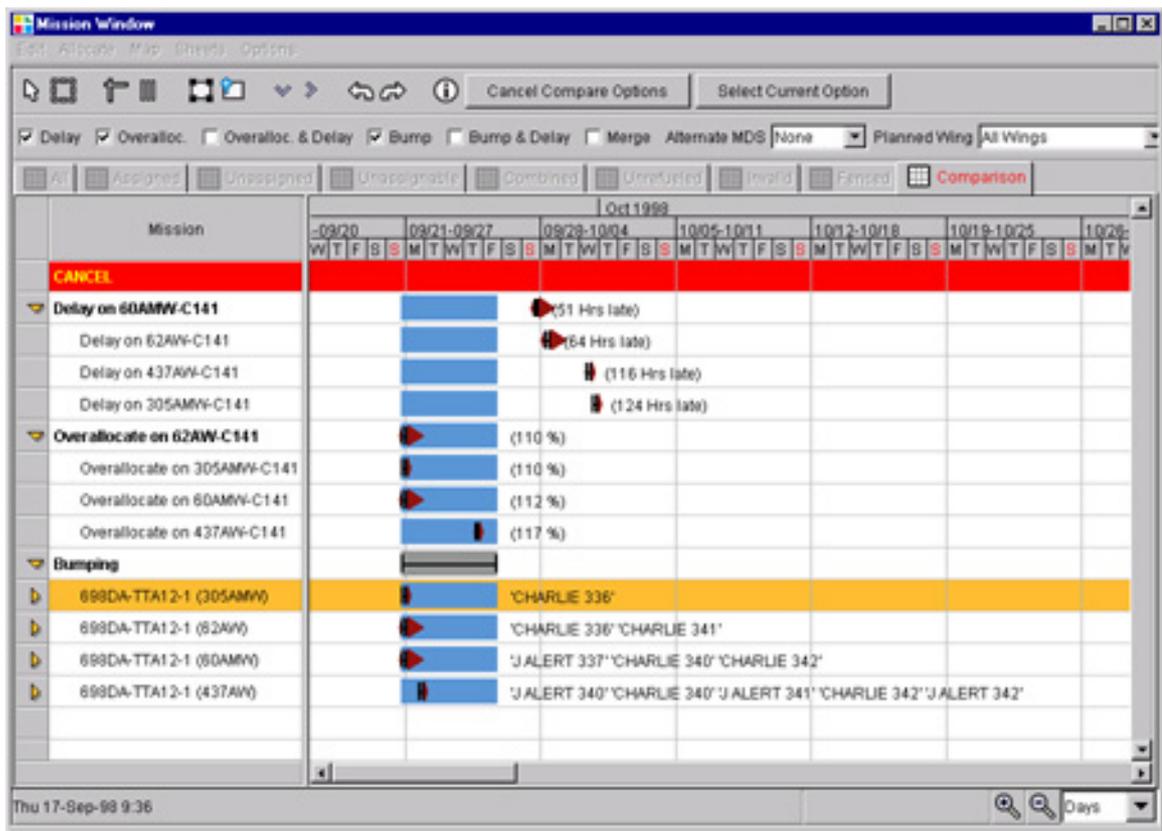


Figure 2.5: Generating Options for an Unassignable Mission

---

```

AssignRequest(AR, configuration)
  While RequiredFuelAR > 0 Do
    M ← GenerateNewTankerMission(AR)
    If AssignMission(M, configuration)
      Then RequiredFuelAR ← RequiredFuelAR − FuelCapacityM
      Else StatusAR ← unassignable; Return
    EndWhile
  StatusAR ← assigned
End

```

---

Figure 2.6: Air Refueling AssignRequest Procedure

In this procedure a mission generation and assignment loop is iterated until either the fuel requirement has been satisfied or it is determined that the AR request is unsupported. As indicated, the full range of AssignMission configurations available for airlift allocation are equally applicable in the air refueling allocation context.

#### 2.4.4 Mission Combination

A second core function provided by the Barrel Allocator is mission combination. This capability is defined by coupling the basic AssignMission procedure with a procedure for generating possible mission pairings.

More specifically, the process of merging a given mission  $M$  with another existing mission proceeds in three basic steps:

1. Generate a set of possible composite missions - In this step, all possible composite missions involving  $M$  are generated. For any pair of missions  $(M_1, M_2)$ , there are two potential pairings: one in which mission  $M_2$  flies after the end of mission  $M_1$ 's last leg; and one in which mission  $M_1$  flies after mission  $M_2$ . A possible combination is one which (1) satisfies the conjunction of constraints established by spatial and temporal restrictions, and (2) establishes user-imposed limits on acceptable candidates. The user may specify three additional constraints on acceptance:

- a minimal percentage reduction in total duration of the combined mission,
- a maximum layover time between missions, and
- a maximum distance between the destination of the first mission and the origin of the second.

The output of this step can be seen as the set of all notional composite missions  $\overline{M}$  resulting from concatenation of possible combination pairs  $(M_1, M_2)$ .

2. Generate set of feasible composite missions - In this step `AssignMission` is applied to each candidate  $\overline{M}$  identified in step 1 to determine the set of composite missions that are resource feasible (i.e., there is available resource capacity to support the composite mission over its entire duration). Although currently only *feasible* allocations are considered, any of the relaxed `AssignMission` configurations could also be used.
3. Selection of best candidate - Once all feasible combinations have been determined, the candidate providing the largest overall reduction in airplane flying time is selected.

Figure 2.7 summarizes the mission combination procedure. Note that although `CombineMission` performs only pairwise merging of missions, the combination of larger numbers of missions can be accomplished by recursively applying the algorithm to composite missions.

## 2.4.5 Incremental Optimization

The above described procedures are designed to flexibly support a continuous, interactive resource management process. New missions are integrated incrementally as problem constraints permit, and in circumstances where missions cannot be feasibly accommodated, a range of constraint relaxation and solution change options can be explored to find acceptable compromises and increase the number of supportable missions. In operation, the system provides an efficient, mixed-initiative decision-making environment – a 2 week interval of missions extracted from the current AMC data base (approximately 1000 missions, 5000 flights) is scheduled from scratch in less than 5 seconds on a Pentium 4 1GHz machine. More typical incremental planning and scheduling actions are executed in real-time.

---

```

CombineMission( $M, MaxLayover, MaxDistance, ReqReduction$ )
   $PossibleCombinations \leftarrow \emptyset$ 
  For each mission pair  $(M_1, M_2) : (M_1 \vee M_2 = M) \wedge (M_1 \neq M_2)$  Do
     $\overline{M} \leftarrow \text{GenerateCompositeMission}(M_1, M_2)$  ;  $\overline{M} = M_1 + M_{1,2} + M_2$ 
    If  $\text{ConstraintsSatisfied}(\overline{M}, MaxLayover, MaxDistance, ReqReduction)$ 
      Then  $PossibleCombinations \leftarrow PossibleCombinations \cup \{\overline{M}\}$ 
    EndFor
   $Candidates \leftarrow \emptyset$ 
  For each  $\overline{M} \in PossibleCombinations$  Do
    If  $\text{AssignMission}(\overline{M}, feasible)$  Then  $Candidates \leftarrow Candidates \cup \{\overline{M}\}$ 
  EndFor
   $SelectCandidate(Candidates, Eval_{MaxTripRed})$ 
End

```

---

Figure 2.7: Mission Combination Procedure

This efficiency is gained principally through reliance on search heuristics, and within the core Allocator procedures, automated solution generation is strongly biased by mission priority. This coincides well with AMC business practice, but can also sometimes result in missed opportunities to support additional lower priority missions.

To compensate for such circumstances, we have been exploring the development and use of incremental optimization procedures, which can be applied at extended computational cost when decision time constraints permit. One simple example provided within the current Allocator release is a “resource usage compression” capability, which applies the basic mission combination procedure discussed above to the entire set of missions allocated to a wing (or set of wings) over some interval, to identify opportunities for more efficient recycling of assets and to reclaim resource capacity for additional tasking.

Another incremental optimization procedure we have recently developed attempts to incorporate additional missions by temporarily de-emphasizing mission priority as a pre-emption criterion and exploring other feasible <resource, allocation interval> assignments for selected higher priority missions [Kramer and Smith, 2003]. One consequence of the priority-based allocation bias of the core `AssignMissions` procedure is that some (higher priority) missions with greater scheduling flexibility may be assigned before other (lower priority) missions with less scheduling flexibility; and this possibility can lead to sub-optimal greedy commitments that leave lower priority missions unassignable. The `MissionSwap` procedure (see Figure 2.8) is designed to provide a framework for locally rearranging assignments to take better advantage of the collective scheduling flexibility associated with a given set of competing missions. It proceeds by retracting some number of currently assigned missions, which frees up capacity sufficient to enable insertion of the (previously unassignable) input mission. `MissionSwap` is then recursively called for each retracted missions that cannot be immediately assigned elsewhere (by a subsequent call to `AssignMission`), with the additional constraint that once a mission has been reassigned via a swap move, it is “protected” from further retraction.<sup>5</sup> In our current implementation, the results of `MissionSwap` are incorporated conditionally; If the procedure bottoms out with feasible reassignments for all retracted missions, then the results are accepted as a new improved solution. Alternatively, if all retracted missions cannot be feasibly reassigned, then the solution is restored to

---

<sup>5</sup>It should also be noted that since this and other incremental optimization procedures utilize the same underlying search infra-structure, additional constraints can always be introduced to prohibit the system from changing specific allocation decisions if greater control over solution change is required.

---

```

MissionSwap( $M, ProtectedMissions$ )
     $ConflictSet \leftarrow ComputeCapacityConflicts(M)$ 
     $Retracted \leftarrow RetractMissions(ConflictSet, ProtectedMissions)$ 
    if  $Retracted = \emptyset$  then Return( $\emptyset$ ) ; no possibility to assign  $M$ 
     $ProtectedMissions \leftarrow ProtectedMissions \cup \{M\}$ 
    AssignMission( $M, feasible$ )
    AssignMissions( $Retracted, feasible, least-flexible-first$ )
    For each ( $i \in Retracted \wedge status_i = unassignable$ ) do
         $ProtectedMissions \leftarrow MissionSwap(i, Protected)$ 
        if  $ProtectedMissions = \emptyset$  then Return( $\emptyset$ ) ; unable to reassign  $i$ 
    EndFor
    Return( $ProtectedMissions$ ) ; success
End

```

---

Figure 2.8: Mission Swap Procedure

its prior state and the original input mission is returned to “unassignable” status”.

Central to the effectiveness of this swapping procedure is the heuristic employed to select which mission or missions to retract in a given conflict situation. Conceptually, one would like to retract the mission (or missions) that possess the greatest potential for reassignment. One simple estimate of this potential is the scheduling flexibility provided by a mission’s feasible execution window. More precisely, we define the flexibility of a given mission  $m$  as

$$Flex_m = \frac{\sum_{r \in R_m} alloc-dur_{r,m}}{(lft_m - est_m) \times |R_m|}$$

where  $R_m$  is the set of alternative resources (i.e., wings) that could support  $m$ ,  $alloc-dur_{r,m}$  is the duration that resource  $r$  would require to perform  $m$ ,  $est_m$  is the earliest start time of  $m$ , and  $lft_m$  is the latest finish time of  $m$ . This measure of scheduling flexibility gives rise to the following retraction heuristic:

$$MaxFlex = i \in C : Flex_i \leq Flex_j \forall j \neq i$$

where  $C$  is a set of conflicting missions in the  $ConflictSet_u$  for some unassignable task  $u$ . This  $MaxFlex$  heuristic drives the **RetractMissions** step of **MissionSwap**.

Preliminary experimental analysis of the performance of the `MissionSwap` procedure on representative AMC mission data at different levels of resource contention has indicated an ability to substantially improve the airlift schedules initially produced by `AssignMissions` in certain circumstances. As well, the *MaxFlex* heuristic has been shown to consistently outperform several, more complex “contention-based” retraction heuristics at a fraction of the computational cost. The reader is referred to [Kramer and Smith, 2003] for a complete description of this incremental optimization procedure and for full details of these initial performance results. Our current work is investigating the performance/cost tradeoffs associated with variants of `MissionSwap` that carry out additional search when initial retraction choices don’t pan out. We anticipate full incorporation of this capability in a future Allocator release.

A final longer-term approach to incremental optimization that we have been exploring involves the use of randomization to boost the performance of search heuristics [Cesta *et al.*, 2002, Cicirello and Smith, 2002]. This work assumes an iterative sampling approach, where solutions (schedules) are generated repeatedly through non-deterministic application of base search heuristics. The key idea is to bias the degree of randomness to the level of discrimination provided by the heuristic in different decision contexts; when the heuristic clearly favors one choice over others, choose more deterministically, and vice versa when the heuristic is less informing choose more randomly. These techniques have yielded strong performance on classical scheduling problems but have not yet been extended to deal with the full range of constraints in the Barrel Master allocation problem.

## **2.5 Project History and Status**

The AMC Allocator was developed initially under sponsorship of the Joint Department of Defense Advanced Research Projects Agency/USAF Research Laboratory (DARPA/AFRL) Planning Initiative Program. Following early success within this core technology development program in the area of constraint-based planning and scheduling, we were put in contact with AMC in hopes of identifying potential avenues for future technology transition. The AMC Barrel Master allocation problem was selected as an initial application focus, and several successively more sophisticated Allocator prototypes were developed over the following two years. Following positive evaluation by AMC personnel in 1999, a decision was made to transition the Allocator technology into operations within the TACC. In April

2000, project sponsorship shifted from DARPA/AFRL to AMC, with research and development work continuing under subcontract to Northrop Grumman Information Technology (NGIT), the prime contractor of AMC's operational Consolidated Mobility Planning System (CAMPS). This collaboration with NGIT is ongoing.

A version of the Allocator supporting airlift allocation was first incorporated as a module of the AMC's Consolidated Air Mobility Planning System (CAMPS) for user testing and experimental evaluation in September 2000. In this preliminary insertion, where the goal was to gain experience and build user acceptance, provisions were made to shadow the operational CAMPS business process rather than directly manipulate the data in the CAMPS data base. In retrospect, this decision turned out to be ill-advised, since users essentially had to do "double work" to make use of the new technology. Nonetheless, considerable insight was gained regarding how to better insert the Allocator technology into current CAMPS business processes. Following this feedback, and a programmatic decision to refocus on the Tanker Barrel's problem and processes, an enhanced AR (Air Refueling) Allocator module has been developed and integrated into CAMPS. In May 2003, this version of the Allocator is scheduled for initial operational release as part of CAMPS Release 5.4. At this point, Tanker Barrels will commence training for actual operational use.

## **2.6 Summary and Current Directions**

In this paper we have described the incremental, constraint-based scheduling model that underlies the design of the AMC Allocator. The model is particularly well suited to the continuous nature of the AMC Barrel Master allocation problem, where newly planned missions must be integrated into an evolving global schedule on an ongoing basis. In this environment, it is important to manage solution change, as it can be unproductive and costly to continually redirect various executing agents. The incremental scheduling and allocation procedures provided by the Allocator directly address this issue. When possible, they allow new missions to be incorporated without change to previous allocation decisions. In situations where non-disruptive allocation is not possible, the procedures support generation of multiple change options and promote selective, controlled manipulation of previous allocation decisions. These capabilities are provided in a real-time, interactive decision-support framework.

Our current research aims at broadening the scope and applicability of the Allocator func-

tionality. One major thrust aims at integration with “state of the world” information updates. Although current application of the technology within the Barrel Master office aims primarily at supporting the AMC mission planning process in advance of execution, the incremental scheduling and allocation capabilities provided by the Allocator are equally applicable in the context of execution management, where unexpected events (e.g., aircraft failures, base closures) often necessitate reassessment of current plans. Furthermore, there is increasing momentum within AMC and the TACC toward tighter integration and information flow between planning and execution offices, which can be expected to provide the basis for more execution-driven planning and allocation processes. Our work here is focusing on (1) techniques for reconciling actual execution status with expectations contained in the current schedule and recognizing the need for change, (2) tools for visualizing current replanning problems and opportunities, and (3) strategies and heuristics for responding to unexpected execution events.

A second thrust of our current research is on expanding the core search procedures to incorporate a larger and more diverse set of mission planning and scheduling constraints. In some cases, these model extensions correspond directly to the additional types of constraints that come into play in closing the loop with execution. Others relate to a desire to explore the applicability our current incremental constraint-based search approach to upstream mission planning problems.

## **2.7 Acknowledgements**

Many individuals have contributed to the development and deployment of the AMC Allocator software system. The authors would like to thank Susan Buchman, Mark Burstein, Charlie Collins, David Crimm, Chuck Dearth, Brian Gloyer, David Hildum, Ora Lassila, Dirk Lemmermann, Gary Pelton, Lindy Resnor, Mark Shieh, Seppo Torma, Rod Thornton, Crystal Whittenburg, and Joe Zhou. The research reported in this article was sponsored in part by the Department of Defense Advanced Research Projects Agency and the US Air Force Research Laboratory under contracts F30602-97-2-0227 and F30602-96-D-0058, by the USAF Air Mobility Command under subcontract 10382000 to Northrop-Grumman Information Technology, and by the Robotics Institute at Carnegie Mellon University.

## Chapter 3

# An Ontology for Constructing Scheduling Systems

**Summary:** In this chapter, we consider the use of ontologies as a basis for structuring and simplifying the process of constructing domain-specific problem-solving tools. We focus specifically on the task of scheduling. Though there is commonality in scheduling system requirements and design at several levels across application domains, different scheduling environments invariably present different challenges (e.g., different dominating constraints, different objectives, different domain structure, different sources of uncertainty, etc.), and hence we can expect high-performance application systems to require customized solutions. Unfortunately, the time and cost associated with such domain-specific system development at present is typically quite large.

Our work toward overcoming this application construction bottleneck has led to the development of OZONE, a toolkit for configuring constraint-based scheduling systems. A central component of OZONE is its scheduling ontology, which defines a reusable and extensible base of concepts for describing and representing scheduling problems, domains and constraints. The OZONE ontology provides a framework for analyzing the information requirements of a given target domain, and a structural foundation for constructing an appropriate domain model. Through direct association of software component capabilities with concepts in the ontology, the ontology promotes rapid configuration of executable systems and allows concentration of modeling effort on those idiosyncratic aspects of the target domain. The OZONE ontology and toolkit represent a synthesis of extensive prior work in developing constraint-based scheduling models for a range of applications in manufacturing, space and transportation logistics.

We first motivate the use of ontologies as model building tools, establishing linkages to recent concepts in software engineering and proposing an extended view of ontologies that includes capability descriptions. We then describe our perspective on the structure of planning and scheduling domain models and summarize major components of current OZONE scheduling ontology.<sup>1</sup>

---

<sup>1</sup>An earlier version of this chapter appeared as Smith, Stephen F. and Becker, Marcel A., “An ontology for

### 3.1 Ontologies and Model Building

In recent years, the field of software engineering has placed increasing emphasis on software reusability as a key to reducing the time and cost of application system construction and maintenance. Techniques for development and (re)use of software components have received wide attention and use [Biggerstaff and Perlis, 1989, Krueger, 1992], and tools that support system development from reusable building blocks are maturing [Cotter, 1996, Smith, 1990, Batory and O'Malley, 1992]. Despite this activity, however, the systematic development of applications from components remains an open issue. One obstacle stems from the lack of communication and coordination between component developers (who must *design for reuse*) and component users (who *design with reuse*) [Becker and Díaz-Herrera, 1994]; overly complex components are difficult to reuse while overly simple components do not provide sufficient building blocks. Two current areas of software engineering research aimed specifically at promoting reusability are (1) domain analysis [Hess *et al.*, 1990, Arango and Prieto-Díaz, 1991] and (2) software architectures [Garlan and Shaw, 1994, Clements, 1996]. Methodologies for domain analysis center around formulation of a *domain model*, which is intended to precisely delineate the scope of an application domain, the objects in this domain, desired system functionalities and features, and the dimensions along which these functionalities vary. Research in software architecture has focused on categorizing reusable architectural styles [Garlan *et al.*, 1994], on architectural description languages [Shaw and Garlan, 1994], and on architectural patterns [Gamma *et al.*, 1994] that support composition of components. What is missing are mechanisms to support the transition from (abstract) domain models to specific architectural designs and system implementations.

