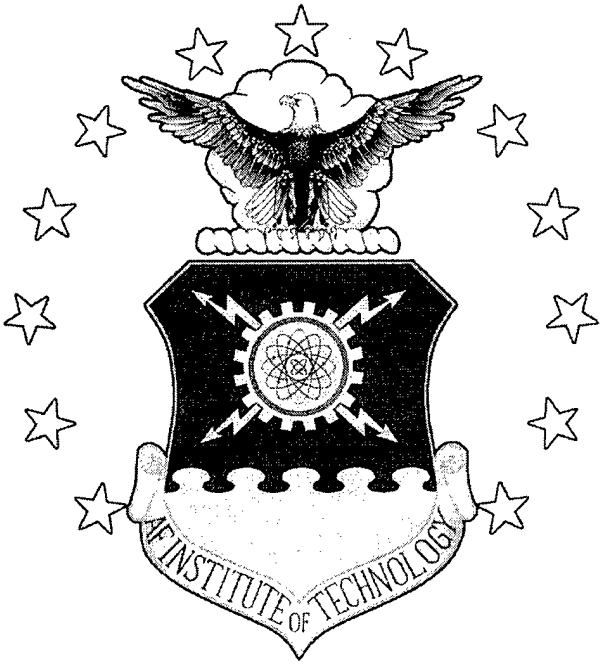
3-2000

A Human-Centered Approach for Designing Decision Support Systems

Sean G. Kern



A Human-Centered Approach for Designing Decision Support Systems

THESIS

Sean C. G. Kern, First Lieutenant, USAF

AFIT/GCS/ENG/00M-10

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/00M-10

A HUMAN-CENTERED APPROACH FOR DESIGNING DECISION SUPPORT SYSTEMS

THESIS

Presented to the faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

Sean C. G. Kern, B. S.

First Lieutenant, USAF

March 2000

AFIT/GCS/ENG/00-10

A HUMAN-CENTERED APPROACH FOR DESIGNING DECISION SUPPORT SYSTEMS

THESIS

Sean C. G. Kern, B. S.
First Lieutenant, USAF

Approved:

Dr. Henry B. Potoczny Committee Chair	3 March 2000 Date
Dr. Michael T. Cox Committee Member	3 Mar 00 Date
Maj. Michael L. Talbert Committee Member	3 Mar 2000 Date
Maj. Scott A. DeLoach Committee Member	3 Mar 00 Date

ACKNOWLEDGMENTS

I would like to thank the Lord for surrounding me with a supportive family, exceptional friends, and expert faculty. My wife Angie and my daughters Amber and Caitlin dealt with an absentee husband and father at times and for that I am sorry. I can't give back to you the time we've missed, but I can guarantee the quality of the time we have left. Thank you for your love and understanding. I would also like to thank my mom and dad for instilling in me a good work ethic and the value of setting goals. It's your turn now Megan!

The Lord blessed me with some of the most spiritually mature and intelligent friends I have ever known. They kept me grounded and on track. Thanks to my fellow Hawkeye students for the many hours of deep, light-hearted, and professional discussions we've had.

I would like to thank my thesis advisor, Dr. Henry Potoczny, for allowing me the extreme latitude I needed to complete my thesis. I would like to thank Dr. Michael Cox for his willingness to assume responsibility for a student from another school. Thank you for encouraging my interests in mixed-initiative systems and planning. I would also like to thank Major Michael Talbert for helping the "figuratively" challenged. What I could say in a thousand words, he could help me capture in a single figure. Finally, I would like to thank Major Scott DeLoach for presenting an excellent course sequence in software agents (even though he made us learn JAFMAS).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The choice to include the human in the decision process affects four key areas of system design: problem representation, system analysis and design, solution technique selection, and interface requirements specification. I introduce a design methodology that captures the necessary choices associated with each of these areas. In particular I show how this methodology is applied to the design of an actual decision support system for satellite operations scheduling.

Supporting the user's ability to monitor the actions of the system and to guide the decision process of the system are two key considerations in the successful design of a decision support system. Both of these points rely on the correct specification of human-computer interaction points. Traditional, computer-centered system design approaches do not do this well, if at all, and are insufficient for the design of decision support systems. These approaches typically leave the definition of human-computer interaction points till after the component and system level designs are complete. This is too late however since the component and system level design decisions can impose inflexible constraints on the choice of the human-computer interaction points. This often leads to the design of human-computer interaction points that are only "good enough." These approaches result in ill-conceived problem representations and poor user-system interaction points because the system lacks the underlying architecture to support these constructs efficiently. Decision support systems require a new, human-centered design approach rather than the traditional computer-centered approaches.

A HUMAN-CENTERED APPROACH FOR DESIGNING DECISION SUPPORT SYSTEMS

I. Introduction

Satellite operations squadrons throughout Air Force Space Command spend countless man-hours generating and maintaining *mission schedules*. A mission schedule lists all maintenance activities and operations activities necessary to guarantee the effective management and operation of all *resources* associated with the satellite operations mission. A maintenance activity typically involves taking one or more hardware devices (i.e., resources) offline, thereby making the device(s) unavailable for operations. An operations activity involves contacting the satellite in order to collect mission data or to perform satellite configuration management.

In general four key groups collaborate to generate mission schedules: orbital analysis, satellite engineering, maintenance, and mission scheduling. Orbital analysis produces reports required by satellite engineering; satellite engineering generates satellite operations requests based on the orbital analysis reports; maintenance generates maintenance requests based on published operating instructions; and mission scheduling consolidates all requests, performs conflict resolution, and produces the mission schedule. Each request (known generically as a *schedule request*) identifies a particular action or *activity* that is to be executed. Currently no automation support exists to assist these groups in their individual scheduling responsibilities or in their collaborative scheduling efforts.

The goal of this research is to define a methodology for designing a space operations and maintenance decision support scheduling system. The culmination of this research is the design and implementation of a system following the methodology proposed herein. The proof-of-concept system aids the users by reducing the hours required to generate and maintain mission schedules as well as assistance in maintaining the domain rules that govern scheduling. Furthermore the system proves that the design approach is highly effective at identifying essential system characteristics critical to successful generation and maintenance of mission schedules. These characteristics include problem representation, visualization, and user control of the schedule process. The remainder of this chapter is organized into the following sections: Section 1.1 provides background on the current scheduling process, Section 1.2 defines the problems associated with the process, Section 1.3 defines the goal of this research, and Section 1.4 outlines the basic objectives of this research.

1.1 Background

Mission scheduling is conducted in several phases ranging over a period of approximately 30 days. Over the course of this period mission scheduling collects scheduling requests and generates the mission schedule. In general satellite operations squadrons conduct mission scheduling similar to the process depicted in Figure 1. A highly simplified example is provided to describe the scheduling process. Details of each group involved in the process are outlined following the example.