Within the artificial intelligence community, issues of knowledge acquisition and knowledge sharing have raised similar software reuse challenges, and have pushed research in related directions. Knowledge engineering, for example, has evolved from a process concerned principally with knowledge extraction, where each application is considered uniquely, to a model construction process, where applications are categorized according to type of task and applicable methods [Wielinga *et al.*, 1992, Steels, 1990]. Similarly, trends toward definition of generic tasks and task-specific problem solving architectures (e.g., [Chandrasekaran, 1986]) are motivated by many of the same reasons that underlie software architecture research. One important area of recent research in knowledge-based systems has been the

---

Constructing Scheduling Systems”, Proceedings AAAI Spring Symposium on Ontological Engineering, Palo Alto, CA, 1997.

development and use of *ontologies* [Gruber, 1993, Uschold, 1996, Swartout *et al.*, 1996]. This work has concentrated primarily on issues of reusable or sharable knowledge bases, focusing on formalizing particular bodies of knowledge, on languages for encoding ontologies, and on methodologies for ontology construction. The broader relevance of ontologies to design and specification of task-specific problem solvers has also been recognized [Wielinga *et al.*, 1992, Wielinga and Schreiber, 1993], but has received less attention.

The work reported in this paper takes a similar, extended view of the role of ontologies. We advocate the use of ontologies as a means of bridging the gap between domain analysis and application system construction. Our basic approach is to consider an ontology as a framework for specifying models in a particular problem domain, i.e., a meta-model that provides a vocabulary for formulating application models in this problem domain, as well as a set of constraints on what can be expressed. The scope of the ontology is restricted to a particular problem domain, which permits much stronger assumptions to be made with regard to system architecture and sub-structure. On this basis, concepts in the ontology can be explicitly linked to software component capability descriptions, enabling the ontology to serve both as an mechanism for indexing and retrieving relevant software components and as a specification of overall configuration requirements. More generally, the association of component capabilities with concept definitions in the ontology promotes direct configuration of executable systems from specification of an abstract domain model.

This approach to application system construction underlies the design of OZONE, an object-oriented toolkit for configuring constraint-based scheduling systems [Smith *et al.*, 1996, Becker, 1998]. In the sections below, we describe the ontology that OZONE provides for formulating scheduling domain models.

## **3.2 The Structure of Domain Models in OZONE**

As discussed above, the OZONE scheduling ontology can be characterized as a meta-model of the domain of scheduling. It provides a language for describing those aspects of the scheduling domain that are relevant to construction of an application system, and a set of constraints on how concepts in the language fit together to form consistent domain models. Consistency, in this context, relates to the information and knowledge required to insure executability of the model. Generally speaking, the ontology serves to map user-interpretable descriptions of an application domain to application system functionality.

This linkage is established within the OZONE ontology through the inclusion of *capabilities* as an integral part of concept definition. Capabilities provide an operational semantics to the concepts defined in the the ontology, in a form that reflects a specific bias with respect to application system design. In particular, the ontology presumes an underlying constraint-based solution framework and scheduling system architecture [Smith, 1994, Smith *et al.*, 1996]; this commitment follows directly from the strong match of constraint-based techniques to the decision-support requirements of practical scheduling environments. Capabilities, then, encapsulate reusable components for configuring and customizing constraint-based solution methods. For example, the concept of a “resource” contributes capabilities for querying and managing its available capacity over time, and different resource types (e.g., reusable, consumable) provide specific “implementations”. Given a solution method that incorporates these capabilities, the ontology provides a direct basis for its customization to match the resources in any target domain.

In the remainder of this section, we summarize the basic components of the OZONE scheduling ontology. By convention, we use capitalization to distinguish specific concepts that are included. We start with an overview of the principal concepts involved and their inter-relationships, and then consider each individually in more detail.

### 3.2.1 Basic components of scheduling models and their relationships

Like several contemporary process modeling and ontology development efforts [Uschold *et al.*, 1996, Gruninger and Fox, 1994, Le Pape, 1994, Lee *et al.*, 1996, Tate, 1996, Smith, 1989] the OZONE scheduling ontology adopts an activity-centered modeling viewpoint. Scheduling is defined as a process of feasibly synchronizing the use of RESOURCES by ACTIVITIES to satisfy DEMANDS over time, and application problems are described in terms of this abstract domain model. Figure 3.1 illustrates the base concepts involved and their structural relationships. A DEMAND is an input request for one or more PRODUCTS, which designate the GOODS or SERVICES required. Satisfaction of DEMANDS centers around the execution of ACTIVITIES. An ACTIVITY is a process that uses RESOURCES to produce goods or provide services. The use of RESOURCES and the execution of ACTIVITIES is restricted by a set of CONSTRAINTS.

These five base concepts of the ontology - **DEMAND**, **ACTIVITY**, **RESOURCE**, **PRODUCT**, and **CONSTRAINT** - together with the inter-relationships depicted in Figure 3.1, de-

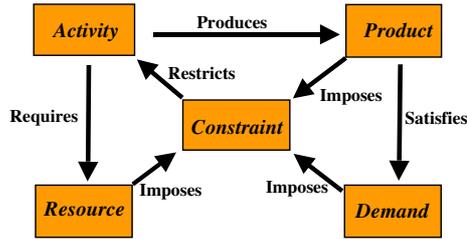


Figure 3.1: Abstract Domain Model

fine an abstract model of a scheduling domain, and a framework for analyzing and describing particular application environments. Associated with each concept definition are terminologies for describing basic properties and capabilities. Properties define attributes or parameters of relevance to specifying an executable scheduling model. The abstract model and its properties are extensible through concept specializations to define more specific models for various subdomains. Figure 3.2 indicates model specializations for two such subdomains: manufacturing production scheduling and transportation scheduling. Capabilities designated in the abstract model, alternatively, establish protocols for operationalizing concept definitions in terms of the component functionality required to compose overall solution methods. Specializations of concepts in the abstract model, then, provide a library of implementations that reflect important ontological distinctions. In this respect, the abstract model underlying the OZONE ontology can be viewed as a template for specifying executable domain models.

### 3.2.2 Associated Capabilities

The capabilities defined in the abstract domain model relate generally to aspects of solution and constraint management, and, as indicated earlier, are rooted in an underlying constraint-based problem solving model. In OZONE, plans and schedules are represented as networks of ACTIVITIES, with an ACTIVITY containing various decision variables (e.g., start time, end time, assigned resources). To construct a schedule that satisfies a given input DEMAND, it is necessary to first instantiate a set of ACTIVITIES that will produce (provide) the designated PRODUCT. This instantiation process is accomplished by *Instantiate-Product-Plan*, a joint capability of DEMAND and PRODUCT that integrates prototype plan information defined by the PRODUCT with the specific parameters of the triggering DEMAND. One consequence of *Instantiate-Product-Plan*, for example, is the imposition of constraints on the

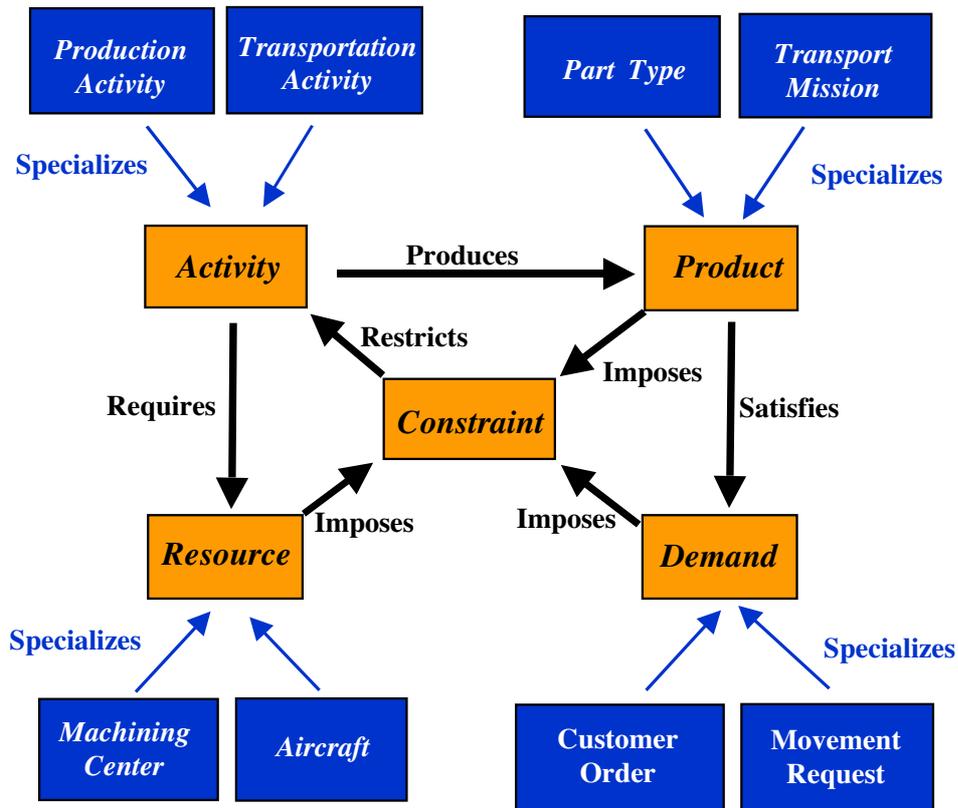


Figure 3.2: Layered models of scheduling subdomains

start and end times of instantiated activities following from the READY and DUE DATES specified in the DEMAND.

To schedule an ACTIVITY, it is necessary to choose specific RESOURCES, which involves determining intervals where resources have capacity available to support execution of the ACTIVITY, and subsequently allocating capacity of chosen RESOURCES to ensure that they will not be used by other ACTIVITIES. The semantics of allocating (and de-allocating) resource capacity varies according to the type of RESOURCE involved, and hence a RESOURCE provides primitive *Allocate-Capacity* and *Deallocate-Capacity* capabilities. To this end, a RESOURCE maintains a representation of its available capacity over time. A third RESOURCE capability, *Find-Available-Time*, uses this representation to provide a mechanism for identifying intervals of time where the RESOURCE currently has available capacity.

A *Find-Schedulable-Time* capability is associated with an ACTIVITY, which intersects the availability intervals found by a given set of required RESOURCES. This capability, and a companion *Find-Alternative-Resources* capability, provide generic primitives for elaborating a space of alternative decisions. Another pair of ACTIVITY capabilities, *Reserve-Resources* and *Free-Resources*, provide complementary primitives for committing to and retracting specific scheduling decisions. Both *Reserve-Resources* and *Free-Resources* rely, in turn, on the *Propagate-Constraints* capabilities associated with temporal and value CONSTRAINTS to incrementally update the possible values for various decision variables.

### 3.3 The OZONE Scheduling Ontology

In the following subsections, we consider the five basic components of the OZONE scheduling ontology in more detail. In doing so, we assume the existence of basic temporal concepts such as TIME-INTERVALS and TIME-POINTS (c.f. [Allen, 1984]).

#### 3.3.1 DEMANDS

##### **Concept Definition.**

A **DEMAND** is a request for goods and services, or more generically PRODUCTS, that the system being modeled can provide. DEMANDS specify the input goals that drive the system, along with any CONSTRAINTS that must be taken into account when achieving them. The set of outstanding DEMANDS at any point determine the current scheduling problem to be solved.

##### **Properties.**

A DEMAND has several defining properties:

- **PRODUCT** - The PRODUCT is the object of the DEMAND. It specifies the type of good or service that is requested.
- **RELEASE-DATE** - The earliest time an ACTIVITY for achieving the DEMAND can start.

- **DUE-DATE** - The latest time an **ACTIVITY** for achieving the **DEMAND** should end.
- **TEMPORAL-RELATIONS** - These are synchronization constraints with respect to achievement of other system **DEMANDS**
- **PRIORITY** - The relative importance of the **DEMAND**, providing a basis for establishing a partial ordering over the entire set of demands.
- **ACTIVITIES** - The set of activities that fulfill the **DEMAND**. As indicated earlier, these are determined by *Instantiate-Product-Plan*, a joint capability of **DEMAND** and **PRODUCT** concept definitions.

For most types of **DEMANDS**, there will be additional parameters which further specify the requested **PRODUCT**. **DEMAND** parameters will vary for different types of goods or services, but typical parameters include:

- **QUANTITY** - A parameter relating to the size of the **DEMAND** (e.g., the number of goods requested, the amount of material to be processed)
- **MATERIAL** - A parameter relating to the type of material that must be processed
- **ORIGIN, DESTINATION** - If the **DEMAND** is a request for material to be moved, then **ORIGIN** and **DESTINATION** locations are necessary parameters

### 3.3.2 PRODUCTS

#### **Concept Definition.**

A **PRODUCT** is a good or service provided by some system of interest. A **PART-TYPE** is a typical **PRODUCT** of a manufacturing system; a transportation system alternatively provides **TRANSPORT-SERVICES**. A **PRODUCT** is realized through execution of some set of activities. A **DEMAND** for a **PRODUCT** is considered satisfied when all of these activities have completed.

## Properties.

From the standpoint of managing system activities in response to external DEMANDS, properties of interest in defining a PRODUCT relate to the mapping from DEMANDS to ACTIVITIES. Specifically, a PRODUCT definition includes the following:

- **ACTIVITIES** - the set of processing steps required to produce or provide the PRODUCT (i.e., a plan for realizing this PRODUCT)
- **RESOURCES** - the set of resources that can be utilized to execute various ACTIVITIES of the PRODUCT plan.

A PRODUCT specification, together with the constraints and parameters of a requesting DEMAND, enables instantiation of a set of ACTIVITIES for fulfilling the DEMAND (*Instantiate-Product-Plan*). From a scheduling perspective, these ACTIVITIES contain the decision variables (start times, end times, assigned resources) of the problem to be solved; and the instantiation process restricts the domains of these decision variables according to the constraints specified in the DEMAND.

### 3.3.3 RESOURCES

#### Concept Definition.

Central to the definition of our scheduling ontology is the concept of a RESOURCE. A **RESOURCE** is an entity that supports or enables the execution of ACTIVITIES. RESOURCES are generally in finite supply and their availability constrains when and how ACTIVITIES execute. Making efficient use of RESOURCES in support of multiple, competing ACTIVITIES is the crux of the scheduling problem, and, from the standpoint of constructing scheduling models, the distinguishing characteristics of RESOURCES relate to constraints on their availability.

The availability of a RESOURCE can be defined generally in terms of some dynamically changing aspect of state. Most typically, a RESOURCE is modeled as providing some amount of CAPACITY, a numeric quantity which varies over time as a function of allocating the RESOURCE to various ACTIVITIES and its associated allocation semantics. This is the approach taken in [Fadel *et al.*, 1994, Uschold *et al.*, 1996]. However, there are also

RESOURCES whose availability is more a function of qualitative state: ACTIVITIES require the RESOURCE to be in a particular state or subset of possible states (e.g., to be *idle* as opposed to *busy*) rather than requiring that the RESOURCE have a sufficient amount of CAPACITY. Hence we distinguish two broad classes of resources from the standpoint of availability:

- **CAPACITATED-RESOURCES** - RESOURCES whose availability is characterized in terms of the amount of CAPACITY that is available. In this case, concept specializations provide capabilities for maintaining a representation of available capacity over time (*Increase-Capacity*, *Decrease-Capacity*), for allocating and deallocating capacity to activities (*Allocate-Capacity*, *DeAllocate-Capacity*), and for finding periods where capacity is available (*Find-Available-Time*).
- **DISCRETE-STATE-RESOURCES** - RESOURCES whose availability is a function of some discrete set of possible state values. Here definitions provide analogous capabilities for querying, updating and protecting state values over time.

In the case of CAPACITATED-RESOURCES, constraints on availability (i.e. usage of capacity) depend on several different properties of the resource. One determining characteristic is whether RESOURCE CAPACITY is used or consumed by an ACTIVITY when it is allocated:

- A **REUSABLE-RESOURCE**, is a RESOURCE whose capacity becomes available for reuse after an ACTIVITY to which it has been allocated finishes. We say that the ACTIVITY uses the RESOURCE [Uschold *et al.*, 1996]
- A **CONSUMABLE-RESOURCE**, is one whose CAPACITY, once allocated to an ACTIVITY does not become available again. We say that the ACTIVITY consumes the RESOURCE.

Though we could further distinguish a third class, *RENEWABLE-RESOURCES*, to refer to RESOURCES that have their CAPACITY increased by ACTIVITIES [Fadel *et al.*, 1994], we instead consider production of RESOURCE CAPACITY to be a separable issue. In our model, ACTIVITIES utilize RESOURCES to produce PRODUCTS. In a resource producing ACTIVITY, the RESOURCE CAPACITY generated is the PRODUCT (or output) of the

ACTIVITY; it is not assuming the role of a RESOURCE in this context. Moreover, RENEWABILITY is a property that is equally relevant to REUSABLE-RESOURCES as well as CONSUMABLES. Any RESOURCE can be designated as RENEWABLE by additionally defining it to be a PRODUCT.

A second aspect of RESOURCES that impacts usage (or consumption) of CAPACITY by ACTIVITIES is physical structure. In this respect, RESOURCES can be classified as:

- **ATOMIC-RESOURCE** - This is a RESOURCE that is not divisible and can only be configured to support one process at a time. We can distinguish two subtypes:
  - A **UNIT-CAPACITY-RESOURCE** can only be used by one ACTIVITY during any given TIME-INTERVAL. In this case we could equivalently model the RESOURCE as a discrete state variable with two values : *busy* and *idle*.
  - A **BATCH-CAPACITY-RESOURCE** can support multiple ACTIVITIES if there is sufficient capacity, and if they they require the same resource configuration and are temporally synchronized to occur over the same TIME-INTERVAL. BATCHING-COMPATIBILITY constraints specify the commonality in resource configuration that is required of multiple ACTIVITIES for simultaneous use of a BATCH-CAPACITY-RESOURCE. These constraints are defined with respect to different types of ACTIVITIES that the RESOURCE can support. For example, for two TRANSPORT-ACTIVITIES to be supported by the same vehicle at the same time, both have to require transport between the same locations.
- An **AGGREGATE-RESOURCE** represents a pool of resources, which may be composed of smaller AGGREGATE-RESOURCES or ATOMIC-RESOURCES. The CAPACITY of an AGGREGATE-RESOURCE reflects the collective CAPACITY of its constituent SUB-RESOURCES. This CAPACITY can be independently allocated to multiple activities over any given TIME-INTERVAL, subject only to any constraints induced from the structure of the aggregated SUB-RESOURCES. Based on the nature of SUB-RESOURCE structure, we can define several types of AGGREGATE-RESOURCE:
  - **HOMOGENEOUS-RESOURCE-POOL** - An AGGREGATE-RESOURCE composed of  $n$  SUB-RESOURCES of the same type. HOMOGENEOUS-RESOURCE-POOLS can be further differentiated as:

- \* **SIMPLE-CAPACITY-POOL** - A HOMOGENEOUS-RESOURCE-POOL which is composed of  $n$  UNIT-CAPACITY-RESOURCES and can thus simultaneously support  $n$  independent activities. This corresponds to the definition of CAPACITATED-RESOURCE given in [Fadel *et al.*, 1994].
- \* **STRUCTURED-CAPACITY-POOL** - A HOMOGENEOUS-RESOURCE-POOL composed of  $n$  BATCH-CAPACITY-RESOURCES or  $n$  AGGREGATE-RESOURCES of capacity  $c$ , having total CAPACITY  $n * c$ . This type of resource can simultaneously support  $n$  independent activities only if the capacity required by any one activity  $\leq c$ . Any extra capacity over a given TIME-INTERVAL can potentially be used to support additional activities, but only if COMPATIBILITY constraints are satisfied.

– **HETEROGENEOUS-RESOURCE-POOL** - An AGGREGATE-RESOURCE that is composed of RESOURCES of different types and CAPACITIES.

Regardless of the level of detail at which RESOURCE allocation decisions are to be considered in a given domain (e.g., at the level of ATOMIC-RESOURCES or higher), AGGREGATE-RESOURCES capture the hierarchical structure of domain resources in most environments. One consequence is that the unavailability of an AGGREGATE-RESOURCE over a given TIME-INTERVAL always implies the unavailability of its constituent SUB-RESOURCES over the same TIME-INTERVAL.

### Properties.

The properties of a RESOURCE of primary interest here are those which affect its availability and utilization. Lets first consider availability. In the case of a CAPACITATED-RESOURCE, availability is a function of its **CAPACITY**. CAPACITY is a QUANTITY (or set of QUANTITIES) of some unit measure (e.g., volume, weight, number of activities) that is available for allocation to ACTIVITIES over time. The allocation of a CAPACITATED-RESOURCE to an ACTIVITY implies use or consumption of some amount of CAPACITY, and the number of ACTIVITIES that can be simultaneously supported is limited by the total CAPACITY of the RESOURCE. We can distinguish between different types of CAPACITY models, which impose different CAPACITY CONSTRAINTS:

- A **UNIFORM-CAPACITY** model represents CAPACITY as a scalar QUANTITY. The CAPACITY-CONSTRAINT of a RESOURCE with UNIFORM-CAPACITY re-

quires that, at any point in time, the sum of the CAPACITY used/consumed by all supported ACTIVITIES  $\leq$  the CAPACITY of the RESOURCE.

- A **HETEROGENEOUS-CAPACITY** model represents CAPACITY as a vector of two or more UNIFORM-CAPACITIES, reflecting partitioned sub-CAPACITIES. For example, a ship might have separate cargo holds. The CAPACITY-CONSTRAINT of a RESOURCE with HETEROGENEOUS-CAPACITY is the conjunction of the CAPACITY-CONSTRAINTS associated with constituent UNIFORM-CAPACITIES.
- A **MULTI-DIMENSIONAL-CAPACITY** model defines CAPACITY in terms of two or more QUANTITIES, with each contributing a separate CAPACITY-CONSTRAINT that must be satisfied. For example, the capacity of an aircraft might be defined in terms of both maximum weight and volume. In the case of MULTI-DIMENSIONAL-CAPACITY, the CAPACITY-CONSTRAINT requires that for each different unit measure, the sum of the CAPACITY utilized by all supported ACTIVITIES  $\leq$  the CAPACITY of the RESOURCE.

In the case of a DISCRETE-STATE-RESOURCE, “availability” corresponds to being in a state that matches the condition of the ACTIVITY that requires the resource. A DISCRETE-STATE-RESOURCE has a set of possible STATE-VALUES. If the RESOURCE is controllable, individual STATE-VALUES can be additionally defined as PRODUCTS; this allows linkage to ACTIVITIES for bringing about their specific values. Allocation of a DISCRETE-STATE-RESOURCE to an ACTIVITY implies commitment to (or protection of) a specific STATE-VALUE over some TIME-INTERVAL, and multiple ACTIVITIES can be simultaneously supported, as long as compatible STATE-VALUES are required.

In many cases, usage of a RESOURCE also depends on other physical properties. Generally, the physical properties of interest will be a function of the domain, but some fairly generic examples include its **SPEED**, which constrains how long ACTIVITIES take to perform, and its **RANGE**, which affects whether it can be used for a particular ACTIVITY or not. Another general physical property of a REUSABLE-RESOURCE is its **SETUP-DURATION**, which specifies how long it takes to configure the RESOURCE for use by a particular ACTIVITY. We can distinguish different types of SETUP-DURATION models:

- A **CONSTANT-SETUP-TIME** model implies that the RESOURCE requires a fixed amount of time to be configured for use by an ACTIVITY, regardless of its prior state.

- **STATE-DEPENDENT-SETUP-TIME** - Model implies that the amount of time required to configure the RESOURCE for use by an ACTIVITY is variable and depends on the specific prior configuration of the RESOURCE. A special form of STATE-DEPENDENT-SETUP-TIME is **SEQUENCE-DEPENDENT-SETUP-TIME**, where setup time is assumed to be a function of the last ACTIVITY that was processed using the RESOURCE.

Other **USAGE-RESTRICTIONS** can also limit the availability of RESOURCES:

- **UNAVAILABILITY-INTERVALS**-A TIME-INTERVAL where a RESOURCE cannot be allocated is one simple type of USAGE-RESTRICTION. UNAVAILABILITY-INTERVALS can reflect **RESOURCE-BREAKDOWNS**, periods where **DOWN-SHIFTS** or **RESOURCE-MAINTENANCE** have been planned, or other unmodeled circumstances.

- **CUMULATIVE-USAGE-CONSTRAINTS**-There may also be restrictions on the total amount of RESOURCE use permitted over a given TIME-INTERVAL.

### 3.3.4 ACTIVITIES

#### **Concept Definition.**

An **ACTIVITY** represents a process that can be executed over a certain time interval. An ACTIVITY requires RESOURCES to execute and its execution both depends on and affects the current state of these RESOURCES. An ACTIVITY can also have other EFFECTS (e.g., PRODUCTS are produced, other enabling RESOURCE states are established), and it is these EFFECTS that lead ultimately to satisfied DEMANDS. An ACTIVITY may be decomposable into a set of more-detailed SUB-ACTIVITIES, enabling processes to be described at multiple levels of abstraction.

#### **Properties.**

From the standpoint of the problem solver, an ACTIVITY designates a set of decision variables. The action of *scheduling* an ACTIVITY involves determining values for these variables. The basic decision variables associated with an ACTIVITY are:

- **START-TIME, END-TIME**, which delineate the interval during which the ACTIVITY will occur, and
- **ASSIGNED-RESOURCES**, which indicates the set of RESOURCES allocated to the ACTIVITY

An ACTIVITY has a number of properties that constrain the values that can be assigned to these decision variables:

- **DURATION** - the time required for the ACTIVITY to execute.
- **RESOURCE-REQUIREMENTS** - the set of RESOURCE usage/consumption constraints that must be satisfied for the ACTIVITY to execute.
- **RELATIONS** - the set of TEMPORAL-RELATIONS between this ACTIVITY and others.
- **DEMAND** - the DEMAND that the ACTIVITY was instantiated to satisfy. The DEMAND imposes EARLIEST-START-TIME and LATEST-FINISH-TIME constraints, and associates PRIORITY information.
- **PARAMETERS** - depending on the type of ACTIVITY, there may be one or more PARAMETERS relating to the ACTIVITY's associated DEMAND. For example, if the associated DEMAND is for a QUANTITY of some PRODUCT, then the ACTIVITY might also have a QUANTITY, in this case indicating the portion of the total QUANTITY that it produces.
- **STATUS** - an ACTIVITY may be in one of several states: *UNSCHEDULED*, *SCHEDULED*, *IN-PROCESS*, or *COMPLETED*.

Following a constraint-based problem solving orientation, an ACTIVITY provides capabilities for incrementally allocating resources and making variable assignments (*Reserve-Resources*), for retracting previous assignments (*Free-Resources*), and for propagating the consequences of these decisions to related ACTIVITIES (*Propagate-Constraints*). An ACTIVITY thus maintains EARLIEST and LATEST bounds on its START-TIME and END-TIME, as well as a set of currently feasible RESOURCE-ALTERNATIVES. An ACTIVITY also defines primitives for exploring alternative sets of resource assignments (*Find-Alternative-Resources*) and alternative intervals where resources are simultaneously available (*Find-Schedulable-Time*).

### 3.3.5 CONSTRAINTS

#### Concept Definition.

Generally speaking, a **CONSTRAINT** restricts the set of values that can be assigned to a variable. In the scheduling domain, **CONSTRAINTS** restrict the assignment of **START** and **END-TIMES** and the allocation of **RESOURCES** to **ACTIVITIES**. From this perspective, we can identify several basic types:

- **VALUE-COMPATIBILITY-CONSTRAINTS** restrict the values of non-temporal decision variables, and specify conditions under which a value assignment to a given variable is compatible with those of other variables or properties in the model. In the case of basic scheduling models, these **CONSTRAINTS** relate specifically to **RESOURCE** assignment decisions and are referred to as **RESOURCE-COMPATIBILITY-CONSTRAINTS**. They designate the conditions under which a given **RESOURCE** (or type of **RESOURCE**) can be feasibly used to perform a given **ACTIVITY**. They may represent physical capabilities and limitations of **RESOURCES**, or external (e.g., user-imposed) restrictions.

We can distinguish two varieties of **VALUE** (or **RESOURCE**) **COMPATIBILITY-CONSTRAINTS**:

- A **STATIC-COMPATIBILITY** specifies a resource usage condition that depends on some other static property of the **ACTIVITY** that requires the **RESOURCE** (e.g., parameters of its associated **DEMAND**, **PRODUCT** characteristics, other properties of the **ACTIVITY** itself). For example, an aircraft has a maximum range and may also be capable of carrying only certain types of cargo. Depending on the type of cargo to be moved and the distance of the cargo's destination (both parameters of the input **DEMAND**), this aircraft may or may not be a compatible (feasible) **RESOURCE** assignment. From a problem solving perspective, **STATIC-COMPATIBILITY-CONSTRAINTS** can be applied when an **ACTIVITY** is first instantiated to prune the space of alternatives in advance of scheduling.
- A **DYNAMIC-COMPATIBILITY** specifies a compatibility condition or dependency between two **RESOURCE** assignments (or more generally between two

decision variables). It may involve separate RESOURCE assignments for a single ACTIVITY (e.g., the chosen air crew must be qualified to fly the chosen aircraft) or may constrain the RESOURCE assignments of two distinct activities (e.g., the chosen aircraft for both legs of the flight must be the same). One specific type of DYNAMIC-COMPATIBILITY mentioned earlier is a **BATCHING-COMPATIBILITY**, which dictates the circumstances under which two ACTIVITIES can simultaneously use capacity of a BATCH-CAPACITY-RESOURCE. DYNAMIC-COMPATIBILITY-CONSTRAINTS must be reapplied each time a decision is made.

- **TEMPORAL-CONSTRAINTS** restrict the values of temporal decision variables, i.e., ACTIVITY START-TIMES and END-TIMES. There are two basic types:

- **AN ABSOLUTE-TIME-CONSTRAINT**

places an absolute lower or upper bound on the value of a TIME-POINT. Examples of ABSOLUTE-TIME-CONSTRAINTS previously mentioned include The RELEASE-DATE-CONSTRAINT and the DUE-DATE-CONSTRAINT imposed by a DEMAND.

- A **RELATIVE-TIME-CONSTRAINT**, alternatively, restricts the separation between two TIME-POINTS. According to whether or not the constrained TIME-POINTS belong to the same interval or not, we define two subtypes:

- \* **INTERVAL-RELATIONS** - An INTERVAL-RELATION synchronizes the occurrence of two TIME-INTERVALS (e.g., two ACTIVITIES). It specifies an ordering with respect to the respective START-TIMES and/or END-TIMES of the two related intervals, and the relation may be quantified by a metric LOWER-BOUND and UPPER-BOUND on the temporal separation between ordered TIME-POINTS. An unquantified INTERVAL-RELATION is interpreted as having LOWER-BOUND, UPPER-BOUND values of  $0, \infty$ . The set of INTERVAL-RELATIONS includes:

- **BEFORE** - For two intervals  $I_1$  and  $I_2$ ,  $I_1$  BEFORE  $I_2[lb, ub]$  implies that  $ST(I_2) \geq ET(I_1) + lb \wedge ST(I_2) \leq ET(I_1) + ub$ .
- **SAME-START** - For two intervals  $I_1$  and  $I_2$ ,  $I_1$  SAME-START  $I_2[lb, ub]$  implies that  $ST(I_2) \geq ST(I_1) + lb \wedge ST(I_2) \leq ST(I_1) + ub$ .

- **SAME-END** - For two intervals  $I_1$  and  $I_2$ ,  $I_1$  SAME-END  $I_2[lb, ub]$  implies that  $ET(I_2) \geq ET(I_1) + lb \wedge ET(I_2) \leq ET(I_1) + ub$ .
- **CONTAINS** - For two intervals  $I_1$  and  $I_2$ ,  $I_1$  CONTAINS  $I_2[lb_1, ub_1, lb_2, ub_2]$  implies that  $ST(I_2) \geq ST(I_1) + lb_1 \wedge ST(I_2) \leq ST(I_1) + ub_1 \wedge ET(I_1) \geq ET(I_2) + lb_2 \wedge ET(I_1) \leq ET(I_2) + ub_2$ .
- \* **DURATION-CONSTRAINTS** - A DURATION-CONSTRAINT imposes a LOWER-BOUND or UPPER-BOUND (or both) on the separation between the START and END points of a given TIME-INTERVAL. For interval  $I_1$  and  $[lb, ub]$ ,  $lb \leq ET(I_1) - ST(I_1) \leq ub$ . An ACTIVITY-DURATION and a RESOURCE'S SETUP-DURATION are two previously mentioned types of DURATION-CONSTRAINTS.
- **RESOURCE-AVAILABILITY-CONSTRAINTS** define third class of physical CONSTRAINT which impacts the assignments of both RESOURCES and START/END-TIMES to ACTIVITIES. The various types of CAPACITY-CONSTRAINTS and USAGE-RESTRICTIONS discussed earlier fall into this category.
- **INSTANTIATION-CONSTRAINTS** represent a final class of CONSTRAINT which restricts the creation of decision variables. In the case of basic scheduling models, decision variables are properties of ACTIVITIES and we refer to this category of constraints more specifically as **ACTIVITY-INSTANTIATION-CONSTRAINTS**. ACTIVITY-INSTANTIATION-CONSTRAINTS include restrictions on how DEMANDS can be mapped to sets of ACTIVITIES. For example, in distributing the cargo that must be moved to satisfy a transport DEMAND across several movement ACTIVITIES (e.g., due to vehicle CAPACITY limitations), there may be physical constraints on how the cargo can be disaggregated.

### Properties.

A CONSTRAINT may be considered to be **HARD** or **SOFT**. The problem solver is never allowed to violate HARD-CONSTRAINTS. SOFT-CONSTRAINTS, alternatively, are considered to be **RELAXABLE** if need be. For example, DUE-DATE-CONSTRAINTS are treated as RELAXABLE-CONSTRAINTS in many scheduling contexts. The designation of RELAXABLE-CONSTRAINTS is typically accompanied by a specification of **OBJECTIVES** or **PREFERENCES**. When due dates can be relaxed, for example, minimizing tar-

diness is a common OBJECTIVE. OBJECTIVES and PREFERENCES prioritize the space of possible RELAXATIONS of a CONSTRAINT and provide a basis for measuring solution quality.

### 3.4 Concluding Remarks

The OZONE scheduling ontology is the result of considerable prior experience in building planning and scheduling systems, in application domains ranging from manufacturing production scheduling [Smith, 1994] to space mission planning [Muscettola *et al.*, 1992] to transportation logistics [Smith and Lassila, 1994]. The class library design and implementation have followed from retrospective analysis of these scheduling domains and systems (e.g., [Becker and Díaz-Herrera, 1994]), together with application of object-oriented analysis and design principles [Smith and Lassila, 1994]. As evidence of the efficacy of the model-building approach, OZONE was recently applied to develop a quite substantial prototype for reactive, aero-medical evacuation replanning in just two person-months time [Lassila *et al.*, 1996]. Though bearing similarity in many respects to various strategic deployment scheduling problems previously addressed with OZONE, the medical evacuation domain also required some fundamentally different capabilities (e.g., integration of itinerary routing and resource allocation decision-making). The use of the ontology and associated constraint management capabilities enabled application development to be quickly localized to those aspects of the domain that required component specialization or extension.

The scheduling ontology presented above is mainly concerned with modeling the entities and constraints of a particular domain. It does not address issues relating to the development of problem solving methods and heuristics that best match domain requirements and characteristics. Though not discussed in this paper, the OZONE toolkit does also provide a companion, agenda-based framework for configuring and integrating problem-solving methods to meet domain-specific requirements. One goal of our current research is to extend our ontological approach to domain modeling to cover this aspect of application construction as well.

### **3.5 Acknowledgements**

Ora Lassila has been a principal contributor to the design and development of the OZONE scheduling toolkit, and the work described in this paper has benefited significantly from his association.

This work was sponsored in part by the Department of Defense Advanced Research Projects Agency and Rome Laboratory, Air Force Material Command, USAF, under grant numbers F30602-90-C-0119, F30602-95-1-0018 and F30602-97-C-0227 and the CMU Robotics Institute. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Advanced Research Projects Agency and Rome Laboratory or the U.S. Government.

## Chapter 4

# Interactive Visualizations for Requirements Analysis

**Summary:** In most practical domains, scheduling is not a one-shot generative task of producing time and resource assignments for pre-specified sets of requirements and capabilities, but an iterative process of getting the constraints right. Initial solutions are developed (at some level of detail) to understand the feasibility of various requirements and the sufficiency of assumed resources. This requirements analysis leads to reassessment of what requirements can be reasonably met and what resources need to be made available. Changes are made to the constraints governing requirements and resource availability, and eventually a final schedule is elaborated. At early stages of this user-driven process, seeing a fully instantiated schedule is less important than simply knowing whether a feasible schedule exists and, if not, the approximate magnitude of the shortfalls for different resources and periods of time. To this end, solutions to relaxed versions of complete scheduling problems can provide helpful guidance.

In this paper, we describe a system that provides a *direct manipulation* visual interface for requirements analysis and reconciliation. The interface incorporates a 3D visualization that identifies resource capacity shortfalls in a tractable relaxed version of the problem, which are necessarily shortfalls in the original problem. Our visualization can be contrasted with common 2D scheduling displays such as Gantt Charts and Closure Graphs, which are designed for visualizing complete solutions to a given scheduling problem and hence offer only indirect support for identifying inherently over-constrained regions of the solution space. Alternatively, our visualization directly characterizes this underlying constraint space and provides a direct basis for requirements analysis. An analyst iteratively adjusts problem constraints and visualizes the resulting (relaxed) problem solution until various mismatches between resource requirements and resource availability are satisfactorily reconciled. Once a reasonable compromise is found, the same interface can then be used to guide more detailed scheduling.<sup>1</sup>

---

<sup>1</sup>This Chapter has been submitted for publication, and is currently available as: Mark Derthick and Stephen F. Smith, “An Interactive 1D+2D+3D Visualization for Requirements Analysis”, ICLL Working Paper, The

## 4.1 Introduction

Scheduling is traditionally identified as the task of assigning a pre-specified set of available resources to activities over time to achieve some pre-specified set of demands. The goal is generally a solution that both ensures a feasible behavior in the target domain (i.e., is executable) and optimizes overall system performance. Feasibility can be a function of a large and idiosyncratic set of constraints. Optimization can involve several potentially conflicting objective criteria. Both aspects contribute to the overall complexity of the scheduling problem.

Despite the ultimate objective of producing a feasible schedule that optimizes overall performance, scheduling in most practical domains is concerned with solving a problem of much larger scope, which additionally involves the specification, negotiation and refinement of input requirements and system capabilities. This larger process is concerned with getting the constraints right: determining the mix of requirements and resource capacity that leads to the most effective overall system performance.

In performing this sort of *requirements analysis*, particularly at early stages of the planning process, it may be uninformative, or even counterproductive, to generate and work with fully elaborated schedules. There is a computational cost to finding complete schedules. Further, the constraint violations found in one particular schedule do not necessarily lend insight into the tradeoffs or implications of adjusting specifications. A higher-level analysis is more appropriate.

One commonly employed approach to gaining insight into the structure of a problem is to solve relaxed versions. In the simplest case, a relaxed problem formulation is one that drops one or more constraints from the problem, and/or makes subproblem independence assumptions that ignore specific constraint interactions. One useful property of a relaxed model is that it is optimistic in its predictions. Any infeasibility (i.e., irresolvable constraint conflict) that is detected in solving a relaxed problem is guaranteed to also be present in the full problem. Relaxed scheduling problems can generally be solved more efficiently, and in some cases, without appealing to approximate (heuristic) scheduling procedures. Hence, this approach provides a natural basis for recognizing gross mismatches in system demands and capabilities, and for making changes to problem constraints that reconcile these differences.

In this paper, we build on this notion to specify an interactive environment for requirements

---

Robotics Institute, Carnegie Mellon University, March 2003.

analysis. Central to our approach is a 3D visualization of the relationship between resource demand and capacity over intervals of time, which generalizes from 2D visualizations that are currently used for requirements analysis in large-scale military deployment planning contexts. The visualization is derived from a relaxed problem formulation that can be efficiently solved, allowing real-time animation of consequences as constraints on requirements and resource availability are manipulated. A slight extension of the underlying model allows the same visualization to serve as a basis for user-guided generation of detailed schedules that respect pre-established requirements and resource availability constraints.