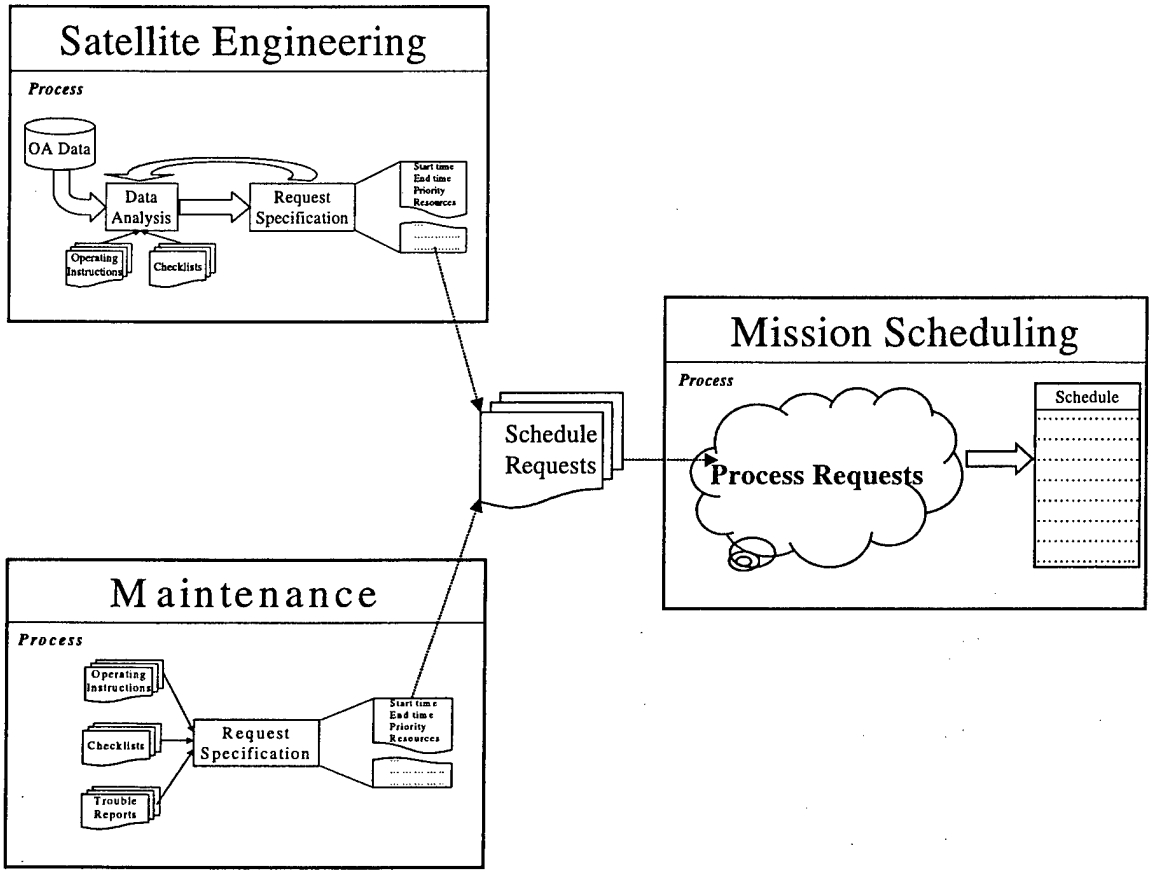


Figure 1: Satellite Operations Squadron Mission Scheduling Process

---

Example: Orbital analysis generates an Antenna Visibility Report for satellite engineering. This report lists the time intervals that each ground antenna can communicate with a given satellite. Based on this report satellite engineering submits a battery reconditioning schedule request to mission scheduling requiring contact with Satellite 1 on Antenna A starting at 0245 for one hour on Day 1. Mission scheduling places the request on the mission schedule. Later maintenance submits an antenna servo maintenance schedule request that requires placing Antenna A offline starting at 0200 for one hour on Day 1. Mission scheduling places this request on the mission schedule. Figure 2 represents the current mission schedule for the first six hours of Day 1 for Antenna A.

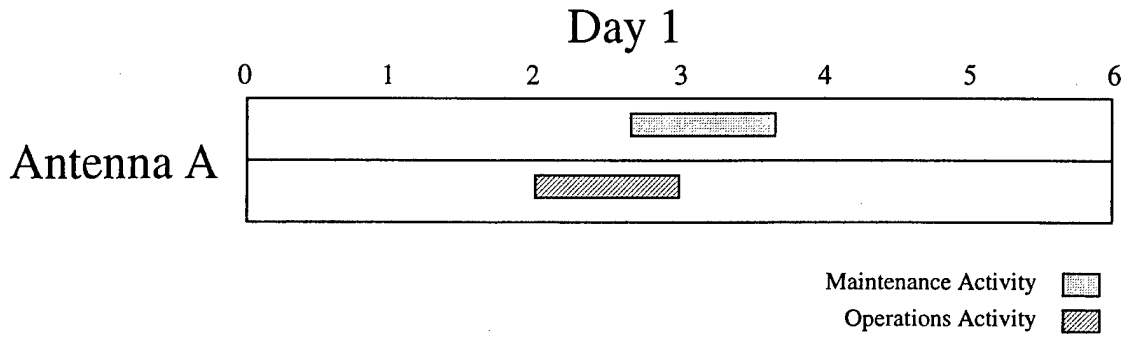


Figure 2: Preliminary mission schedule

Next mission scheduling detects any conflicts between activities on the mission schedule. A conflict does exist since both the maintenance and operations activities require the same resource (i.e., Antenna A) during an overlapping time period. Mission scheduling resolves the conflict by collaborating with maintenance to determine whether an alternative time exists to support the maintenance request. After reviewing the maintenance procedures, maintenance modifies its request to start the activity at hour 3 after the completion of the operations activity. Alternatively mission scheduling could collaborate with satellite engineering to determine whether an alternative antenna could be used instead. Figure 3 shows the resulting, conflict-free mission schedule.

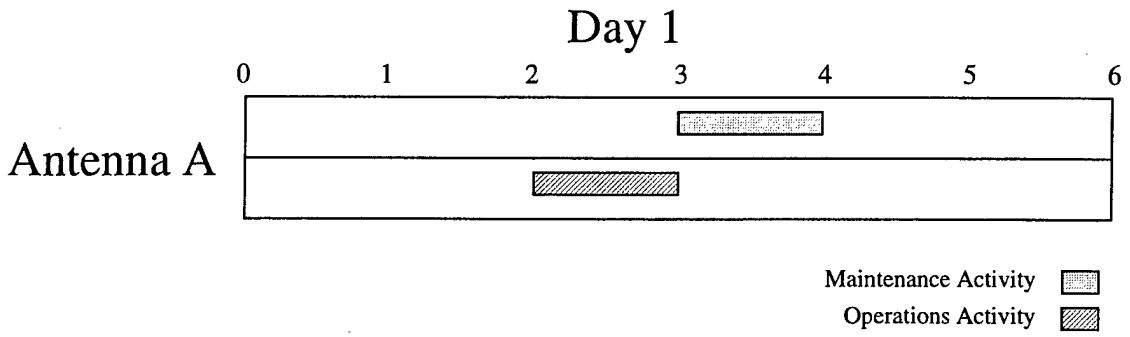


Figure 3: Conflict-free mission schedule

1.1.1 Orbital Analysis

Orbital analysis produces periodic reports for satellite engineering that describe the physical location and operating characteristics of each satellite. These reports are produced on the satellite ground system mainframe and manually transferred to a workstation. Orbital analysis notifies satellite engineering as new reports are produced. Satellite engineering accesses these files electronically.

1.1.2 Satellite Engineering

Satellite engineering determines the operations activities required for successful satellite operations. Review of the orbital analysis reports aids in determining the required operations activities. Once the necessary operational activities are identified each one is manually submitted to mission scheduling via a paper schedule request.

The current process requires satellite engineering to manually verify the filenames of the most current reports produced by orbital analysis and ensure they are placed in the correct satellite engineering workstation directory. Satellite engineering then executes an existing scheduling aid to generate required operations activities.

A myriad of user-developed aids currently support satellite engineering's scheduling efforts, but they are limited in several ways. First aids do not exist to generate all of the operations activities that can be successfully determined from the available orbital analysis reports. Instead only a subset of required operations activities is automatically generated. Thus satellite engineering manually reviews the remainder of the reports to identify the remaining operations activities. Second existing aids do not interact with the mission scheduling aids. Therefore although some activities are automatically generated, they are still submitted to mission scheduling on paper.

1.1.3 Maintenance

Maintenance is typically external to the satellite operations squadron. Maintenance is responsible for maintaining all hardware and software required for successful execution of the satellite operations mission. As such maintenance activities are scheduled on a regular basis or as conditions dictate. Examples of resources include antennas, command and control workstations, and network servers. Required maintenance activities are specified in appropriate maintenance publications and by emergency maintenance requirements. Currently no automation exists that generate required maintenance activities or that support maintenance's interaction with mission scheduling.

1.1.4 Mission Scheduling

Mission scheduling is required to consolidate all schedule requests from satellite engineering and maintenance in order to generate a complete mission schedule. The mission schedule is maintained by manually entering all operations and maintenance activities into a computer database. Any conflicting activities between satellite engineering and maintenance are manually resolved as the activities are entered. This coordination becomes critical and more difficult as the number of schedule requests increases and the activity start time draws closer. The result of this process is a conflict-free mission schedule.

1.2 Problem Statement

The current process for developing mission schedules is labor intensive, error prone, and requires extensive knowledge of the scheduling process. Automation and integration can minimize these problems. The ideal solution is a decision support system that assists users in generating, maintaining, and visualizing mission schedules. It should assist personnel with limited

knowledge of the scheduling process. Also visually representing the current mission schedule will aid in *conflict-avoidance* and conflict-resolution. The research described herein proposes a methodology for designing such a decision support system. Table 1 illustrates current process problem areas and provides a basis for specifying the objectives in Section 1.4 that the methodology supports.

Table 1: Current Process Problems

Group	Problem
Orbital Analysis	1. Must manually notify satellite engineering of new data files
Satellite Engineering	2. Must verify timeliness of data files with orbital analysis 3. Automatically generate only a portion of all requirements 4. Must manually review and compare orbital analysis data files with published operational thresholds to determine the remaining requirements
Maintenance	5. May submit a maintenance activity that immediately conflicts with a previously scheduled activity
Mission Scheduling	6. Must manually consolidate all activity requests 7. Must manually conduct line-by-line analysis of the mission schedule to identify all conflicting activities 8. Must manually resolve all conflicts 9. Must dynamically incorporate new activities at any time, possibly rendering the current schedule obsolete

1.3 Goal

The goal of this research is to define a methodology for designing a decision support system for scheduling satellite operations. The methodology supports the notion of separating the responsibilities of the groups involved where possible, but provides coordination mechanisms where collaboration is necessary. By following the methodology described herein the manually

intensive nature of the scheduling process and the potential for human error is significantly reduced. The methodology is applied to the satellite operations scheduling problem to produce a proof-of-concept system capable of generating a composite mission schedule of all required operations and maintenance activities.

1.4 Objectives

The following objectives are derived from the inherent weaknesses pervading the existing mission scheduling process described in Section 1.2. Addressing these objectives ensures the methodology successfully solves the satellite operations scheduling problem.

1. Automate interactions and information sharing between groups
2. Present a visual representation of the mission schedule
3. Schedule recurring operations and maintenance activities automatically
4. Assist human users in submission of non-standard or anomalous activities
5. Resolve conflicting schedule activities automatically or with the assistance of a human scheduler

1.4.1 Interactions and Information Sharing

Automatically modeling group interaction and information sharing eliminates the necessity of notifying other groups regarding updated data files and the timeliness of information. The use of *software agents* is a popular technique to represent groups and group interactions. Shoham defines an agent as a software entity that continuously and autonomously operates in an environment that may be occupied by other agents and processes [SHO97]. The idea that an agent is always available and can act independently leads to a high level of assurance that group interests are always considered.

A second key point that makes the use of agents ideal for modeling groups and group interactions is the idea of *social ability* [NWA96]. Social ability refers to agents that interact with

other agents or humans by way of an *agent-communication language*. This agent-communication language may be used to represent the human conversations that currently exist between groups. Figure 4 depicts agents in an architecture where the single direction arrows between groups in Figure 1 are replaced by arrows going in both directions. This new architecture reflects social ability.