Our approach provides a basis for recognizing and reacting to periods of infeasibility early in the overall analysis process, before significant time has been invested in generating detailed solutions to versions of the full problem.

- It focuses on visually bounding the space of possible solutions rather than visualizing a particular generated solution, hence providing direct support for diagnosis and reconciliation of infeasible requirements.
- It visualizes the relationships among inter-dependent sets of conflicting demands, which guides analysts to address problems in a logical order. It shows temporally localized groups of conflict sets, and the most constrained conflict sets in each group. It is also easy to see whether encompassing temporal intervals have similar capacity problems, or whether they instead contain excess capacity that might be exploited. This information sheds light on the prospective viability of different strategies for adjusting problem constraints, such as reallocating demands to different resources or loosening their due dates.

The remainder of the paper describes:

- limitations of current tools and approaches to requirements analysis.
- our interface and its use for requirements analysis in the context of a particular resource capacity and demand model from the domain of transportation scheduling.
- extension of the underlying idea to other models of demand, and the complementary use of the approach for user-guided scheduling, as opposed to requirements analysis. Using the same interface for both types of tasks is advantageous because the boundary between the two is fuzzy, and analysis alternates between them.

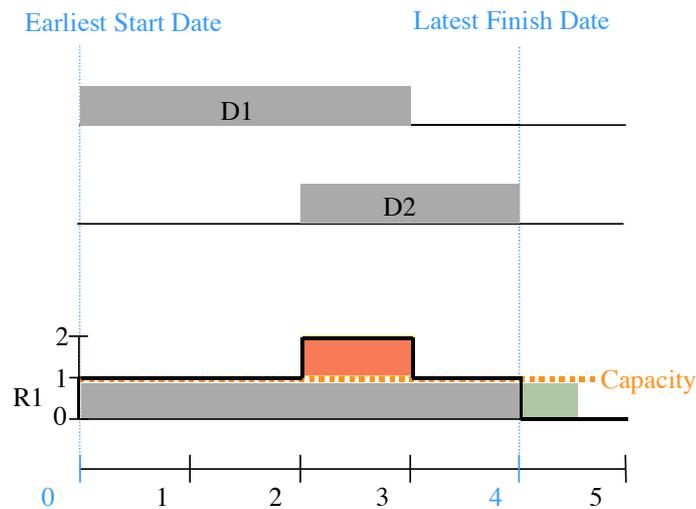


Figure 4.1: A Gantt chart showing the resource usage profile for two demands, D1 and D2, that require resource R1. D1 requires R1 from time 0 to time 3 and D2 requires R1 from time 2 to time 4. The dotted orange line shows the resource capacity. Note that the scheduler has relaxed the resource quantity constraint, rather than the more usual choice to relax the Latest Finish.

- the rather sparse prior work on visualizations to support user-guided scheduling and other aspects of planning and scheduling.

## 4.2 Visualization and Requirements Analysis

Fundamental to the task of requirements analysis is an ability to visualize demand for resources over time in relation to available resource capacity. Current schedule visualization tools provide this kind of support, but only in a rather limited way. The most widely used display is the Gantt Chart, which is designed to show resource usage over time for a fully fleshed out schedule. Figure 4.1 identifies a period of resource over-commitment (orange) where two demands require use of a single resource. Gantt charts also make clear any resource capacity that remains unused (green).

In situations of resource conflict, a common scheduling strategy is to relax the temporal restrictions of various demands and minimize lateness. In this case, an analyst may want

to visualize the lateness. Closure Graphs are commonly used for this purpose in scheduling domains that involve the transport of large amounts of cargo. A Closure Graph tracks the cumulative amount of cargo that is scheduled to be delivered in relation to the required delivery dates. In Figure 2, the required and delivered curves increase in steps as each deadline passes and as each demand is delivered. For instance, the red demand curve jumps from 0 to 761 tons at time 4, and then to 2258 tons at time 6. Therefore 761 tons were due at time 4, and  $2258-761=1497$  more tons were due at time 6. The black curve shows that by time 4 a total of 1680 tons will have been delivered (264 tons at time 2, 313 tons at time 3, and 1103 tons at time 4). The difference in height between the two curves at time 4 shows that at least  $1680-761=919$  tons are being delivered ahead of schedule. On the other hand, if the black scheduled curve is *lower* than red requirements curve at a given time, then less cargo will have been delivered by that time than is required. The difference in height is a lower bound on the number of tons of cargo required by that date that will be late. In Figure 4.2, there will lateness by the 8 th-12th days and by the 18 th-19th days.

Closure Graphs are natural for finite tasks where it makes sense to talk about total cumulative demand. For continuous scheduling tasks like job-shop, boundary conditions can be imposed on the tasks to transform them into finite tasks. For instance, the demands could be those that result from firm unfilled orders. The scheduler would simulate the transitioning from this finite set of demands to future demands by reducing the available resources over time, until none are available for the firm orders. Using this transformation, Closure Graphs as well as the 3D techniques presented later can be utilized for both kinds of task.

In current practice, requirements analysis is typically performed through iterative generation and diagnosis of complete schedules. Given a generated schedule, displays such as these are used to identify problems and deficiencies. Input requirements are then adjusted in an attempt to improve the result, and the scheduler is rerun to see if the changes had the desired effect. This process is inherently inefficient and scheduler specific. For any given conflict, the analyst must determine whether the conflict is a fundamental inconsistency in problem specifications or simply a consequence of flaws (or incompleteness) in the scheduler's decision process. In domains of any complexity, diagnosis requires the teasing apart of many interacting constraints. The level at which analysis is performed is simply too detailed to proceed effectively.

Considering basic schedule visualization tools from the standpoint of requirements analysis, there is an analogous mismatch. These tools are designed principally as mechanisms for

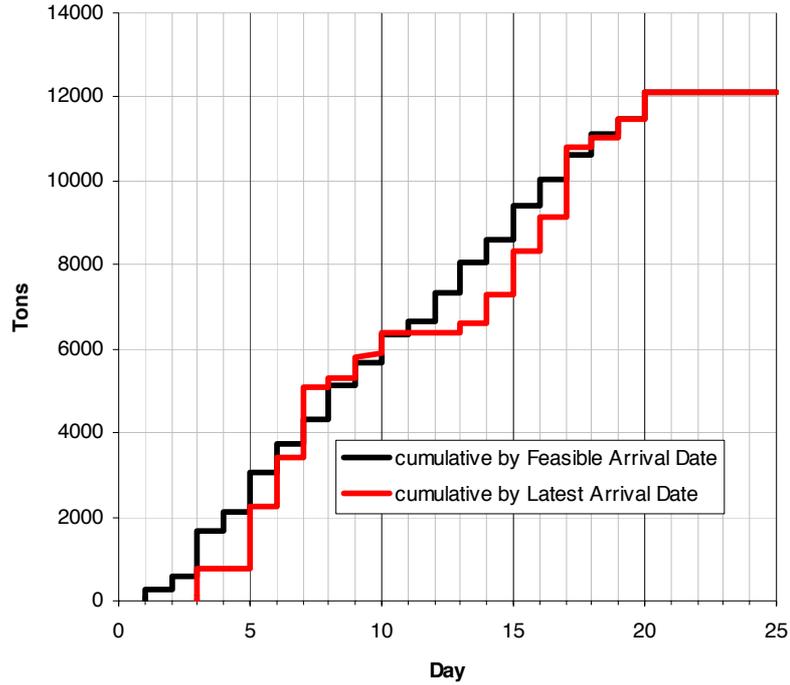


Figure 4.2: A Closure Graph showing lateness from day 8-10 and after 17. The red curve shows cumulative cargo requirements; black shows cumulative scheduled deliveries.

displaying individual solutions (i.e., complete schedules), and are not necessarily well suited to conveying fundamental constraint interactions and conflicts. For example, since temporal constraints on demands specify intervals in which they must be satisfied, visualization of this constraint space requires visualization of intervals (as opposed to points).

Attempts to generalize Gantt Charts so they can visualize this space will inevitably lose information, because a Gantt chart, by definition, shows the instantaneous demand at every time point. Consider the straw man generalization of the Gantt chart depicted in Figure 4.3, which incorporates temporal constraints instead of actual scheduled times. Rather than showing a single demand curve, an upper and lower bound is computed. The demand profiles are swept across the interval from earliest start time to latest finish time. The maximum demand at any point is

$$Upperbound(t) = \sum_{d \in demands} (MAX_{t1: EST(d) < t1 < LFT(d) - duration(d)} Quantity(d, t - t1))$$

where EST is the earliest start time, LFT is the latest finish time, duration is the width

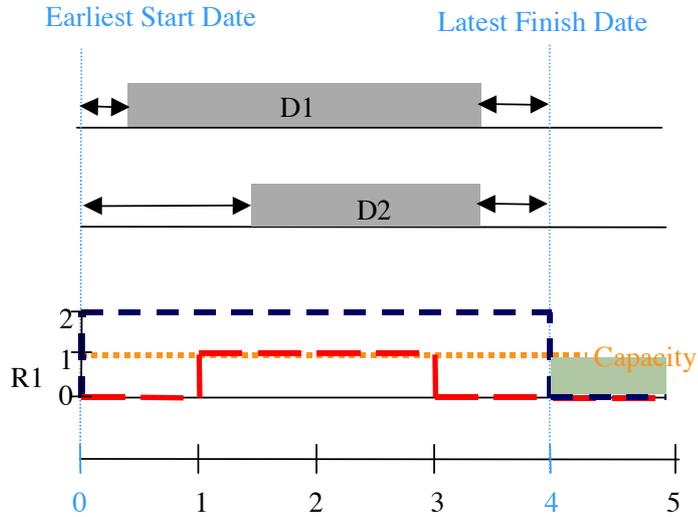


Figure 4.3: Generalized Gantt Chart showing some of the periods of excess capacity (green shading), but no shortfalls. The red dashed line shows the minimum demand for the resource over time for any schedule that respects the Earliest Start, Latest Finish, and demand quantity constraints, but possibly relaxes the resource quantity constraint. The dark blue dashed line shows the maximum demand for the resource across all these schedules.

of the demand, and Quantity is the height of the demand. The lower bound is computed analogously as

$$Lowerbound(t) = \sum_{d \in demands} (MIN_{ti: EST(d) < t1 < LFT(d) - duration(d)} Quantity(d, t - t1))$$

Requirements (demands) are necessarily infeasible if the minimum curve (red dashed line in Figure 4.3) exceeds the capacity curve (orange dotted line) at any time point. There is inevitable wasted capacity if the capacity curve exceeds the maximum (blue dashed line) at any point. In Figure 4.3, the infeasibility of supporting both D1 and D2 within their specified constraints goes undetected. Similarly, if only D1 were present as a requirement, the visualization would not indicate any of the excess capacity. A person could reason that any choice for scheduling a 3-day task in a 4-day interval leaves one day of unused capacity.

These dual shortcomings are eliminated by considering demand and capacity over intervals. Since specifying an interval requires two numbers (a start time and an end time) and specifying capacity or demand over intervals requires a third, 3D visualizations are a natural

fit. A Closure Graph shows the cumulative demand forward in time from a given start point (i.e., the start of the schedule) to any future time point, and characterizes the demand over that corresponding interval. However, from the standpoint of requirements analysis, we are also equally interested in visualizing the demand over any subinterval. Our 3D visualization is a generalization of Closure Graphs that accomplishes this objective and shows the cumulative demand over all subintervals of the scheduling horizon. To illustrate these principles, we consider a simplified version of a deployment scheduling problem, which can be seen abstractly as a single-stage, capacity constrained scheduling problem with no externally imposed ordering constraints between demands.

### 4.3 Visual Interface

We have developed our interface in the context of our work in scheduling military airlift. In this domain, two principal sets of resources are aircraft and [air]ports. Demands are represented as *move requirements*, which specify amount and type of cargo; embarkation (origin) and debarkation (destination) ports; and several target dates. The examples in this paper center on analysis of cargo processing capacity at debarkation ports. From this perspective, the earliest start time (EST) is the earliest time that a move requirement can arrive at its debarkation port, and the latest finish time (LFT) is the time when the cargo being moved must be ready to be transported onward. We use the domain terms for these times, earliest arrival date (EAD) and latest arrival date (LAD) respectively. EADs and LADs are represented as *C-Dates*, relative date offsets from the start of the operation. C1 is the first day of the operation. In our example, the operation is a deployment from the US to Korea, and there are two debarkation ports, Osan and Kimpo. The operation takes 20 days, so C20 is the last day. EAD and LAD have a granularity of one day, which is somewhat confusing. A move requirement whose EAD is the same as its LAD actually has a 24 hour window for processing. Logically, the interval between tick marks on the x-axis would be labeled with the C-Date, and the tick marks would represent midnight. However this was difficult in Excel, so figures other than screenshots label the *end* of each C-Date. For clarity, the curves in the figures have been extended to C25 so they turn horizontal.

Due to poorly integrated military planning tools, initial resource allocations are often grossly out of proportion to demands, even ignoring constraints on port assignment and EAD/LAD. Simply multiplying the capacity of all the ports by the length of the operation

may yield a total inadequate to handle the amount of material that must be moved. Until requirements are brought into balance at this gross level, there is little point in running a detailed scheduler. This is the requirements analysis task that we wish to support.

We begin with the capacity-based version of a Closure Graph visualization (Figure 4.4). For an individual port, or for a set of ports as here, the cumulative tonnage that is required to have been processed by a given day is plotted against the cumulative capacity for the same interval. We use a linear port model parameterized as the number of tons it can process in a day. The heights of these two curves at C20 represent the two totals mentioned above the total demand and capacity for the operation. However, even if the overall capacity is adequate, the Closure Graph offers insight into whether the temporal constraints imposed by the EADs and LADs preclude a feasible schedule. If at any point on the Closure Graph the demand curve exceeds the capacity curve, there can be no feasible schedule. The difference in height represents a lower bound on the amount that will inevitably be late at this point in any possible schedule. The condition where demand outstrips supply is called a shortfall. By looking at capacity instead of the amount transported in a particular schedule our findings are more general and more easily interpretable. By looking at each constraint separately we eliminated the need for the debugging process of identifying which constraint is the bottleneck causing the shortfall.

A move requirement can be represented as a rectangle whose height represents the number of tons to be moved<sup>2</sup>, and whose left and right edges show its EAD and LAD, as shown in Figure 4.5(a). One rectangle shows that 1000 tons are due between C3-C4, and the other shows that 2000 tons are due any time on day C7. Considering both move requirements, 3000 tons are due by day C7. This computation can be done graphically, as shown in Figure 4.5(b). The move requirements are sorted by LAD and placed such that the bottom of the rectangle is aligned with the top of the previous one. Then the demand curve is formed by the bottom and right edges of the rectangles. Figure 4.5(c) shows the result for all the move requirements.

Visualizing the individual move requirements along with the cumulative quantities grounds the logistical analysis in terms that have operational significance. A scheduler can discuss with a commander particular sets of move requirements that are incompatible with the capacity constraints. For instance, if move requirements are color-coded by the type of unit

---

<sup>2</sup>This rectangle representation is different from that used in Figures 4.1 and 4.3, where the area represents the demand quantity, e.g. in ton-days. Here the height represents the demand quantity, e.g. in tons.

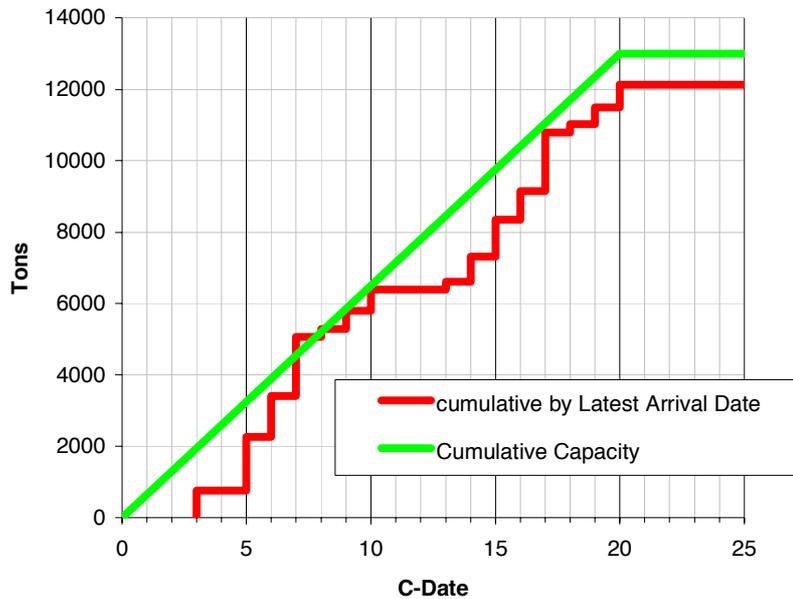


Figure 4.4: Capacity-based Closure Graph. The green line shows cumulative port capacity for both ports over the 20-day transportation plan. As in Figure 2, the red curve shows cumulative demand. Demand exceeds supply over  $[0, 8]$  and  $[0, 18]$ .

that owns the equipment, the scheduler may be able to eyeball quick summaries like you can't get both the engineering company and the Patriot battery to 90% by C8. Grounding the visualization in the concrete and familiar demand representation for the type of scheduling application (here move requirements) may also make the abstract interval-based constraint representations more learnable.

By computing the demand curve from the right edges of the rectangles, we're making the optimistic assumption that we don't have to worry about a move requirement until the moment of its LAD. In reality, the processing of a move requirement will take some amount of time; hence, processing must start some time before the LAD. Later we will generalize the visualization to take this into account, but for now we ignore all constraints on the time course of the processing to obtain a simpler computation. This corresponds to solving a relaxed version of the problem, which instead computes a lower bound on demand. This relaxed problem has previously been termed the “Fully Elastic Cumulative Scheduling Problem” [Baptiste *et al.*, 1999]. It has an efficient solution and thus is well suited for interactive requirements analysis. We discuss alternatives in the section 4.6 below.

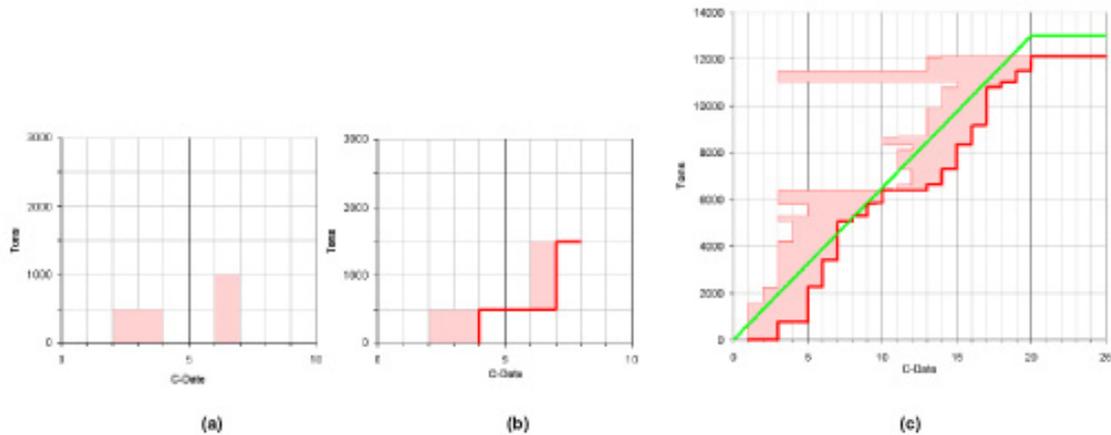


Figure 4.5: Visual computation of demand curve (red) from sorted and stacked move requirements (pink rectangles).

The demand curve loses some information, because it ignores the left edges. Thus, the Closure Graph fails to show all inevitable shortfalls. In effect, it assumes all cargo is available for processing on day C1, and it would show all inevitable shortfalls [in the relaxed model] if that were true. As an example, let's look at how this visualization fails for the data in Figure 5. There are two fairly distinct groups of demand rectangles: Many requirements fall within the interval from C1 to C10 (call it Phase 1), while many others are between C11 and C20 (Phase 2). The Closure Graph does a reasonable job at analyzing Phase 1. The only shortfall during this interval is a tiny one ending at C7, and the two curves are indeed very close at C7. On the other hand, the graph does a poor job at conveying the potential problems associated with Phase 2.