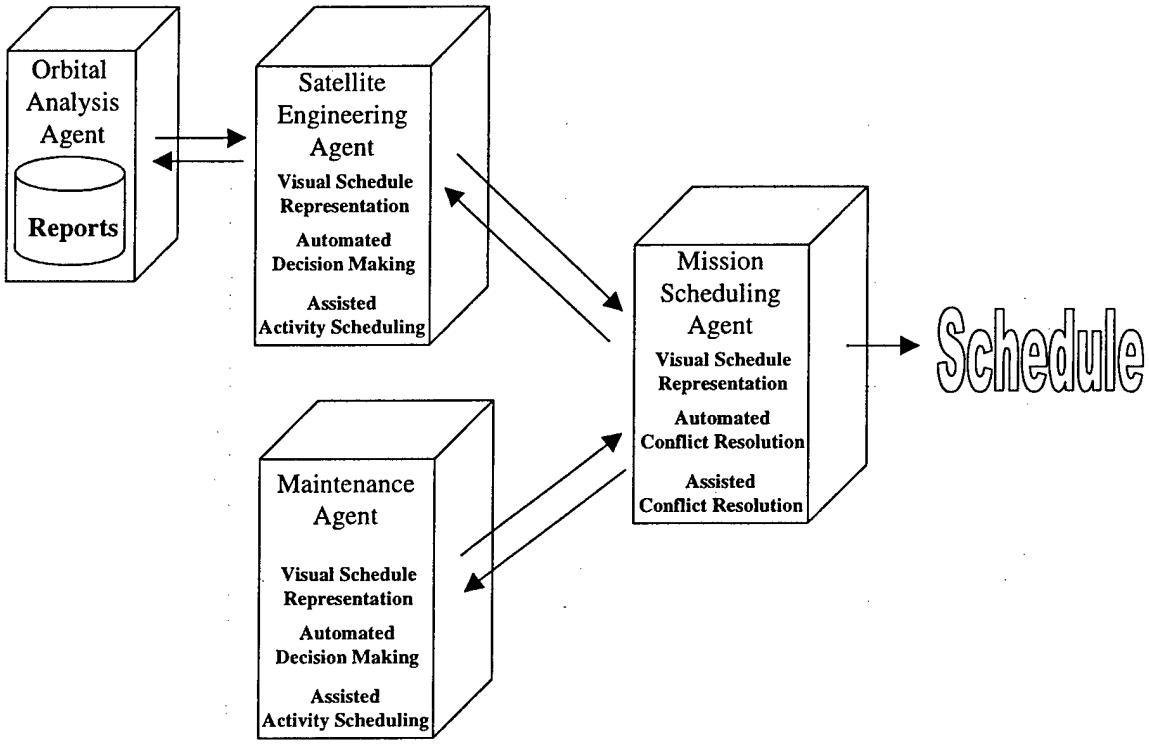


Figure 4: Agent Architecture for Mission Scheduling System

Furthermore agents are often classified by the *roles* they assume in their environment. *Information agents* are agents that can access, retrieve, and manipulate information obtained from any number of information sources. They can also answer queries about the information that they can access [WJ95]. In this role information agents might provide a possible solution to solving the information sharing problem between groups.

Scheduling can be a computationally expensive process. By decomposing and distributing the scheduling process a system can assign part of the problem to an agent that is specifically designed and optimized to work on a given subproblem. In addition the system can take advantage of the processing power on several machines instead of just one. This paradigm suits many real-world scheduling applications that characteristically take place in a distributed environment. Examples include airline scheduling systems and satellite operations.

1.4.2 Schedule Representation

Perry recognized that increased scheduling efficiency can be gained by placing emphasis on the user interface design of a scheduling system as the primary means of producing a conflict free schedule [POP92]. His intent was to add more complex reasoning to the interface as the system matured. Assuming a reasonably mature interface exists, complex reasoning can be added to the system using an *intelligent agent*. Intelligent agents are software agents that act on behalf of users or other programs to carry out a set of operations.

The content and form of the visualized schedule differs based on the needs and interests of each user group. Referring to the previous example maintenance will benefit by viewing a mission schedule in terms of hardware availability (i.e., Antenna A), distinguishing between scheduled maintenance activities and operations activities. Satellite engineering cares less about viewing activities organized by resource and more about viewing the operations activities scheduled for each satellite.

A user interface with an intelligent agent can support automated conflict resolution and schedule request submission. Another benefit of a user interface that can represent the schedule visually is the potential for conflict avoidance. Currently maintenance submits schedule requests in the "blind." In the previous example maintenance was unaware that an operational activity was

already scheduled during the same time interval that maintenance wished to place Antenna A offline. This scenario will always result in mission scheduling conducting conflict resolution. If maintenance had an *a priori* view of the current mission schedule, it could have initially submitted the schedule request for a time that Antenna A was available. Therefore the conflict depicted in the example would be avoided.

1.4.3 Automated Scheduling

One of the significant advantages of an automated scheduling system is the ability to generate all activities and then subsequently place the activities on the mission schedule--conflict-free. A computer system can generate required activities and schedule them much quicker than a human scheduler. This is a good example of leveraging the strengths of computing where it is needed most. Regarding the need for schedule visualization, the system can present how schedule generation takes place (i.e., the reasoning process) in a way that clearly states what decisions are made as well as the conditions on which they are made. This ability is vital to support the varying levels of user interaction required throughout schedule construction and conflict resolution.

1.4.4 Non-standard and Anomalous Scheduling

Due to the dynamic and uncertain environment of space operations, the system can not be imbued with the knowledge capable of generating all activities that are required. Therefore the system must support human interaction. The system can support automated data entry for schedule request submission in a simple, data driven manner or provide monitored, intelligent support. From the previous example an operations request specified the use of Antenna A because it was visible from Satellite 1. Suppose an inexperienced user did not verify the antennas visible from Satellite 1. Instead the user tries to submit the schedule request requiring the use of Antenna B for a time when Antenna B is not visible from Satellite 1. The scheduling system can identify

the problem to the user and suggest an alternate antenna. In another scenario the user wants to submit a schedule request, but has no idea which antennas are visible from Satellite 1. The scheduling system can offer suggestions based on information provided by a relevant information agent.

1.4.5 Conflict Resolution

The system incorporates a principled model of conflict resolution that can be understood by the mission scheduler. This is in the form of classifying conflicts and applying appropriate conflict resolution schemes. Unfortunately the system can not contain all the logic necessary to resolve all possible conflicts. In these cases the human scheduler can resolve any conflicts in the schedule. This interaction can occur by allowing the user to modify activities in the user interface while the system reports the results of each user action.

The bottom line is that the user must interact with the scheduling system on several levels: schedule modification, schedule request input, and conflict resolution. This required interaction begs the question, "How should the system and the human interact to achieve the best schedule results?" In the literature this case is referred to as *mixed-initiative scheduling*. Mixed-initiative scheduling characterizes a scheduling process in which the system and the human interact intimately during the scheduling process. The goal of mixed-initiative scheduling is to generate a higher quality schedule than what the computer or the human alone could generate [OC94] [FAM96] [CV97]. Scheduling operations best suited for automation and those best left to the human scheduler must be identified *a priori* in order to leverage the best capabilities of the human user and the computer system.