Figure 4.6 shows a modified Closure Graph that considers Phase 2 as a separate scheduling problem. It is obtained by ignoring all of the requirements except those entirely within Phase 2, and allocating all of the port capacity during this period to these demands. A small shortfall can be seen at C17 that was missed in the two previous figures. The problem is that little cargo is available for processing on C10, so the port is only partially utilized. If some of the cargo that becomes available on C11 could be made available earlier, it could avoid the lateness problem seen at C17.

Figure 4.7 shows an interactive visualization that allows analysts to see a Capacity-based Closure Graph for any interval. In order to specify subintervals, analysts can drag curtains

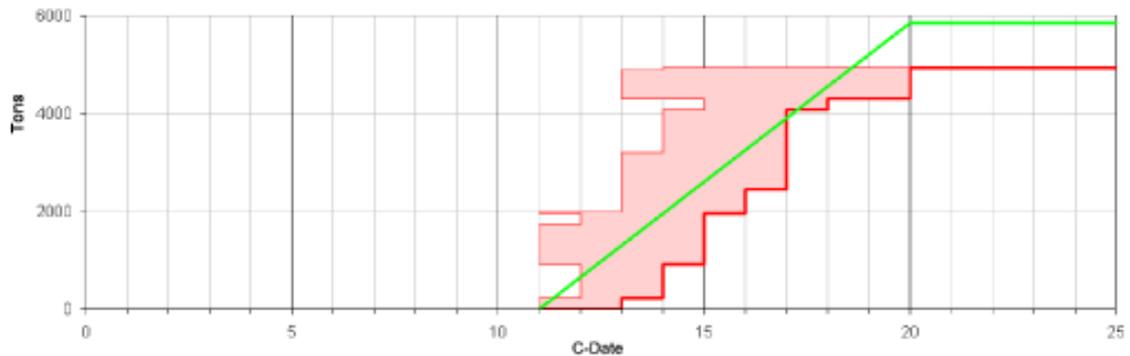


Figure 4.6: Subset of move requirements with  $EAD \geq C12$ .

from the left and right to block out the unwanted days. In the figure, the analyst drags the left curtain to C1 (a) and then to C5 (b). When viewing the interval C2-C20 (a), the appearance is almost identical to the Capacity-based Closure Graph of Figure 5, though revealing a larger shortfall at C7. In (c) the analyst drags the left curtain to C11 and the right curtain to C17 and is now looking at the interval C12-C17. At this point, the same interval as displayed in Figure 6 has been isolated. As the curtains are moved, only those move requirements that fall entirely within the current interval are displayed in color. The rest are grayed out. This performs the same filtering that was done in Figure 6. Leaving grayed-out rectangles in place rather than removing them and compressing the stack (as in Figure 6) allows the analyst to estimate the effect of moving the curtains. For instance, it is clear that moving the left curtain to C10 is a good place to separate Phase 2 from Phase 1, because only one move requirement (which spans almost the whole plan) crosses this boundary.

The cumulative demand and capacity totals over the isolated interval are shown separately as a red bar and a green line to the left of the Closure Graph. For example, in Figure 4.7(c), these are the heights of the red and green curves at C17. The text label shows that the precise amount of the shortfall over the interval C12-C17 is 109 tons.

By dragging the left and right curtains to inspect all possible subintervals from C1 to C20, the analyst can find all periods of inevitable shortfall in port capacity. However, this is not a particularly convenient approach. The analyst is interested in adjusting constraints to reconcile mismatches in demands and resource capacity. It is distracting and tedious to have to attend to and manipulate this sort of display parameter to uncover problems. In fact, when used to this end, the visualization is being treated as a three dimensional display: quantity

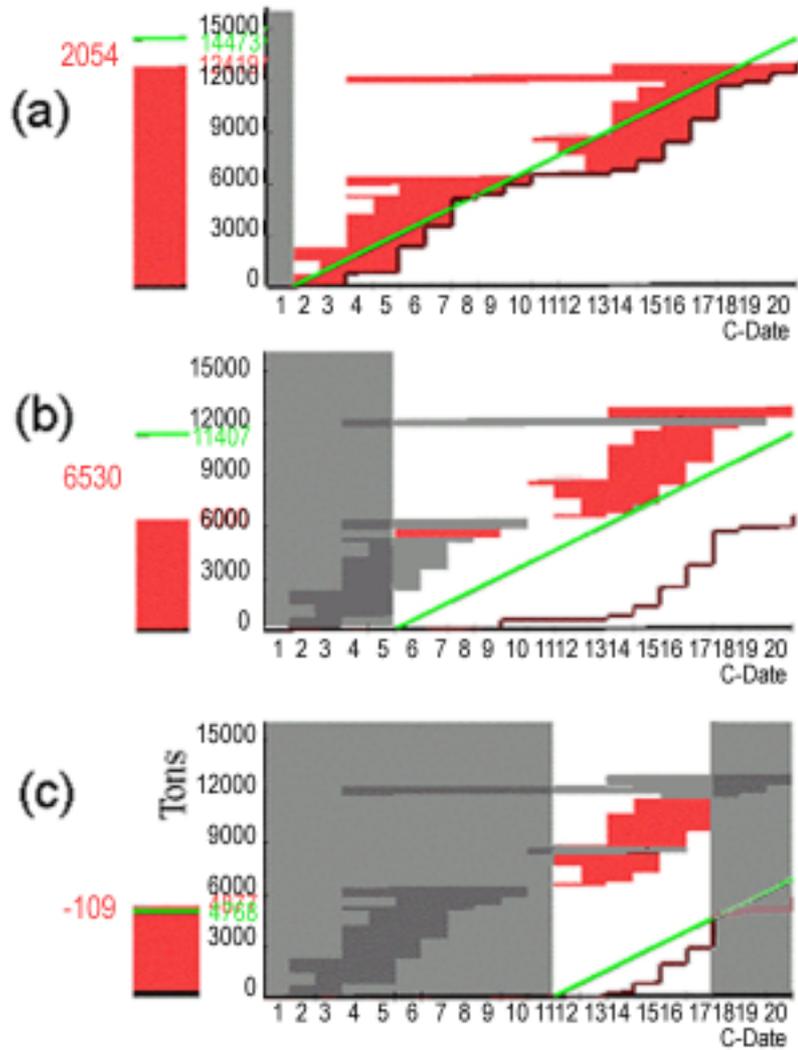


Figure 4.7: Interactive visualization using a C-Date window to filter the move requirements with a time window [C2-C20], [C6-C20], and [C12-C17] (a c) . To the left of the chart are shown the total capacity (vertical red bar) and demand (horizontal green line) over the time window. The bar and line are labeled on the right with their absolute y-coordinates and on the left with their difference. For instance, (c) shows that the total demand over [C12-C17] is 4877 tons, the total capacity is 4768 tons, and the difference is 109 tons.

using y; interval end using x; and interval start using animation (time). It requires much less attention to use a visualization where all three variables are encoded spatially.

Figure 4.8 shows such a visualization. Conceptually, the visualization is composed by computing a cumulative demand bar, like the ones on the left in Figure 4.7, for every possible subinterval and arranging them as columns on a two dimensional grid by their start and end dates. In practice, we found the visualization easier to use when a variable transformation was applied, so the x-axis shows interval midpoint, y shows quantity, and z shows interval length (b). That way the analyst can think about x as later in the plan, and z as less constrained, rather than the more abstract interval start and interval end. Since the time quantum used by the move requirements is 1 day, the requirements form columns whose xz planar dimensions are 1 day. They are diamond-shaped rather than axis-aligned due to the transformation of variables. The two phases of the demands form two mountain ranges that meet and combine additively for the longest intervals, which include both phases. In the fully elastic model, the demand surface increases in discrete jumps at each demand EAD/LAD interval. Using more realistic models it would be continuous. The two dark red curves drawn by hand in (b) show how a slice through the capacity surface corresponds to a 2D Capacity-based Closure Graph for a given start date. In terms of the original variables, the equation relating START and END for a slice is just  $START = \text{constant}$ . In the transformed space this equation becomes  $LENGTH = (\text{MIDPOINT} - \text{constant}) * 2$ . Thus the slope of the slices in the xz plane is 2. Vertical slices in the other diagonal (with slope = -2) represent intervals with a common end point. All plan intervals lie within a triangular region defined by the line with slope 2 representing  $\int \text{interval start}_i = C1$  and the line with slope -2 representing  $\int \text{interval end}_i = C20$ .

Using a linear model of port capacity, the capacity curve is a plane whose height is proportional to interval length (a). The capacity curve can be thought of as a cloud layer. Any bright red peaks that jut above the partially transparent green cloud layer are easy to spot at a glance (c). Their xz coordinate indicates the shortfall interval. Their height above the cloud layer indicates the shortfall magnitude. In this case, there are small (relative to the bar heights) shortfalls over the intervals C1-C7 and C11-C17. These correspond to the two intervals already found using the 2D interface (ending at C7 in Figure 4.7(a) and C17 in Figure 4.7(c).

For n move requirements the minimum surface can be calculated in  $O(n^2)$  time:

1. Find all distinct EADs and LADs, and sort them temporally. Each pair of successive

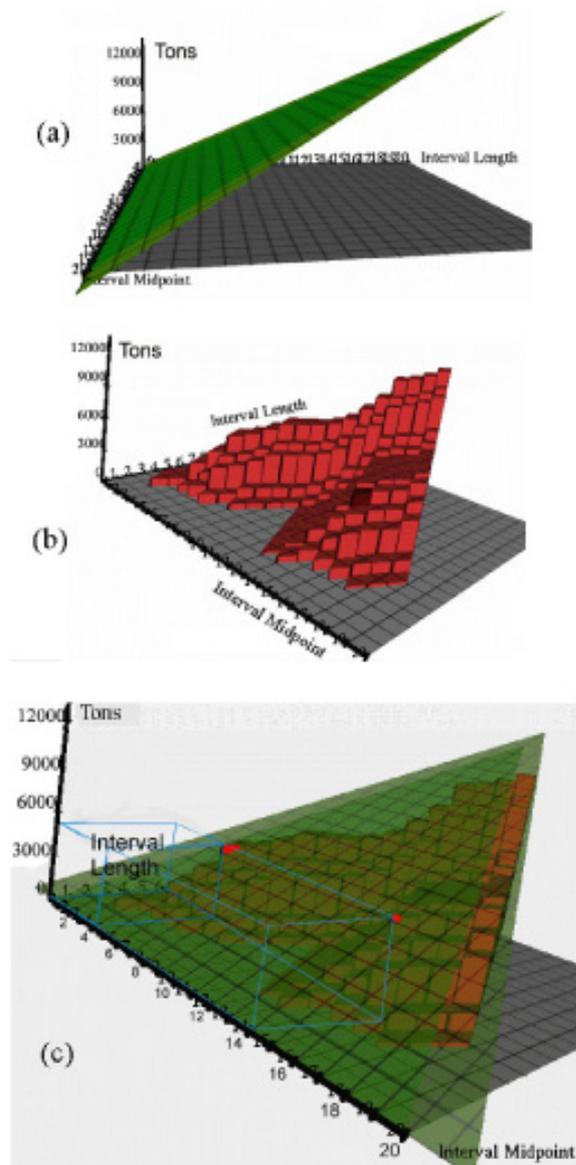


Figure 4.8: The 3D visualization superimposes the green planar capacity surface (a) over the red demand surface (b). Two demand curves, for intervals starting with C1 and C12, have been drawn on top of the demand surface. These two slices show the same information as in Figures 4.5 and 4.6 respectively. In (c) the demand surface breaks through the capacity surface at two points. Wire-frames have been drawn to facilitate reading the coordinates. These intervals have centers at C4 and C14. The lengths are both six. Hence, the intervals are from C1-C7 and C11-C17. An animated computation of the surfaces is available at <http://www.cs.cmu.edu/sage/animations/LTVembedded.ppt> (best to save a file locally rather than run inside a browser).

dates forms a leaf interval.

2. Form the lattice of all  $O(n^2)$  enclosing intervals
3. For each move requirement, add its quantity to the lattice node representing its EAD-LAD interval.
4. Make a pass upward through the lattice accumulating quantities from enclosed intervals. Each node computes its total demand as

$$Q(s, e) = Q_0(s, e) + Q(s + 1, e) + Q(s, e - 1) - Q(s + 1, e - 1)$$

where  $(s, e)$  denotes the start and end points of its interval and  $Q_0$  is the sum from step 3. The subtraction is required because the quantity for an interval is propagated in a wedge upward through the lattice, so that it contributes to both the left and right children of a properly contained interval.

## 4.4 Example Analysis

Lets assume that the small shortfalls over the entire set of ports are not significant enough to warrant changing the requirements. The next step is to break out more constraints and look at them in isolation. In Figure 4.9 the analyst has created two copies of the visualization tool and focused one on each port. It is immediately clear from the height of the red peaks that the port demands are unbalanced; Osan is over-committed while Kimpo is under-utilized. Yellow ovals call attention to the three groups of intervals where the red demand surface rises above the green capacity plane for Osan: one group from 4-8 days long in Phase 1, a second group from 4-8 days long in Phase 2, and a group from 18-20 days long that spans both phases. The shortfall over the entire 20-day plan interval suggests that there is little point in relaxing temporal requirements for the Osan demands, because improvements in some intervals would worsen others. The two Korean destination ports are relatively close; hence, re-routing of move requirements appears to be a viable strategy for minimizing the identified shortfalls. Since we already found small shortfalls in each phase considering all the ports together, we know that re-routing cannot completely solve all capacity problems, but it should significantly reduce them.

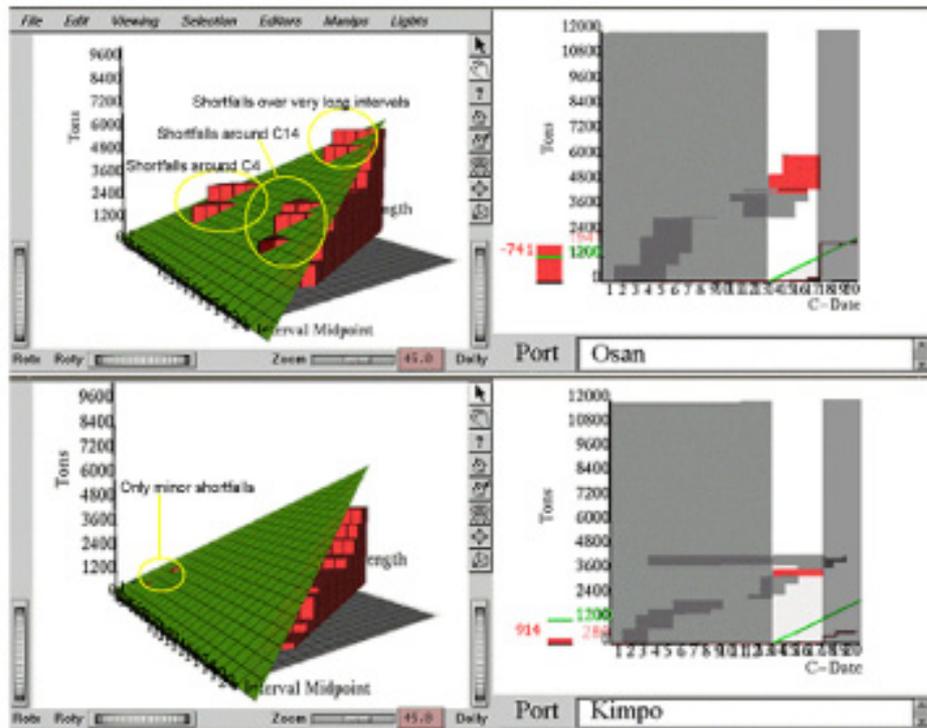


Figure 4.9: The labeled yellow ovals superimposed on the screen shot show that Osan has three significant groups of red shortfall peaks (top) while Kimpo has only one small shortfall (bottom). The gap between the red and green surfaces at the apex of the triangle for Kimpo shows that only about 3/4 of its capacity is used overall.

#### 4.4.1 Shortfall Analysis

A good heuristic is to fix shortfalls over the most constrained (shortest) intervals first. By clicking on the C13-C17 shortfall in the Osan 3D visualization, that interval becomes selected in all six visualizations (1D, 2D, and 3D for each port). The 1D bars and explicit labels provide the easiest view for visually or symbolically computing whether the shortfall at Osan (741 tons) can be entirely absorbed by the excess capacity at Kimpo (914 tons). Thus for this interval, re-routing can eliminate the shortfall.

Analysts must then decide which move requirements can be re-routed without significantly disrupting the operation. Their decision is based on domain knowledge not included in the scheduling requirements: the type of cargo, the military units that own the cargo, their role and mobility, and so on. For this task, the visualizations shown in this paper are inadequate. We envision an integrated interface environment where analysts can drag and drop among multiple coordinated visualization tools. This is discussed further in section 4.5 below.

Figure 4.10 shows the effect of re-routing 835 tons of the move requirements with EAD13 and LAD17 from Osan to Kimpo. The Osan shortfall is indeed eliminated for C13-C17, and nearly so over all of Phase 2. The only remaining Phase 2 shortfalls are over the intervals C10-C17 and C11-C17 at Osan, and C11-C18, C11-C17, C12-C17, and C12-C18 at Kimpo. Only C11-C17 is an overall shortfall over both ports, so further re-routing could eliminate all the other shortfall intervals.

Since only move requirements whose EAD/LAD fell in the interval C13-C17 were changed, this update had no effect on the shortfalls during Phase 1. Of the three groups of shortfall intervals at Osan in Figure 4.9, the 4-8 day Phase 1 group therefore remains unchanged in Figure 4.10. The group for 18-20 day intervals shrinks considerably, because all these intervals contain the C13-C17 interval. There is in fact much information contained in the visualizations about the relationships among shortfall intervals. The spatial layout makes interval containment apparent [Kulpa, 1997], so we could have predicted the qualitative effect of reducing the Phase 2 shortfalls on those of the long intervals. If we had worked on the long intervals first, we might have reduced the shortfalls in ways that don't help the two shorter intervals; therefore, it is best to start with the short ones. Even as analysts work on the short ones, though, the realization that there are overall shortfalls for intervals that contain these subintervals influences their solution strategy. In this example, it suggested that changing EADs and LADs to fix one interval might just make another one worse, and that re-routing

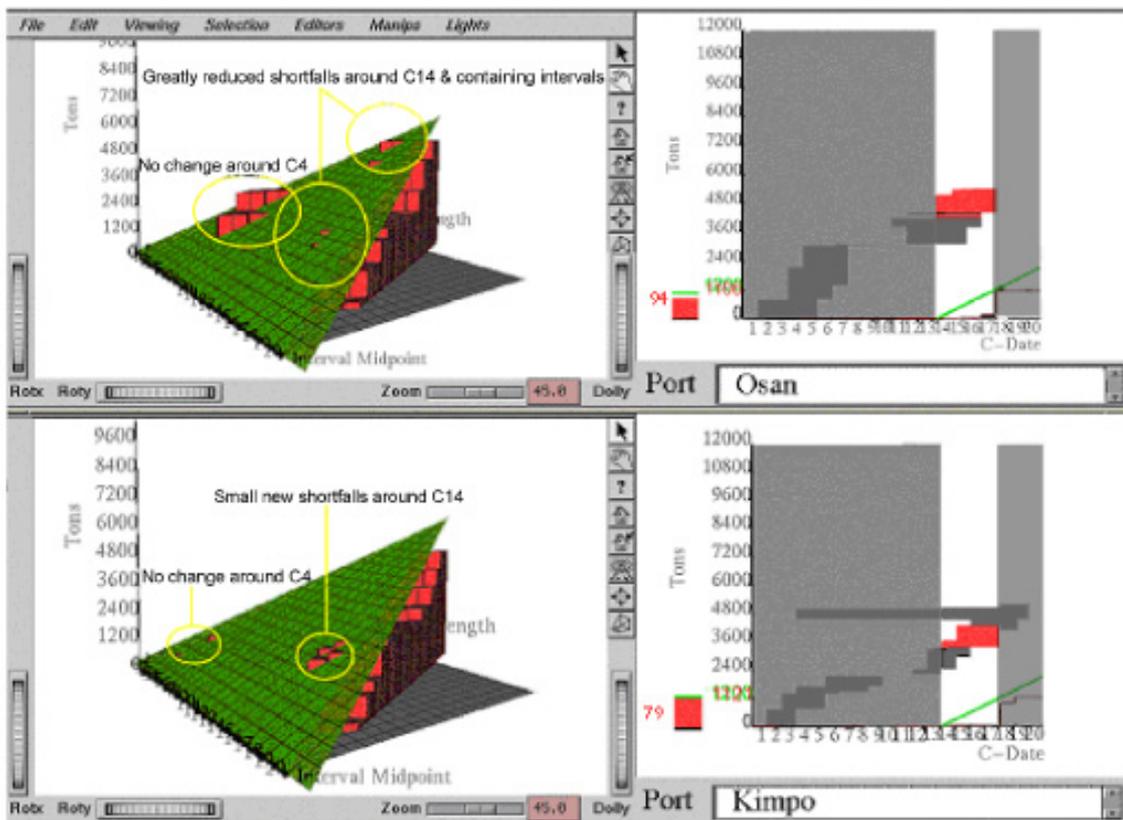


Figure 4.10: After re-routing, Phase 2 shortfalls are largely eliminated.

might be a better avenue to pursue.