1.5 Thesis Overview

Chapter 2 provides the background material required to understand scheduling system development emphasizing mixed initiative characteristics where applicable. Chapter 3 describes the design methodology of the decision support system. Chapter 4 presents the design decisions that resulted from applying the design methodology in Chapter 3. Chapter 5 discusses the development effort involved in the creation of the scheduling system. Finally Chapter 6 discusses conclusions reached during the study and possible future research.

1.6 Summary

Satellite operations squadrons have manual, labor-intensive, tightly interrelated scheduling processes that require high levels of group decision making and information sharing. A current scheduling process was outlined in Section 1.1 and the problems and limitations associated with that process were highlighted in Section 1.2. The goal of this thesis and the objectives for reaching that goal were discussed in Sections 1.3 and 1.4, respectively. By applying current concepts in artificial intelligence scheduling, software agents, and mixed-initiative scheduling, these concerns are addressed and resolved.

II. Background

2.1 Overview

As early as 1972 Simon recognized the problem of allocating resources over time [HAS72]. But it wasn't until the 1980's that *artificial intelligence scheduling* became a mainstream research issue in artificial intelligence, especially in the domains of manufacturing, military transportation, and space [MF94]. Up to that point *artificial intelligence planning* was a primary focus. As a result much of the pioneering efforts in scheduling were shaped by the advances in planning

Many definitions for planning and scheduling can be found. Russell and Norvig define a planning task as "deciding *what* steps are going to be performed," whereas a scheduling task is defined as "deciding *when and where* each step will be performed" [RN95]. Fox provides a more elaborate definition of planning as "selecting and sequencing activities such that they achieve one or more goals and satisfy a set of domain constraints" [MF94]. The concepts of *goals* and *constraints* are central to the planning process. Reasoning about constraints provides the planner the ability to determine the sequences of actions that result in the attainment of the desired goals. Early work in planning recognized the benefit of reasoning about constraints in systems [MS81]. Constraints are used to represent interactions between subproblems in a manner that can reduce the search space resulting in a simpler problem to solve.

Fox further suggests that scheduling is a continuation of the planning process [MF94]. Planning is typically concerned with *ordering* events, without regard to start times, event duration, and other temporal considerations. In contrast scheduling considers *temporal restrictions* as a primary means for constructing schedules. Scheduling assigns resources and

times for each activity with respect to temporal restrictions, resource requirements, and the *capacity* limitations of the set of resources. Reasoning about resources with regard to such characteristics as capacity, location, and type are key points in scheduling.

Fox was the first to use constraint reasoning to direct the construction of schedules [MF94]. Known as *constraint-directed scheduling* it is a powerful approach in searching for a schedule solution, but this method suffers from combinatorial complexity just like the planning problem does. The scheduling problem in general was proven to be NP-Hard [DC87]. As a result a rich set of *heuristics* and problem decomposition strategies exist to account for scheduling's intractability. Other approaches suggest leaving the human in the loop. With only minor direction to the scheduling system it is possible for the human scheduler to reduce the search space considerably.

In summary the scheduling problem has the following characteristics [MF94].

- Time-based: activities selected, sequenced and assigned resources and time of execution
- Multi-agent: each process is an agent for which a schedule is created
- High resource contention: decisions are tightly coupled
- Search is combinatorially explosive

The body of this chapter addresses several aspects that are crucial to understanding the scheduling problem. Section 2.2 focuses on the essential elements necessary for designing scheduling systems. Common characteristics of scheduling problems and methodologies in the large are presented. Section 2.3 addresses the requirements for and benefits of mixed-initiative scheduling. The final section presents some considerations for conducting software system analysis and design.

2.2 Scheduling System Construction

In order for scheduling systems to be successful "in the large," system lifecycle costs must be comparable to other operational systems [SC96]. Rather than developing a system from the ground up each time, a generic representation of the scheduling problem can aid in reducing scheduling system lifecycle costs. If this general representation exists then it can be extended to solve domain-specific scheduling problems.

Several researchers tackled the problem of devising general representations of the scheduling domain. The goal was to develop *object-oriented frameworks* that can capture the domain independent, essential elements of any scheduling process [DEJAVU98], [JCB98], [SB97], [COS99]. Simply by extending the existing object-oriented framework researchers can construct scheduling systems to support any domain dependent scheduling process. Each research effort used standard object-oriented techniques to decompose the scheduling problem into *abstract objects*. With abstract objects defined they then construct general object-oriented libraries used in the construction of scheduling systems. The overall manner and focus in which each project represents the problem is slightly different, but the differences are negligible. It is worthwhile to review the concepts and methodologies of the respective research projects to get a clear, well-rounded treatment of the elemental characteristics of the scheduling problem.

2.2.1 General Scheduling Concepts

Smith and Becker propose an *ontology* that is characterized as a meta-model of the domain of scheduling [SB97]. An ontology is "a formal description of entities and their properties, providing a sharable terminology for describing and representing objects of interest in a given domain" [SB97]. The result of defining a formal scheduling ontology is a base of abstract objects that form a framework for explaining and defining the general schedule problem. This

ontology is used as a tool for analyzing specific scheduling domains and designing the software *concrete objects* necessary to instantiate a given application system.

The five abstract objects in the scheduling ontology are DEMAND, ACTIVITY, RESOURCE, PRODUCT, and CONSTRAINT (all caps refer to specific ontological concepts). These abstract objects are also adopted in some form by other research efforts such as Becks's ODO framework and Cesta's, Bazzica's, and Casonato's O-OSCAR framework [JCB98][CBC97]. Representing the interrelationships between these abstract objects defines an abstract model of the scheduling domain. Figure 5 depicts this abstract model

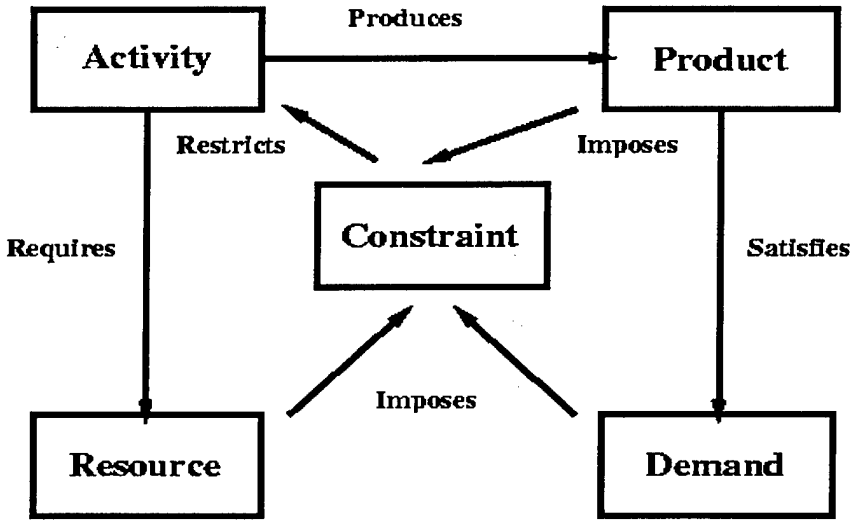


Figure 5: Abstract Scheduling Domain Model[SB97]

Each abstract object has basic *properties* and *capabilities*. Properties define attributes or parameters of relevance for specifying an executable scheduling model. Capabilities establish protocols for operationalizing concrete object definitions in terms of the abstract object functionality required to construct an overall solution [SB97].

Other researchers introduce similar abstract objects. Jorgen defined a *schedule* object as the parent object in the framework [DEJAVU98]. The schedule object consists of three parts:

- a list of resources with scheduled *allocations*
- a list of *jobs* with their *operations*
- a list of constraints

An allocation assigns an operation that is part of a job to a resource. The time of the allocation is described by a *temporal interval* consisting of a start time, a duration, and an end time. Here an operation is equivalent to an atomic activity and a job is equivalent to an aggregate activity as in [SB97]. A resource stores the operations to be performed on it as well as state information such as capacity and availability. An *order* describes the product to be produced and is essentially equivalent to DEMAND mentioned above. Finally a constraint is a relation between two or more scheduling objects or attributes of scheduling objects.

It is clear that a general consensus exists regarding what abstract objects constitute the scheduling domain model. Smith and Becker provide a very detailed specification for their ontological concepts that is presented next [SB97].

2.2.1.1 DEMANDS

A DEMAND is a request for goods or services, known as PRODUCTS in this representation, 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 determines the current scheduling problem to be solved.

A DEMAND has several properties:

- PRODUCT: 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 partial ordering over the entire set of demands.
- ACTIVITIES: The set of activities that fulfill the demand.

2.2.1.2 PRODUCTS

A PRODUCT is a good or service provided by the system. It is realized through execution of some set of activities. A DEMAND for a PRODUCT is considered satisfied when all of those activities are complete.

Properties of interest in defining a PRODUCT relate to the mapping from DEMANDS to ACTIVITIES:

- 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, along with the constraints and parameters of a requesting DEMAND, enables the instantiation of a set of ACTIVITIES for fulfilling the DEMAND. These ACTIVITIES contain the *decision variables* (start times, end times, assigned resources) of the problem to be solved. The instantiation process restricts the domains of these decision variables according to the constraints specified in the DEMAND.

2.2.1.3 RESOURCES

A RESOURCE is an entity that supports or enables the execution of ACTIVITIES. The availability of RESOURCES constrains when and how ACTIVITIES execute. Making efficient use of RESOURCES in support of multiple, competing ACTIVITIES is the crux of the scheduling problem.

A RESOURCE is modeled as providing some amount of capacity, a numeric quantity that varies over time as a function of allocating the RESOURCE to various ACTIVITIES and its associated *allocation semantics*. Allocation semantics refer to the resource's physical structure. A resource may be physically structured to satisfy only one activity at a time or to satisfy multiple, concurrent activities.

The most important RESOURCE properties to consider are those that affect a RESOURCE's availability and utilization. RESOURCE availability is related to its capacity model. Two models exist that impose different *capacity constraints*.

1. *UNIFORM-CAPACITY: represents capacity as a scalar quantity. The capacity constraint for this type of resource dictates that the sum of the capacity used/consumed by all supported ACTIVITIES at any point in time must be ≤ the capacity of the RESOURCE.*

2. *HETEROGENEOUS-CAPACITY: represents capacity as a vector of two or more UNIFORM-CAPACITIES reflecting partitioned sub-capacities.*

In many cases other factors may determine the usage properties for domain resources. These factors are usually application specific, but may include concepts such as speed, range, and unavailability-intervals (due to maintenance for example).

2.2.1.4 ACITIVITIES

An ACTIVITY represents a process that is 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 may be decomposed into a set of more-detailed SUB-ACTIVITIES enabling processes to be described at multiple levels of abstraction.

An ACTIVITY designates a set of decision variables. The action of scheduling an ACTIVITY involves determining the values of these variables. There are several basic decision variables associated with an activity.

- *start time, end time*: delineate the time interval that the ACTIVITY will occur
- *assigned resources*: indicate the set of RESOURCES allocated to the ACTIVITY.

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

- *duration*: the time 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 this ACTIVITY was instantiated to satisfy. The DEMAND imposes *earliest-start-time* and *latest-finish-time* constraints, and associates priority information.
- *status*: an ACTIVITY may be in one of several states: UNSCHEDULED, SCHEDULED, IN-PROCESS, or COMPLETED.

An ACTIVITY provides capabilities for incrementally allocating resources and making variable assignments, for retracting previous assignments, and for propagating the consequences of these decisions to related ACTIVITIES.

2.2.1.5 CONSTRAINTS

A CONSTRAINT restricts the set of values assigned to a variable. CONSTRAINTS restrict the assignment of start and end times and the allocation of RESOURCES to ACTIVITIES. Several types of CONSTRAINTS exist: *value-compatibility-constraints*, *temporal constraints*, and *resource-availability-constraints*. Value-compatibility constraints ensure that the right types of resources are assigned to a given activity. Temporal constraints restrict the values of time-sensitive variables. An example is the due-date constraint of a demand. The due-date restricts the end-time of the last activity selected to satisfy the demand. Finally resource-availability constraints model resource capacities as constraints. That is a resource of capacity 2 is constrained to support no more than two activities at a time.

The primary property of a CONSTRAINT is whether or not it may be modified or violated. The problem solver is never allowed to violate *hard constraints* and *soft constraints* are considered relaxable if need be. The designation of *relaxable constraints* is typically accompanied by a specification of *objectives* or *preferences*. When due dates can be relaxed, minimizing tardiness is a common objective. Objectives and preferences prioritize the space of possible relaxations of a CONTRAINT and provide a basis for measuring *solution quality*.

2.2.2 Scheduling System Frameworks

The four scheduling frameworks presented here have similarities and differences in the way each construct schedules. This section gives an overview of each framework.

2.2.2.1 Déjà Vu

Déjà Vu is a reusable framework for the construction of intelligent interactive scheduling systems [DEJAVU98]. Déjà Vu takes an object-oriented approach to building a reusable

framework. It derives its abstract objects from the large amount of theoretical work available on scheduling. This helps identify objects such as orders (i.e., DEMANDS), jobs (i.e., PRODUCTS), operations (i.e., ACTIVITIES), resources, allocations, and schedules.