We call this the Wack-a-Mole interface because of the resemblance to the amusement parlor game, where wacking down in one place is followed by popping up in another place.

#### 4.4.2 Excess Analysis

Above we insisted on finding a lower bound on demand in the relaxed model so that any problems apparent in the interface necessarily doom any attempted scheduling algorithm to failure. It is also useful to consider the upper bound on demand over each interval. When restricting attention to a particular subinterval, the computation includes all move requirements with an *overlapping* EAD/LAD interval, rather than those that are fully contained in it. The blue surface, curve, and bar in Figure 4.11 show the result. If the maximum demand is lower than the capacity, it means there is excess capacity that can't be of use in any possible schedule. The 3D representation shows a canyon centered on C10 for intervals from 0 to 9 days long where the blue demand surface disappears below the capacity plane. Considering that there are shortfalls at this same port during intervals centered on C4 (see Figure 4.10), it would certainly be helpful to delay due dates to take advantage of the excess capacity. Armed with this knowledge, the analyst must decide which move requirements are good candidates for delay, and then update their LAD until the shortfalls are minimized. Currently analysts in this domain do not attempt to find excess capacity. If it can be effortlessly seen in visualizations, there could be more effective collaboration among analysts working on different operations that use some of the same resources.

The min and max demand necessarily converge over the entire plan interval, which by definition contains all move requirements. The 2D curves can clearly be seen to converge at C20. From a side view like Figures 4.9 and 4.10 the convergence would be clear in the 3D view as well. In the top view of Figure 4.11 it can be deduced from the fact that the tip of the triangle is red. Note that there are two other groups of red intervals, which shows that at Osan the two phases of the plan are completely separated. The first reaches closure before the second starts, so there is no uncertainty in the amount transported over either of these intervals. The divergence of the curves/surfaces for shorter intervals is due to the uncertainty in when each move requirement will be processed within its EAD/LAD interval.

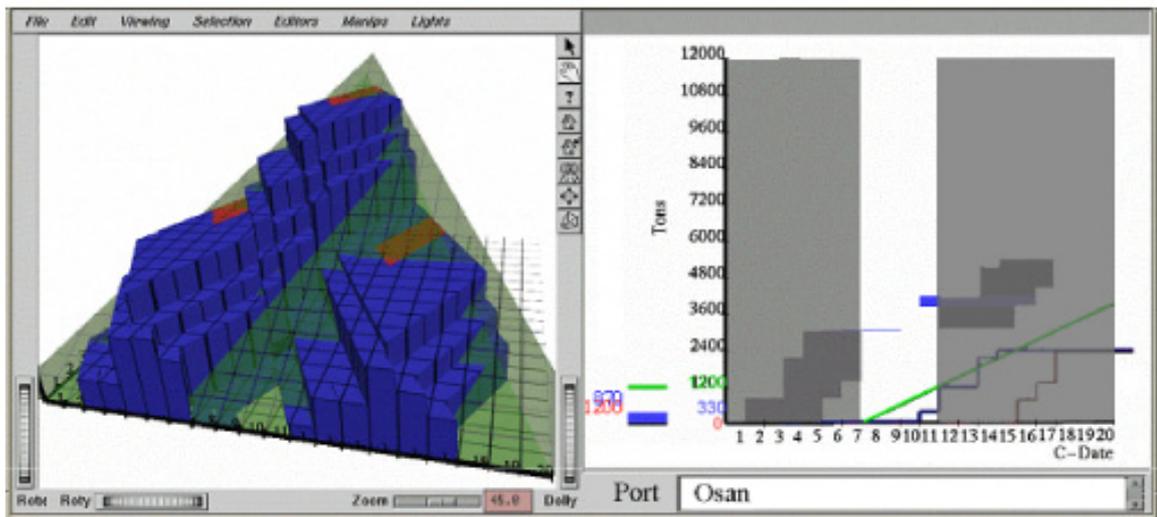


Figure 4.11: The blue surface (left) shows the maximum demand that any plan could possibly require of a resource, for all intervals. The blue curve and the blue bar (right) show the maximum demand over the single interval C7 C11. Comparing them with the green capacity curve and bar shows that there are 870 tons of excess capacity over this interval. Therefore delaying move requirements of Phase 1 or advancing those of Phase 2 are promising strategies.

## 4.5 Integration with Information Analysis

The interface shown in the figures is implemented in Inventor running on an SGI Reality workstation. We are currently porting it to Java3D, which will allow drag and drop interaction with a companion tool called Visage. Visage is an information-centric [Roth *et al.*, 1997] user interface environment for data exploration and for creating interfaces to data-intensive applications. It is being developed by Carnegie Mellon University and Maya Design Group, and runs on Windows and Unix/X workstations. Domain data objects are represented as first class interface objects that can be manipulated using a common set of basic operations, such as drill-down and roll-up, drag-and-drop, copy, Dynamic Query filtering, and dynamic scaling. These operations are universally applicable across the environment, whether graphical objects appear in a hierarchical table, a map, a slide show, a query, or other application user interface. However, Visage is limited to 2D visualizations. Visage includes an API to access ODBC databases, allowing easy coordination with other components of an exploratory data analysis environment. For instance, graphical objects can be dragged across application UI boundaries. Thus analysts will see no distinction between the Java3D component and the rest of Visage.

With the powerful data exploration operations provided by Visage, the Java3D version will allow the kind of integrated analysis suggested in the previous section. The example below works in Visage now. The only missing link is the integration.

To examine the move requirements that contribute to the shortfall at Osan from C13-C17 in more detail, the analyst would drag the red column representing this interval in Figure 9 into a Visage. Figure 4.12 illustrates the use of Visages Outliner drill-down table for this task. First the set of move requirements is partitioned by port of debarkation to isolate those routed to Osan. Then those are partitioned by the type of cargo, in order to pick those that can be most easily transported on the ground. Bulk is the default cargo type when there are no special requirements for transport. In this case, there are twelve move requirements totaling 1200 tons that may be good candidates for re-routing. After editing the port of debarkation in Visage, the Java interface would update, resulting in Figure 4.10.

In order to take advantage of the unused capacity revealed in Figure 4.11, a different Visage interface might be used. In Figure 4.13, the analyst has dropped the set of C1-C6 move requirements on an interval chart showing their EAD/LAD interval. Color encodes the mission the unit that owns the cargo. If some of the missions, perhaps support, are

Outliner		
	quantity	<None>
C13-C17 Move_Requirements	---	
>by port_of_debarkation	---	
for KIMPO-INTL	118	
for OSAN-AB	193	
>by cargo_type	---	
for PAX	61	
for OUTSIZE	39	
for OVERSIZE	81	
for BULK	12	
Move_Requirement_16_20_144	144	
Move_Requirement_16_20_153	153	
Move_Requirement_18_21_157	157	
Move_Requirement_18_21_55	55	
Move_Requirement_18_21_81	81	
Move_Requirement_19_22_19	19	
Move_Requirement_26_30_144	144	
Move_Requirement_26_30_153	153	
Move_Requirement_26_31_55	55	
Move_Requirement_28_31_157	157	
Move_Requirement_28_31_81	81	
Move_Requirement_29_32_19	19	

Figure 4.12: The top line of the table represents the set of move requirements that would be dragged in from the Java interface. Successive drill-downs partition this set by port and then, for the Osan move requirements, by the type of cargo.

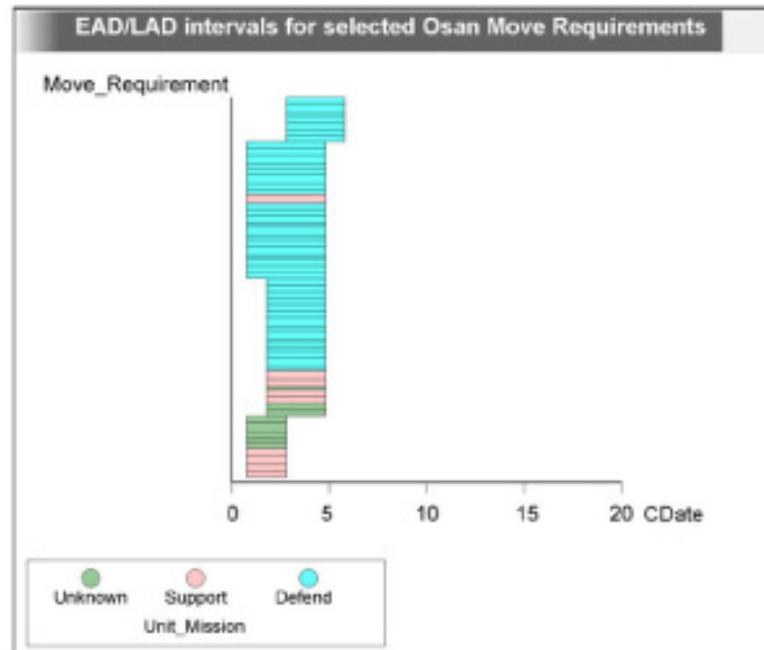


Figure 4.13: Visage visualization of all the move requirements that comprise the C1-C6 interval bar in Figure 11, showing their EAD/LAD interval and the mission of the unit that owns the cargo. Either endpoint of any of the bars can be dragged to update the EAD or LAD for that move requirement. Multiple move requirements can be edited simultaneously using selection prior to dragging.

lower priority they would be candidates for delay. Here the EAD and LAD can be edited by dragging the intervals (or interval edges) within the chart. After each mouse move, the Java interface would update. This continuous feedback would make it easy to delay just the right amount without having to do mental arithmetic.

## 4.6 Other Resource Models

Our logistics planning example has relied on a simple model of resources and resource usage. In this model, available capacity is expressed in terms of a quantity per time unit (e.g., tons per day), and demands are expressed as quantities that are required over specific time intervals (e.g., EAD to LAD). Within the designated time interval that a given demand re-

quires capacity, we assumed a fully-elastic processing model. Under this model, there are no constraints on the time course of processing. The entire processing of a given demand can occur at any instantaneous point within its required interval. Allowing for such impulse functions produces the unrealistic discrete steps in the requirements surface.

There are other, stronger problem relaxations that also can be solved efficiently, most notably the Partially Elastic Cumulative Scheduling Problem analyzed in [Baptiste *et al.*, 1999]. This relaxation could be directly substituted for the fully elastic problem model used in the preceding sections. For ease of presentation, we chose not to.

Perhaps a more commonly used model of resource usage is one where both the amount of capacity required by a demand and the duration for which the capacity is required are fixed. Based on the shape of the demands in a Gantt Chart, we call these rectangular demand models. Since resource usage can no longer include impulse functions, its cumulative (integral) value must vary continuously with interval length and midpoint.

In the elastic case, demand is always zero for degenerate intervals so the relationship between demand and capacity at the base of the triangle where  $z = 0$  is not important. In the rectangular case, we want to compare demand and capacity at each point in time as well as over longer intervals. In the interface shown in Figures 4.8 - 4.10, both go to zero and can't be visually compared. We would therefore plot the demand and capacity *rates*, dividing the heights by the interval lengths. Then the capacity plane would be horizontal instead of sloped. We tried this variation for the elastic case, but found it more helpful to see the absolute amount of shortfalls.

With this modification, the  $z = 0$  slice through the demand and capacity curves shows the same instantaneous 2D demand curves that are used to determine resource contention peaks in so-called “profile-based” scheduling algorithms (discussed below in the next section). However, the extended computation of lower and upper bounds over all possible intervals will show additional shortfalls in situations like Figure 4.10, and will also indicate how resource contention intervals relate to one another over longer intervals. For each move requirement and each interval, we can find the offset from its EAD that generates the maximum and minimum intersection of the actual demand profile with the interval. Adding these bounds for all move requirements generates the heights of the surfaces for each interval. The analysts experience of the interface remains the same. Note also that it is no longer guaranteed (by itself) to uncover all possible conflicts. Ensuring a feasible solution will typically require additional scheduling decisions.

## 4.7 User-Guided Scheduling

In practice, the line between requirements analysis and scheduling is not a crisp one. Thus far we have been promoting the use of our interface as a means for reconciling conflicting requirements before scheduling and expediting the process of obtaining an acceptable solution. We do not expect that its use will eliminate the need for human intervention and analysis during the scheduling process. There are two reasons. First, no formal scheduling model ever captures all real world requirements. The analyst must enforce these unmodeled constraints. Second, even if a feasible schedule exists, finding one in most practical domains is NP-hard and heuristic schedulers may not be able to find an acceptable one unaided. Perhaps a better way to see the switch from requirements analysis to scheduling is as a step-wise transition to a richer (i.e., higher fidelity) model that incorporates a more precise and more complete accounting of domain constraints. More generally, one could imagine a hierarchy of progressively less relaxed models that eventually bottoms out at the scheduler's base model. When viewed this way, it seems natural to expect the same sort of interactive support and constraint manipulation capabilities to be available during detailed scheduling.

To see how our interface might extend to support a more detailed scheduling process, let's reconsider its use during the requirements analysis stage of problem solving. Early on, resources and demands will tend to be so poorly matched that even simple relaxed models that assume independent resources and ignore ordering dependencies will reveal shortfalls. Analysts rely on the minimum demand surface in our 3D visualization to find these shortfalls. Once these gross problems are resolved, analysts can begin to turn attention to the maximum demand surface and use it to find and manipulate intervals where there is excess capacity. On the one hand, they want the maximum surface to lie below the capacity surface to ensure there is enough excess to handle interactions and remain robust to later changes in specifications. On the other hand, they don't want it to fall too far below the capacity surface because that represents excess and wasted capacity. Therefore, the goal is to make the maximum surface approximate the capacity surface.

Profile-based scheduling algorithms [Beck, 1999, Cesta *et al.*, 1998a] are based on this idea of leveling peaks in the maximum demand curve (a 2D counterpart of the maximum demand surface). For a given peak, a *conflict set* of demands contributing to the peak is gathered. Ordering constraints are posted among these demands until the maximum curve lies completely below the capacity curve. At this point, any schedule satisfying the resulting

temporal constraint graph will be feasible. When all peaks have been leveled, a schedule can be extracted in polynomial time.

Using our interface, we can support the same problem solving procedure applied to peaks in 3D surfaces rather than 2D curves. In the simplest case, analysts can be directly involved in the basic problem solving cycle and given responsibility to interactively drive the peak-leveling process. The system would re-compute minimum and maximum demand surfaces at each step. If the minimum demand surface is ever found to exceed the capacity surface, then an infeasible state has been reached, and analysts must decide whether to modify problem constraints (change requirements) or to retract previous decisions (backtrack). Otherwise, analysts could use the maximum demand surface to first identify which peak to level, and then to decide which pair of competing demands to sequence. Computationally, this application of our 3D visualization implies a stronger form of constraint checking than is typically exploited in the base conflict analysis associated with these scheduling algorithms. In particular, the computation of conflicts over various intervals amounts to incorporating additional summation constraints and edge-finding techniques [Carlier and Pinson, 1994, Nuijten, 1994], in essence enabling earlier (visual) detection of a larger set of potential short-falls.

## 4.8 Related Work

There is a wealth of research in the area of constraint analysis and propagation that is aimed principally at defining conditions for early detection of infeasible solutions and search space pruning during automated schedule construction. Such techniques are all candidates for use in supporting interactive requirements analysis. The key is finding a good visual interpretation. Showing analysts a visualization can provide insight into the structure of the problem that goes beyond the litmus test that analytic methods provide.

Actual work in the area of visualizing planning and scheduling requirements and solution spaces is rather sparse. The early concept of **Electronic Leitstands** [Kanet and Sridharan, 1990] was aimed at providing graphical support and controls for managing production schedules in manufacturing contexts. However, in practice, this concept has translated mostly to interactive Gantt chart displays and little attention has been given to the general task of requirements analysis. In an experiment comparing interactive Gantt charts to static ones, no benefit was found [Sharit, 1985] (cited by [Eindhoven, 1997]).

**AsbruView** [Miksch *et al.*, 1998] is a 3D visualization of skeletal plans that shows a hierarchy of alternative subplans. It shows conditions, intentions, and effects of each subplan and is intended to provide physicians with an overview of clinical protocols. These goals are very different from ours. Our scheduling model ignores exactly the kind of complex interactions that AsbruView shows. Instead, we focus on aggregate demand and capacity for resources.

**IDA** is a visualization-based planning interface that is similar in spirit to our work [StAmant, 2001]. It generates abstractions, which are a form of relaxation, for analysts to constrain. For instance, the system aggregates the movement of multiple army units, allowing the analyst to give high-level guidance such as "overall there should be a retreat." However, in their planning domain individual goals are much more interdependent than in our scheduling problem. Therefore, these abstractions cannot take advantage of the simple additive models that we leverage for efficient constraint propagation and effective visualization. Indeed, St. Amant characterizes the process of mixed initiative constraint addition and propagation as exploratory data analysis, thus resembling the debugging stage of fully elaborated schedules that follows the requirements analysis process.

**IS-diagrams** [Kulpa, 1997] are 2D spatial layouts of intervals, and are the first use of the representation we use in the xz plane of our 3D interface. IS-diagrams are used to elucidate the inferences of interval algebra, and are used with two other innovative spatial representations. For instance, one can quickly find when two people have 45 minutes free at lunchtime.

## 4.9 Summary

Requirements analysis, the task of reconciling the task requirements against the resources available, requires a significant amount of human effort. Previous scheduling research is almost exclusively devoted to finding an admissible schedule given fixed resources and requirements. The tools used for detailed scheduling are also used for requirements analysis because nothing else is available. Solving relaxed versions of the scheduling problem make it easier to understand where bottlenecks are, because there is less interaction to untangle. Maintaining conflicting constraints rather than instantiating a complete schedule also simplifies debugging, because the tradeoffs are explicit. Our 3D visualization lays out all possible conflict intervals so they can be seen at a glance.

Previous visualizations of conflicts based on Gantt charts compare instantaneous usage and resource capacity. Closure Graphs show resource usage only for intervals starting at the beginning of the operation. Both of these 2D approaches fail to show complete conflict sets. By visualizing demand and capacity for every interval, our 3D interface allows the analyst to find conflicts earlier in the analysis process. In addition to showing all temporally localized conflict sets, it is easy to see whether larger temporal intervals containing multiple conflict sets are near or over capacity. This gives advance warning about the difficulty of finding a consistent schedule because imposing constraints to solve one problem may make it more difficult to solve others. As a generalization of 2D profiles, our interface is upward compatible with existing constraint-posting scheduling algorithms.

Integrating the resource utilization interface with other Visage visualizations will further broaden the class of tasks that can be performed in a unified interface environment. Using multiple coordinated displays, analysts will be able to revise requirements in ways that minimally affect operational outcomes, and reconcile situations of conflicting requirements more efficiently.

Our interface unites the tasks of requirements analysis and user-guided scheduling. Analysts interleave performance of these tasks, and there is no crisp distinction between the two. The fact that scheduling can be attacked with automated algorithms has resulted in the artificial separation that has been reflected in previous interfaces. In our interface, analysts guide the scheduling process by posting constraints that level peaks in the maximum demand surface, just as in some previously developed scheduling algorithms. However, when peaks in the minimum demand surface are found, it is not necessarily seen as a signal to backtrack. The conflict can also be addressed by modifying requirements. In our domain, minimum peaks commonly occur before any scheduling decisions have been made. In this case, revising requirements is the only alternative. In fact, getting the constraints right generally occupies more of analysts time than scheduling per se.

## Chapter 5

# Incremental Resource Allocation and Scheduling for Effects-Based Operations

**Summary:** In this chapter, we summarize a Technology Integration Experiment (TIE) carried out collaboratively with AFRL Rome to produce a "Jump-Start Demonstration" for AFRL's "Effects-based Operations Advanced Technology Demonstration program. Broadly speaking, this TIE was aimed at demonstrating technology support for compressing the air operations planning cycle, through tight inter-leaving of effects-based planning, targeting and resource allocation processes. Our starting point for participation in this effort was a System called ACS (Air Campaign Scheduler), which was initially developed under DARPA's "JFACC after Next" program. ACS is a second application system built using the Ozone Scheduling Framework, and adapts the techniques and components used in the AMC Allocator to the problem of allocating platforms and munitions to air strike missions over time. To accomplish the EBO "Jump Start" TIE, ACS was extended and integrated with two companion technologies: (1) a planning tool called CAT (Causal Analysis Tool), developed at AFRL and designed to reason about the effects of various strike actions, and (2) a targeting tool called JTT (Joint targeting Tool) used to support selection of appropriate weaponeering solutions. A small planning scenario involving the generation of approximately 400 air strike missions was successfully demonstrated.

## 5.1 Introduction

The concept of “Effects-Based Operations” (EBO) aims at shifting the driving focus of air operations planning. Whereas air campaign planning has historically proceeded from a target-centric perspective, EBO advocates more explicit reasoning about what effects can best achieve overall air campaign goals, and then back from these determined desirable effects to the selection of appropriate actions (targets). One implication of a shift to an effects-based planning process is the need for a much tighter integration with the current execution state. Since the effects of actions (intended or otherwise) and their relationship to air campaign goals directly drive subsequent planning decisions, the effectiveness of planning will depend heavily on the timely receipt and incorporation of new information about the execution environment (e.g., the results of actions, counter-actions, etc.). Thus, a concept frequently coupled to the notion of effects-based operations is that of a “Dynamic Air Execution Order” (DAEO). Whereas in current practice, an air execution order (i.e., the schedule of missions to be flown) is generated and released in a “batch-oriented” manner (e.g., once a day), the concept of a DAEO is more akin to a “living schedule” that is updated and expanded continuously as missions execute, effects are observed and reactions by planners are taken.

From a planning and scheduling perspective, support for effects-based operations requires technologies that can operate in a flexible and incremental manner. New tasks/actions must be continually integrated into the current plan/schedule without wholesale disruption to currently planned and scheduled air missions. Likewise, plans and schedules must be dynamically revised, as the results of prior actions and changes in resource status become known. The AFRL Effects-Based Operations Advanced Technology Demonstration (EBO ATD) was established to explore this technology vision. As part of this effort, Carnegie Mellon University was tasked to support the development of a “jump-start” system and to integrate an incremental resource allocation and scheduling tool.

In this chapter, we describe this Technology Integration Experiment (TIE). We first review ACS, an incremental air campaign scheduling application built in OZONE that was used as the basis for this work. Then we introduce the other technology components involved in the TIE and describe the operation of the integrated system. Finally, we summarize the results that were obtained and identify some future research directions.

## 5.2 Air Campaign Scheduler (ACS)

Our starting point for participation in the EBO TIE was a system called ACS (Air Campaign Scheduler) [Myers *et al.*, 2001a, Zhou and Smith, 2002]. The ACS scheduler adapts the incremental constraint-based search techniques underlying the AMC Allocator application to the air operations domain. ACS was developed originally under the DARPA JFACC After Next Program.

The overall functional scope of ACS is depicted in Figure 5.1. It is provided with three broad types of inputs:

1. - a set of *Demands*, relating to input tasks (targets or DMPIs - Desired Mean Point of Impact) to be scheduled and their corresponding constraints. Most basically, an input demand specifies a target (with an associated type or category code), a location, a priority, and a desired probability of kill. It may also specify a time window, and dependencies among other input tasks (e.g., parent objective, sequencing constraints relative to other strikes).
2. a set of *Capabilities*, specifying (1) what types and amounts of resource capacity (assets, munitions) are available for use, (2) where they are positioned in theater and over what interval(s) they may be used, and (3) a table of weaponeering solutions, that maps the effectiveness of different platform/munitions pairs to various target category codes,
3. *World State* information, indicating such exogenous factors about the execution environment as threats and weather, as well as information relating to execution results.

The output from ACS provides inputs to feed generation of a Dynamic Air Execution Order (a “continuous” form of a Master Air Attack Plan). ACS generates a set of assigned strike missions designating, for each input target/DMPI demand:

- the set of sorties to be flown (possibly converging on the target from different bases),
- the numbers of aircraft and munitions to be expended from each base and,
- the precise time windows for various stages of the flight itinerary (including TOT windows).

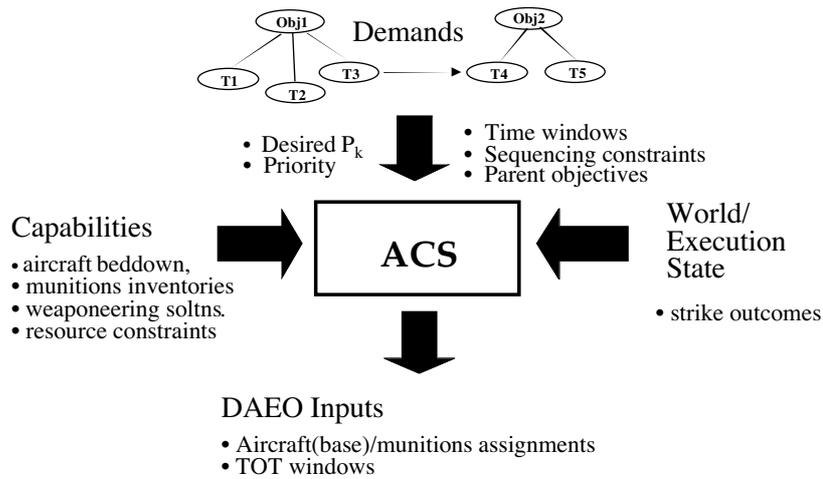


Figure 5.1: ACS Functional Scope

ACS provides a range of air campaign scheduling capabilities. In generative mode, it can be used to efficiently generate assignments of aircraft and munitions to a given set of input target/DMPI demands. As suggested above, these assignments take into account such considerations as target priorities, desired levels of destruction, time-on-target (TOT) windows, temporal sequencing constraints, feasible weaponering solutions, and aircraft/munitions positioning and availability constraints. ACS can also be used in incremental mode, both (1) to accommodate and integrate new demands into a continuously evolving air campaign schedule, and (2) to reactively reallocate in response to unexpected changes in the execution status (e.g., loss of aircraft, insufficient destruction effect). Finally, ACS provides capabilities for selective (user-driven) relaxation of constraints, providing a basis for exploring alternatives (e.g., delaying missions, surging) in situations where all constraints cannot be satisfied. The ACS user interface (See Figure 5.2) provides a range of displays for visualizing and incrementally manipulating input constraints and allocation decisions.

The ACS scheduler has been evaluated on a range of data sets including Blue Flag data obtained from the C2 Battle labs, CTEM test scenarios and programmatic data scenarios generated within the JFACC and EBO programs. An assignment for a representative problem from the Cyberland scenario data of the JFACC program consisting of 2857 DMPIs (involving approximately 4042 sorties) is solved from scratch on an 1.1 GHz Pentium IV in under 30 seconds. Incremental revisions and additions to the schedule are performed in real time. In extensive experimental studies, incremental solution change procedures developed

in ACS have been also shown to achieve significantly greater stability in the plan over time without significant adverse effect on solution quality metrics or computational expense, in comparison to “re-solving from scratch” replanning strategies. These incremental rescheduling strategies generally enable a better performance balance than minimally-disruptive (but potentially poorer quality) revision techniques. [Myers *et al.*, 2001a, Zhou and Smith, 2002]. ACS has also been integrated with SRI’s CPEF planning system to produce JPS (Joint Planner/Scheduler), a multi-agent system for incremental continuous management of air campaign plans and schedules [Myers *et al.*, 2001b].

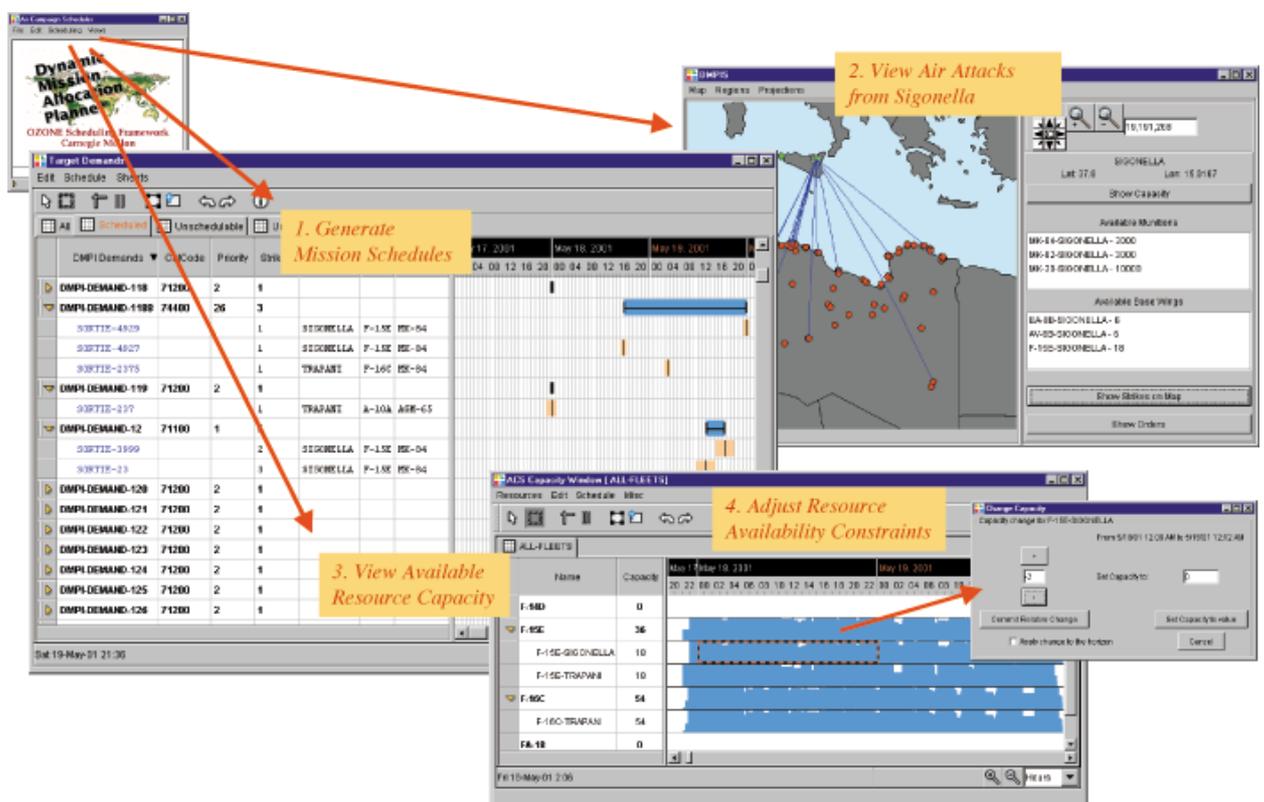


Figure 5.2: ACS Graphical User Interface

### 5.3 The EBO Jump-Start System

To produce the EBO Jump-Start system, ACS was integrated with two complementary planning technologies:

- **Causal Analysis Tool (CAT)** - CAT is a decision-theoretic planning and analysis tool developed in-house by AFRL Rome. It utilizes Bayesian integration of local probability estimates of action effects to produce:
  - prediction of plan outcomes
  - dynamic monitoring of plan execution through evidence incorporation
  - valuing of information to support monitoring

In the current context, CAT provides a representational infra-structure and inference tools for interactive development of air campaign plans, and a suite of analysis tools for assessing generated plans/schedules

- **Joint Targeting Tool (JTT)** - JTT is an operational system that provides interactive capabilities for selecting targets and determining potential weaponeering solutions. For purposes of the jump-start demonstration, a JTT simulator was developed by AFRL Rome and used as a placeholder for the operational system.

Figure 5.3 shows the flow of information between CAT, the JTT simulator and ACS. A planning episode begins with the specification of a air campaign plan using CAT. Using causal projections of the probabilities associated with achieving various effects, overall CINC objectives are interactively transformed into a hierarchical network of actions. Within this plan network, high-level actions (e.g., deny access to location *Loc*) are progressively refined into directly taskable, leaf actions (e.g., destroy bridge *B*). The set of leaf level actions are then communicated to JTT to develop specific target sets. Using JTT, each leaf level action is expanded into one or more “strike actions”, where a given strike action designates a specific target/DMPI (e.g., the bridge foundation) and a number of alternative weaponeering solutions for carrying out the strike. This set of strike actions is communicated back to CAT and added as a final ply of the campaign plan previously developed. The set of strike actions is also communicated to ACS at this point, for resource sourcing and scheduling.

Within ACS, the strike actions (targets) generated in JTT are then transformed into air strike missions, assigned to specific air units, and scheduled over time. As indicated above, JTT provides a set of weaponeering solution alternatives and preferences to ACS for each strike action. This provides one dimension of the search for a good air campaign schedule. As each strike action is received from JTT, a query is made to CAT to retrieve any constraints associated with this action. By virtue of a given strike action’s position in CAT’s hierarchical

plan network, it may inherit (1) start and end time constraints (e.g., the bridge must be destroyed before 08:00), (2) causal dependencies (e.g., the bridge must be destroyed before further actions to isolate enemy forces), and (3) percentage of effort levels (e.g., devote 10% of effort to action of denying access to location *Loc*). Given the actions and constraints provided as inputs, ACS computes a time and resource feasible set of air strike missions that accommodates the best possible mix of targets given available assets.

Once computed, the schedule is then communicated back to CAT for analysis of effectiveness. For example, CAT’s underlying probabilistic model can be used to estimate the extent to which the set of “supportable” strike missions can be expected to accomplish stated CINC objectives. This analysis might motivate revisions to the campaign plan, and any such revisions can then be selectively communicated back to ACS (through JTT), and rescheduling actions can be taken to determine the impact on the current schedule.

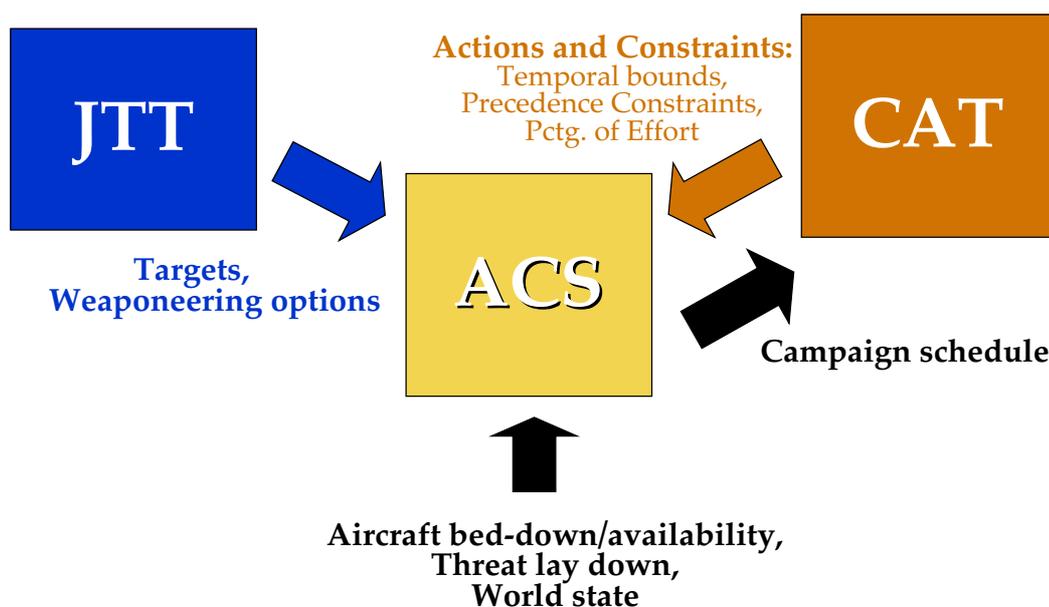


Figure 5.3: Information flows between CAT, JTT and ACS in the EBO Jump Start System.

## 5.4 Results

The integrated CAT-JTT-ACS system was successfully configured and initially demonstrated within 3 months time. The demonstration involved a planning scenario that required approx-

imately 400 air strikes to be carried out over a one month time frame. In the six months following the initial demonstration, enhanced versions of the integrated system were produced that incorporated capabilities for allocating resources to match specified percentage of effort constraints, for responding to unexpected loss of assets (temporarily or permanently), and for factoring a simple model of enemy threat into the allocation process. These achievements are particularly noteworthy given that ACS's entry into the "EBO Jump Start" demonstration system followed an unsuccessful one-year effort to accomplish a similar functional capability using an alternative air campaign scheduling technology. Our success provides further evidence of the maturity, flexibility and configurability of ACS and the underlying Ozone scheduling framework.

Current research with ACS is focused in two broad directions:

- *Factoring in the Adversary* - The current ACS model incorporates a very simple model of threat and does not reason explicitly about adversarial actions in the allocation and scheduling process. One direction of our future work is to develop and evaluate a set of ACS extensions for incorporating greater knowledge of enemy threats into the allocation model, to enable more informed assessment of predicted effects, more effective analysis of allocation alternatives, and more accurate accounting of available resources.
- *Coordination of supporting resources and derivative missions* - ACS currently allocates only those assets directly responsible for achieving the desired effects associated with input demands (target strikers, CAP aircraft). A second thread of research aims to extend this allocation model to include the full complex of support resources and missions required (e.g., escorts, jammers, refuelers), and thus provide a framework for integrated management of all campaign resources.

# Chapter 6

## Other Research Publications

### 6.1 Configurable, Mixed-Initiative Planning and Scheduling Systems

**Marcel A. Becker, *Reconfigurable Architectures for Mixed-Initiative Planning and Scheduling*, Ph.D. Thesis, Graduate School of Industrial Administration and The Robotics Institute, Carnegie Mellon University, July, 1998.**

**Abstract:** This thesis addresses the problem of building software applications for planning and scheduling systems. Most planning and scheduling problems are NP-Complete even for simplified formulations. The practical utilization of planning and scheduling systems as decision support tools requires not only dealing with the computational complexity in a reasonable way, but also considering the uncertainties associated with executing plans and schedules in a real environment. Given the complexity of the problem, implementing applications capable of generating high quality solutions for these problems is a time consuming activity. Although each application domain has its own challenges and idiosyncrasies, certain level of similarities can be identified across related problems. The development time would clearly be reduced if good implementations of these common functionalities could be adapted to be used in new applications. Despite all the software reuse effort, few applications explore these commonalities. most of the currently implemented systems and solutions are too problem specific, or too complex to provide reuse opportunity.

Motivated by recent efforts from the software reuse and from the knowledge acquisition

community, I designed and implemented a modeling framework based on an ontological model - the ozone modeling framework. The ontology defines a domain specific terminology that can be used as a language for describing planning and scheduling models. By associating capabilities or functionalities to the concepts in the ontology, a functional model can be obtained. If this functionality is supported by an actual implementation, executable models can be easily generated. The development framework proposed is composed of this planning and scheduling ontology, the implementation of the capabilities described by the ontology as an object-oriented class library, and a software tool that implements the mechanisms needed to generate executable software applications from model descriptions.