The core of Déjà Vu is a framework of abstract classes representing the basic scheduling theory. Abstract classes also represent basic forms of constraints. This abstract core enables an application independent definition of several scheduling concepts:

- *schedule evaluation* (all constraints stored in a constraint list are evaluated and aggregated)
- scheduling tasks (exchange of operations on a resource, exchange of jobs, ...)
- algorithms that apply and compare applicable scheduling tasks to find better schedules
- graphic entities like windows, panes, and text fields to represent scheduling objects on the desktop

An abstract root class already contains many methods sufficient for handling schedules. Representing application-specific information in the schedule class or overloading general methods of the schedule class with more efficient domain-dependent strategies further specializes a schedule. Each schedule type has its own method for deciding which scheduling tasks are applicable and how it is performed.

Déjà Vu views scheduling as a process controlled by constraints and guided by several objective functions. Constraint types such as temporal and resource constraints are typical of scheduling processes. Other domain specific constraints may also exist. Deriving existing Déjà Vu constraint classes produces new constraint types. This reduces the overall system development effort.

A constraint is modeled as a relation between two or more scheduling objects or entities. Such scheduling objects could be activities or resources. The relation is mapped on a *satisfaction degree* that evaluates how good this constraint is satisfied in the actual schedule. Different

constraint types obtain domain-dependent weights reflecting the constraint's importance for the domain. A schedule is then evaluated by a *weighted aggregation* of all satisfaction degrees.

Déjà Vu recognizes the necessity and advantages of keeping the person in the loop. A key characteristic of any usable scheduling system in a complex, dynamic environment is that it be adaptable and under full control of the user to overrule outdated system rules, operations, and other system functions deemed unacceptable by the user. In this sense Déjà Vu is designed to give the user ability to let the system schedule automatically or perform some of the scheduling tasks manually. The framework supports these interaction levels by defining *scheduling tasks* that provide a common interface with methods for undoing, redoing, evaluating, and executing schedule alterations. All actions are context-dependent. Therefore the system knows which actions are allowed in certain contexts and modifications to the schedule are made if an allowed action is performed. The following scheduling tasks are defined and are easily extended if other tasks are necessary for an application.

- allocate a job as early as possible
- allocate a job after another job
- allocate a job at a certain time
- remove a job (back into the list of orders)
- exchange two adjacent jobs
- move a job to another position
- exchange an operation with an adjacent operation
- move an operation to another place on the same resource
- move an operation to a different resource
- shift an operation

Scheduling tasks are used to transform a schedule into a new and similar schedule. If several tasks are applicable a procedure chooses the task apply. Some look-ahead technique is often used to determine the best scheduling task to apply given some measure of the resulting schedule quality. Therefore the comparison of schedules by an evaluation function is necessary. Déjà Vu allows the user to select between different heuristics (tabu search, simulated annealing,

iterative deepening, and genetic algorithms) to improve the search and to set different parameters of these algorithms individually.

2.2.2.2 ODO

"The ODO framework is an approach that views the modeling and solution of a constraint-based scheduling problem from a unified model that combines common components and isolates essential differences" [CB98]. Furthermore the structure of the framework fosters understanding of constraint-directed scheduling algorithms. It is capable of modeling a wide variety of existing, as well as derived, constraint-directed scheduling algorithms. These algorithms are broadly categorized into three classes: *heuristic commitment techniques*, *propagators*, and *retraction techniques*. In addition to these classes, there are several other components that comprise the ODO framework: *constraint graphs*, *scheduling strategy* or *policies*, and *commitments*. The *constraint graph* is used as the primary means of representing a scheduling problem. *Commitments* are asserted into and retracted from the constraint graph by the *policy*.

A *heuristic commitment technique* is a procedure that finds new commitments to assert in the graph to (heuristically) move towards a solution [CB98]. The technique is divided into two steps. First it performs some measurement of the constraint graph to distill information about the search state. Second it uses this information to heuristically choose a commitment to add to the constraint graph. This information is commonly referred to as a *texture measurement*.

Texture measurements form the basis of the heuristic commitment techniques in ODO. Each heuristic is characterized by what type of commitments are allowed, how commitments are chosen, how much constraint propagation is performed, and what the acceptance and termination criteria are. Figure 6 depicts a generic representation of many heuristic search approaches.

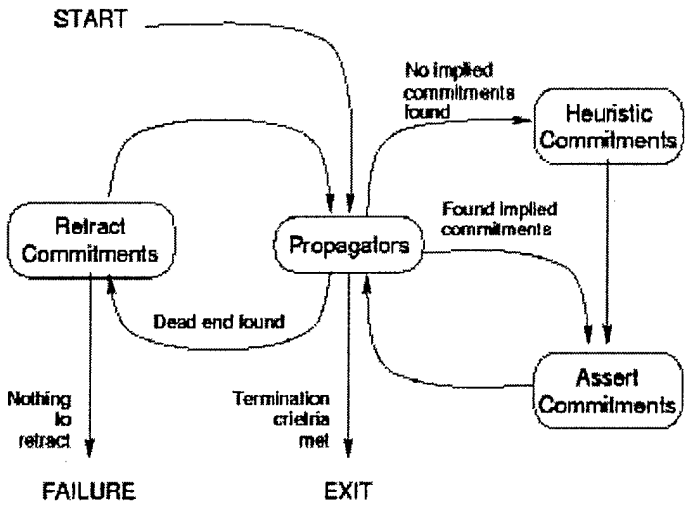


Figure 6: The ODO Problem Solving Framework

A *propagator* is a procedure that examines the existing search state to find commitments that are logically implied by the current constraint graph, but are not explicitly present [CB98]. By making these constraints explicit they are used to prune the number of possibilities to be explored in the search space. A key requirement is that a propagator be sound in the commitments that it makes. A propagator never infers a constraint that is not a logical consequence of the current problem state. The result of making more constraints explicit is that other propagators might then be used to further prune the search space. The amount of constraint propagation that enables a problem-solver to be the most efficient varies with the problem solver, the application domain, and the problem solving context [CL94]. Therefore control is required when applying propagators.

The last algorithm class is *retraction techniques*. A retraction technique is a procedure for identifying existing commitments to remove from the constraint graph. The following example is taken from [JCB98].

Assume that a search algorithm moves through a sequence of states $S=(s_0,\ldots,s_k)$ *as a result of the assertion of a number of commitments* $A=(a_1,\ldots,a_k)$. *Furthermore, assume that a mistake is made: as a result of one or more of the commitments in* A *the search has reached a state* s_k *which is inconsistent with respect to the constraints in the problem. This is a dead-end in the search.*

To resolve the dead-end some commitments are retracted ($C \in A$). The retraction component of the search strategy must answer two questions:

1. Which commitments should be retracted?
2. In retracting a commitment that was made, say at state s_i, where $i < k$, what should be done with the intervening commitments, those made in all states s_j, where $i < j < k$?
3.