The applicability and validation of the solution approach is obtained by applying this framework to build scheduling applications in three different domains: two problems in the area of transportation and logistics - strategic deployment and aero-medical evacuation; and one problem in the area of resource-constrained project scheduling.

**Marcel A. Becker and Stephen F. Smith,"Mixed-Initiative Resource Management: The AMC Barrel Allocator", *Proceedings 5th International Conference on AI Planning and Scheduling (AIPS-2000)*, Breckenridge, CO, April, 2000.**

**Abstract:** In this paper, we describe the *Barrel Allocator*, a scheduling tool developed for day-to-day allocation and management of airlift and tanker resources at the USAF Air Mobility Command (AMC). The system utilizes an incremental and configurable constraint-based search framework to provide a range of automated and semi-automated scheduling capabilities, including generating an initial solution to the fleet assignment problem, selective re-optimization of resource allocations to incorporate new higher priority missions while minimizing solution change, merging of previously planned missions to reduce non-productive flying time, and generation and synchronization of tanker missions to satisfy air refueling requirements. In situations where all mission requirements cannot be met, the system can generate and compare alternative constraint relaxation options. The current version of Barrel Allocator will go into operational use at AMC as a module of Release 2.0 of AMC's Consolidated Air Mobility Planning System (CAMPS) in early 2000.

## 6.2 Temporally Flexible Scheduling Algorithms

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, “Profile-Based Algorithms to Solve Multi-Capacitated Metric Scheduling Problems”, *Proceedings 4th International Conference on Artificial Intelligence Planning and Scheduling*, Pittsburgh, PA, June, 1998.**

**Abstract:** Though CSP scheduling models have tended to assume fairly general representations of temporal constraints, most work has restricted attention to problems that require allocation of simple, unit-capacity resources. This paper considers an extended class of scheduling problems where resources have capacity to simultaneously support more than one activity, and resource availability at any point in time is consequently a function of whether sufficient unallocated capacity remains. We present a progression of algorithms for solving such multiple-capacitated scheduling problems, and evaluate the performance of each with respect to problem solving ability and quality of solutions generated. A previously reported algorithm, named the Conflict Free Solution Algorithm (CFSA), is first evaluated against a set of problems of increasing dimension and is shown to be of limited effectiveness. Two variations of this algorithm are then introduced which incorporate measures of temporal flexibility as an alternative heuristic basis for directing the search, and the variant making broadest use of these search heuristics is shown to yield significant performance improvement. Observations about the tendency of the CFSA solution approach to produce unnecessarily over-constrained solutions then lead to development of a second heuristic algorithm, named Earliest Start Time Algorithm (ESTA). ESTA is shown to be the most effective of the set, both in terms of its ability to efficiently solve problems of increasing scale and its ability to produce schedules that minimize overall completion time while retaining solution robustness.

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, “Scheduling Multi-Capacitated Resources under Complex Temporal Constraints”, *CMU Robotics Institute Technical Report CMU-RI-TR-98-17*, June 1998.**

**Abstract:** Most CSP scheduling models make the restrictive assumption that a resource can only support a single activity at a time (i.e., it is either available or in-use). However, in many practical domains, resources in fact have the capability to simultaneously support multiple activities, and hence availability at any point is a function of unallocated *capacity*. In this paper, we develop and evaluate algorithms for solving multi-capacitated scheduling problems. We first define a basic CSP model for this extended problem class, which provides

a basic framework for formulating alternative solution procedures. Using this model, we then develop variants of two different solution approaches that have been recently proposed in the literature: (1) a profile-based procedure - which relies on local analysis of potential resource conflicts to heuristically direct the problem solving process, and (2) a clique-based procedure - which exploits a global analysis of resource conflicts at greater computational cost. In each case, improvements are made to previously proposed techniques. Performance results are given on a series of problems of increasing scale and constrainedness, indicating the relative strengths of each procedure.

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, "An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows", *Proceedings 16th International Joint Conference on Artificial Intelligence, July, 1999.***

**Abstract:** In this paper, we extend and integrate previously reported techniques for resource constrained scheduling to develop a CSP procedure for solving RCPSP/max, the resource constrained project scheduling problem with time windows (generalized precedence relations between start time of activities). RCPSP/max is a well-studied problem within the Operations Research community and the presence of a large set of benchmark problems provides a good opportunity for comparative performance analysis. Our base CSP scheduling model generalizes previous profile-based approaches to cumulative scheduling by focusing on global analysis of minimal conflicting sets rather than pairwise conflict analysis. This generalization increases the tendency for more effective conflict resolution. Since RCPSP/max is an optimization problem, other ideas from prior work are adapted to embed this base CSP model within a multi-pass, iterative sampling procedure. The overall procedure, called ISES (Iterative Sampling Earliest Solutions), is applied to the above mentioned set of benchmark problems. ISES is shown to perform quite well in comparison to current state-of-the-art procedures for RCPSP/max, particularly as search space size becomes limiting for systematic procedures.

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, "Greedy Algorithms for the Multi-Capacitated Metric Scheduling Problem", *Proceedings 1999 European Conference on Planning, September, 1999.***

**Abstract:** This paper investigates the performance of a set of greedy algorithms for solving the Multi-Capacitated Metric Scheduling Problem (MCM-SP). All algorithms considered are variants of ESTA (Earliest Start Time Algorithm), previously proposed in [Cesta *et al.*, 1998a]. The paper starts with an analysis of ESTA's performance on different classes of

MCM-SP problems. ESTA is shown to be effective on several of these classes, but is also seen to have difficulty solving problems with heavy resource contention. Several possibilities for improving the basic algorithm are investigated. A first crucial modification consists of substituting ESTA's pairwise analysis of resource conflicts with a more aggregate and thus more powerful Minimal Critical Set (*MCS*) computation. To cope with the combinatorial task of enumerating *MCS*s, several approximate sampling procedures are then defined. Some systematic sampling strategies, previously shown effective on a related but different class of scheduling problem, are found to be less effective on MCM-SP. On the contrary, a randomized *MCS* sampling technique is introduced, forming a variant of ESTA that is shown to be quite powerful on highly constrained problems.

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, "Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems", *Proceedings 18th National Conference on Artificial Intelligence, Austin, TX, July, 2000.***

**Abstract:** One challenge for research in constraint-based scheduling has been to produce scalable solution procedures under fairly general representational assumptions. Quite often, the computational burden of techniques for reasoning about more complex types of temporal and resource capacity constraints places fairly restrictive limits on the size of problems that can be effectively addressed. In this paper, we focus on developing a scalable heuristic procedure to an extended, multi-capacity resource version of the job shop scheduling problem (MCJSSP). Our starting point is a previously developed procedure for generating feasible solutions to more complex, multi-capacity scheduling problems with maximum time lags. Adapting this procedure to exploit the simpler temporal structure of MCJSSP, we are able to produce a quite efficient solution generator. However, the procedure only indirectly attends to MCJSSP's objective criterion and produces sub-optimal solutions. To provide a scalable, optimizing procedure, we propose a simple, local-search procedure called *iterative flattening*, which utilizes the core solution generator to perform an extended iterative improvement search. Despite its simplicity, experimental analysis shows the iterative improvement search to be quite effective. On a set of reference problems ranging in size from 100 to 900 activities, the iterative flattening procedure efficiently and consistently produces solutions within 10% of computed upper bounds. Overall, the concept of iterative flattening is quite general and provides an interesting new basis for designing more sophisticated local search procedures.

**Amedeo Cesta, Angelo Oddi and Stephen F. Smith, “A Constraint-based Method for Project Scheduling with Time Windows”, *Journal of Heuristics*, 8:109-136, 2002**

**Abstract:** This paper presents a heuristic algorithm for solving RCPSP/max, the resource constrained project scheduling problem with generalized precedence relations. The algorithm relies, at its core, on a constraint satisfaction problem solving (CSP) search procedure, which generates a consistent set of activity start times by incrementally removing resource conflicts from an otherwise temporally feasible solution. Key to the effectiveness of the CSP search procedure is its heuristic strategy for conflict selection. A conflict sampling method biased toward selection of minimal conflict sets that involve activities with higher-capacity requests is coupled with a non-deterministic choice heuristic to guide the base conflict resolution process, and this CSP search is embedded within a larger iterative-sampling search framework to broaden search space coverage and promote solution optimization. The efficacy of the overall heuristic algorithm is demonstrated empirically on a large set of previously studied RCPSP/max benchmark problems.

# Bibliography

- [Allen, 1984] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.
- [Arango and Prieto-Díaz, 1991] G Arango and R. Prieto-Díaz. Domain analysis concepts and research directions. In R. Prieto-Díaz and G Arango, editors, *Domain Analysis and Software Modeling*, pages 9–32. IEEE Computer Society Press, Los Alamitos, CA, 1991.
- [Baptiste *et al.*, 1999] P. Baptiste, C. LePape, and W. Nuijten. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.
- [Baptiste *et al.*, 2001] P. Baptiste, C. LePape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Kluwer Academic Publishing, Boston MA, 2001.
- [Barnhart *et al.*, 1998] C. Barnhart, N.L. Boland, L.W. Clarke, E.L. Johnson, G.L. Nemhauser, and R.G. Sheno. Flight string models for aircraft fleet and routing. *Transportation Science*, 32(3):208–220, August 1998.
- [Batory and O’Malley, 1992] D. Batory and S. O’Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Transactions on Software Engineering and Methodology*, Oct. 1992.
- [Beck, 1999] J.C. Beck. *Texture Measurements as a Basis for Heuristic Commitment Techniques in Constraint-Directed Scheduling*. PhD thesis, Department of Computer Science, University of Toronto, Toronto CA, 1999.
- [Becker and Díaz-Herrera, 1994] M.A. Becker and J.L. Díaz-Herrera. Creating domain specific libraries: a methodology, design guidelines and an implementation. In *Proceedings*

- of 1994 3rd International Conference on Software Reuse, Rio de Janeiro, Brazil, November 1994.
- [Becker and Smith, 2000] M.A Becker and S.F Smith. Mixed-initiative resource management: The amc barrel allocator. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, pages 32–41, Breckenridge CO, April 2000. The AAAI Press.
- [Becker, 1998] M.A. Becker. *Reconfigurable Architectures for Mixed-Initiative Planning and Scheduling*. PhD thesis, Graduate School of Industrial Administration and The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 1998.
- [Biggerstaff and Perlis, 1989] T.J Biggerstaff and A.J. Perlis. *Software Reusability*. ACM Press, 1989.
- [Carlier and Pinson, 1994] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.
- [Cesta *et al.*, 1998a] A. Cesta, A. Oddi, and S.F Smith. Profile-based algorithms to solve multi-capacitated metric scheduling problems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-98)*, pages 214–223, Pittsburgh, June 1998. AAAI Press.
- [Cesta *et al.*, 1998b] A. Cesta, A. Oddi, and S.F Smith. Scheduling multi-capacitated resources under complex temporal constraints. Technical Report CMU-RI-TR-98-17, Robotics Institute, Carnegie Mellon University, June 1998.
- [Cesta *et al.*, 1999a] A. Cesta, A. Oddi, and S.F Smith. Greedy algorithms for the multi-capacitated metric scheduling problem. In *5th European Conference on Planning*, UK, September 1999.
- [Cesta *et al.*, 1999b] A. Cesta, A. Oddi, and S.F Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *Proceedings 16th International Joint Conference on Artificial Intelligence*, Stockholm, July 1999.
- [Cesta *et al.*, 2000] A. Cesta, A. Oddi, and S.F Smith. Iterative flattening: A scalable method for solving multi-capacity scheduling problems. In *Proceedings 18th National Conference on Artificial Intelligence*, Austin TX, July 2000.

- [Cesta *et al.*, 2002] A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.
- [Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, 1986.
- [Cheng and Smith, 1997] C. Cheng and S.F. Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research, Special Issue on Scheduling: Theory and Practice*, 70:327–357, 1997.
- [Cicirello and Smith, 2002] V. Cicirello and S.F. Smith. Amplification of search performance through randomization of heuristics. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 124–137, Ithaca NY, September 2002. Springer Verlag.
- [Clarke *et al.*, 1996] L.W. Clarke, E.L. Hane C.A., Johnson, and G.L. Nemhauser. Maintenance and crew considerations in fleet assignment. *Transportation Science*, 30:249–260, 1996.
- [Clements, 1996] P.C. Clements. Coming attractions in software architecture. Technical Report CMU/SEI-96-TR-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., Jan. 1996.
- [Cotter, 1996] S. Cotter. *Inside Talligent Technology*, chapter The Talligent programming model: framework concepts. Addison-Wesley, Reading, MA., 1996.
- [Dechter *et al.*, 1991] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1):61–96, May 1991.
- [Eindhoven, 1997] V.W. Eindhoven. A review of the applicability of or and ai scheduling techniques in practice. *International Journal of Management Science*, 25(2):145–153, 1997.
- [Fadel *et al.*, 1994] F.G. Fadel, M.S. M.S. Fox, and M. Gruninger. A generic enterprise resource ontology. In *Proc. of 3rd. IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, West Virginia, April 1994.
- [Gamma *et al.*, 1994] E. Gamma, R. Helm, R. Johnosn, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley, 1994.

- [Garlan and Shaw, 1994] D. Garlan and M. Shaw. An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, Pittsburgh, PA., January 1994.
- [Garlan *et al.*, 1994] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environment. In *Proc. of ACM SIGSOFT'94 Symposium on Foundations of Software Engineering*, New Orleans, LA., Dec. 1994.
- [Gruber, 1993] T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-04, Stanford University, Palo Alto, CA., Aug. 1993.
- [Gruninger and Fox, 1994] M. Gruninger and M.S. Fox. An activity ontology for enterprise modeling. Technical Report Technical Report, Ind. Eng. Department, University of Toronto, 1994.
- [Hess *et al.*, 1990] J.A. Hess, W.E. Novak, P.C. Carrol, S.G. Cohen, R.R. Holibaugh, K.C. Kang, and A.S. Peterson. A domain analysis bibliography. Technical Report SEI-90-SR-3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., June 1990.
- [Kanet and Sridharan, 1990] J.J. Kanet and V. Sridharan. The electronic leitstand: a new tool for shop scheduling. *Manufacturing Review*, 3(3):161–170, 1990.
- [Kramer and Smith, 2002] L. Kramer and S.F Smith. Optimizing for change: Mixed-initiative resource management with the amc barrel allocator. In *Proceedings of the 3rd International Workshop on Planning and Scheduling for Space*, Houston, TE, October 2002.
- [Kramer and Smith, 2003] L. Kramer and S.F Smith. Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems. In *Proceedings 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003. Springer Verlag.
- [Krueger, 1992] C.W. Krueger. Software reuse. *Computing Surveys*, 24(2):131–183, June 1992.
- [Kulpa, 1997] Z. Kulpa. Diagrammatic representation for a space of intervals. *Machine Graphics and Vision*, 6(1):5–24, 1997.

- [Lassila *et al.*, 1996] O. Lassila, M.A. Becker, and S.F. Smith. An exploratory prototype for aero-medical evacuation planning. Technical Report CMU-RI-TR-96-02, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA., January 1996.
- [Le Pape, 1994] C. Le Pape. Implementation of resource constraints in ILOG schedule: A library for the development of constraint-based scheduling systems. *Intelligent Systems Eng.*, Summer, 1994.
- [Lee *et al.*, 1996] J. Lee, G. Yost, and PIF Working Group. The PIF process interchange format and framework. Technical Report Working Paper No. 180, MIT Center for Coordination Science, May 1996.
- [Miksch *et al.*, 1998] S. Miksch, R. Kosara, Y. Shahar, and P. Johnson. Asbruvew: Visualization of time-oriented, skeletal plans. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-98)*, pages 11–18, Pittsburgh, June 1998. AAAI Press.
- [Mussettola *et al.*, 1992] N. Muscettola, S.F. Smith, Cesta A., and D. D’Aloisi. Coordinating space telescope operations within an integrated planning and scheduling framework. *IEEE Control Systems*, 12(2), Feb. 1992.
- [Myers *et al.*, 2001a] K.L. Myers, S.F. Smith, D.W. Hildum, P.A. Jarvis, R. de Lacaze, and J. Zhou. The jfac planner/scheduler. Final report for contracts f30602-97-2-0066 (cmu) and f30602-97-c-0067 (sri), DARPA, December 2001.
- [Myers *et al.*, 2001b] K.L. Myers, S.F. Smith, D.W. Hildum, P.A. Jarvis, and R. de Lacaze. Integrating planning and scheduling through adaptation of resource intensity estimates. In *Proceedings of the European Conference on Planning*, 2001.
- [Nuitjen, 1994] W. Nuitjen. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, Belgium, 1994.
- [Roth *et al.*, 1997] S.F. Roth, M.C. Chuah, S. Kerpedjiev, J.A. Kolojejchick, and P. Lucas. Towards an information visualization workspace: Combining multiple means of expression. *Human-Computer Interaction Journal*, 12(1):131–185, 1997.
- [Rushmeier and Knotogiorgis, 1997] R.A. Rushmeier and S.A. Knotogiorgis. Advances in the optimization of airline fleet assignment. *Transportation Science*, 31(2):159–169, May 1997.

- [Sharit, 1985] J. Sharit. Supervisory control of a flexible manufacturing system. *Human Factors*, 27(1):47–59, 1985.
- [Shaw and Garlan, 1994] M. Shaw and D. Garlan. Characteristics of high-level languages for software architecture. Technical Report CMU/SEI-94-TR-23, School of Computer Science and Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., December 1994.
- [Smith and Lassila, 1994] S.F. Smith and O. Lassila. Configurable systems for reactive production management. In *Knowledge-Based Reactive Scheduling*, Amsterdam (The Netherlands), 1994. IFIP Transactions B-15, North-Holland.
- [Smith *et al.*, 1996] S.F. Smith, O. Lassila, and M.A. Becker. Configurable, mixed-initiative systems for planning and scheduling. In A. Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, 1996.
- [Smith, 1989] S.F. Smith. The OPIS framework for modeling manufacturing systems. Technical Report CMU-RI-TR-89-30, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA., December 1989.
- [Smith, 1990] D.R. Smith. KIDS: A semiautomatic program development system. *IEEE Transaction of Software Engineering*, 16(9):1024–43, Sept. 1990.
- [Smith, 1994] S.F. Smith. OPIS: A methodology and architecture for reactive scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*, chapter 2. Morgan Kaufmann Publishers, 1994.
- [StAmant, 2001] R. StAmant. Intelligent data analysis in an interactive planning simulation. Technical Report Technical Report, Computer Science Dept., North Carolina State University, 2001.
- [Steels, 1990] L. Steels. Components of expertise. *AI Magazine*, 11(2):29–49, 1990.
- [Swartout *et al.*, 1996] B. Swartout, R. Patil, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. submitted to 1996 Banff Knowledge Acquisition Workshop, July 1996.
- [Tate, 1996] A Tate. Towards a plan ontology. *AI\*IA Notizie (Journal of the Italian Association for AI)*, 9(1), March 1996.

- [Uschold *et al.*, 1996] M. Uschold, M. King, S. Moralee, and Y Zorgios. The enterprise ontology v. 1.1. <http://www.aiai.ed.ac.uk/~enterprise/enterprise/ontology.html>, University of Edinburgh, UK., March 1996.
- [Uschold, 1996] M. Uschold. Building ontologies: Towards a unified methodology. Technical Report AIAI-TR-197, University of Edinburgh, September 1996.
- [Wielinga and Schreiber, 1993] B.J. Wielinga and A.T. Schreiber. Reusable and sharable knowledge bases: A european perspective. In *Proc. Int. Conf. on Building and Sharing Very Large-Scaled Knowledge Bases '93*, Tokyo, 1993. Japan Information Processing Dev. Center.
- [Wielinga *et al.*, 1992] B.J. Wielinga, W.V. Velde, G. Schreiber, and H. Akkernamans. The KADS knowledge modelling approach. In *Proc. 2nd Japanese K. A. for Knowledge-Based Systems Workshop*. Hitachi Advanced Research Lab., 1992.
- [Zhou and Smith, 2002] Q. Zhou and S.F Smith. A priority-based pre-emption algorithm for incremental scheduling with cumulative resources. Technical Report CMU-RI-TR-02-11, Robotics Institute, Carnegie Mellon University, June 2002.