Different retraction techniques supply different answers to these questions. Figure 7 depicts a dead-end search.

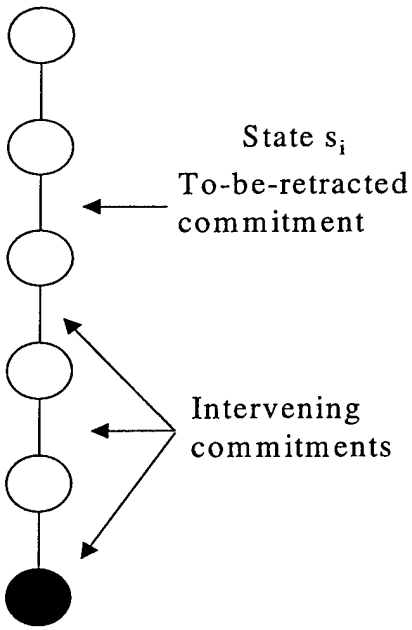


Figure 7: A Dead-end Search

In conjunction with these techniques there is also a list of user-defined conditions known as *termination criteria* that characterize the end search state. These criteria take the form of solution definitions, limits on the search in terms of CPU time, number of commitments, and other measures. The constraint graph contains a representation of the current state of the problem in the form of variables, constraints, and objects built from variables and constraints. These constraints and variables are aggregated to form the components of the scheduling problem. Lower level components may include interval variables that are assigned to an interval of integer values and constraints expressing various mathematical relationships (i.e., less-than, equal) among interval variables. At the aggregate level the constraint graph may represent activities, temporal relations, resources, and inventories with minimum and maximum constraints.

A commitment is a constraint, a variable, or a set of constraints and variables that the search strategy adds to or removes from the constraint graph. The *assertion* and *retraction* of commitments are the only search operators. Assertion of a commitment is the process of adding the problem objects in the commitment to the constraint graph. Retraction of a commitment is the process of removing a commitment from the constraint graph. Conceptualizing the search space as a tree of constraint graphs, state transitions from one constraint graph to a neighboring constraint graph is a result of either asserting or retracting commitments.

The overall goal of a heuristic in ODO is to reduce contention on resources (i.e., reduce the aggregate demand on resources). The scheduling problem in this framework is not viewed as one of optimization, but simply one of satisfaction. In this case reducing contention across the system can lead to one or more satisfying solutions. ODO defines a policy as the exact specification of how each step in the general search loop is performed. Therefore the policy is the actual embodiment of the heuristic-based algorithm that is operating on the constraint-based

representation of the scheduling problem. Texture measures guide the decisions made at these steps. Figure 8 depicts the general search loop in ODO.

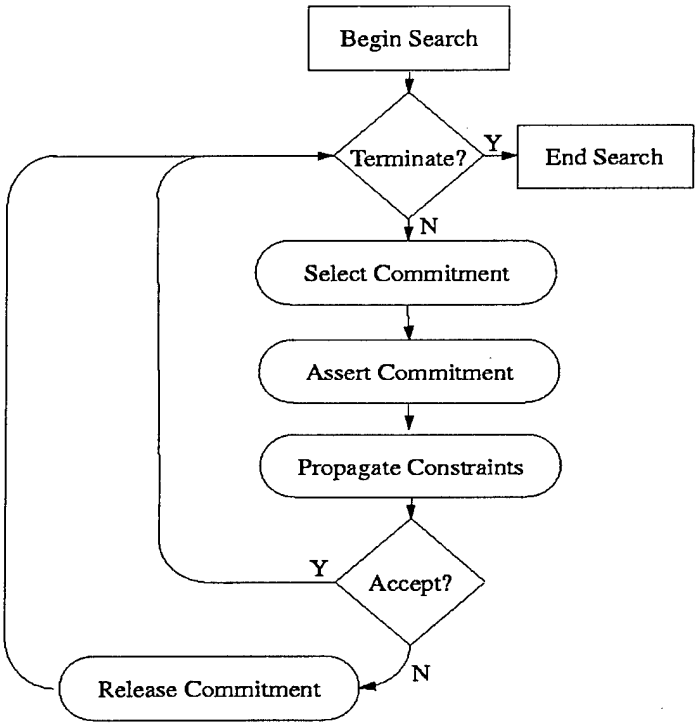


Figure 8: General Search Loop [JCB98]

2.2.2.3 OZONE

OZONE is a toolkit for configuring a constraint-based scheduling system. 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. By associating concrete object capabilities directly with abstract objects in the ontology, executable systems are rapidly configured and the modeling effort of domain-specific aspects receives the most attention.

OZONE defines scheduling as a process of feasibly synchronizing the use of resources by activities to satisfy demands over time. The five abstract objects in the ontology together with

their interrelationships define an abstract model of a scheduling domain and a framework for analyzing and describing particular application environments. The abstract model and its properties are extensible through abstract object specializations to create concrete objects that define specific models for various subdomains.

OZONE defines a problem solving organization that distinguishes two components: a *decision-making component* and a *constraint-management component.* The decision-making component is responsible for making choices among alternative scheduling decisions and retracting those that have since proved undesirable. The constraint-management component propagates the consequences of these decisions and incrementally maintains a representation of the current set of feasible solutions. Schedule construction, revision, and improvement proceed iteratively within a basic *decide and commit cycle.*

OZONE provides an application skeleton in the form of a generic constraint-based scheduling system. Constructing a scheduler using this skeleton approach requires several steps.

- Selecting suitable classes from the library (this involves mapping the application onto the ontological model)
- Combining the selected classes into more complex services, using conceptual and architectural techniques
- Extending the existing classes to provide domain-specific functionality (this usually involves specializing or overriding methods provided by the library)

OZONE scheduling is formulated as a *reactive process*, reflecting the fact that a schedule at any level or stage of the planning process is a dynamic evolving entity and is continually influenced by changing mission requirements, decision-making perspectives and goals, and changing execution circumstances.

2.2.2.4 O-OSCAR

Cesta, Bazzica, and Casonato developed the Object-Oriented Scheduling Architecture (O-OSCAR) for managing requests for a data relay satellite [CBC97]. The architecture combines object-oriented and artificial intelligence methodologies to develop a comprehensive scheduling problem approach. The research focuses primarily on the representation of the scheduling domain, the dynamic maintenance of a solution, and the interaction with different types of users.

Artificial intelligence techniques provide an approach to a scheduling problem based on three fundamental aspects:

- representation of the domain and solution management
- generation of satisfactory or optimal solutions
- interaction with the user

Figure 9 depicts the basic aspects of the artificial intelligence approach.

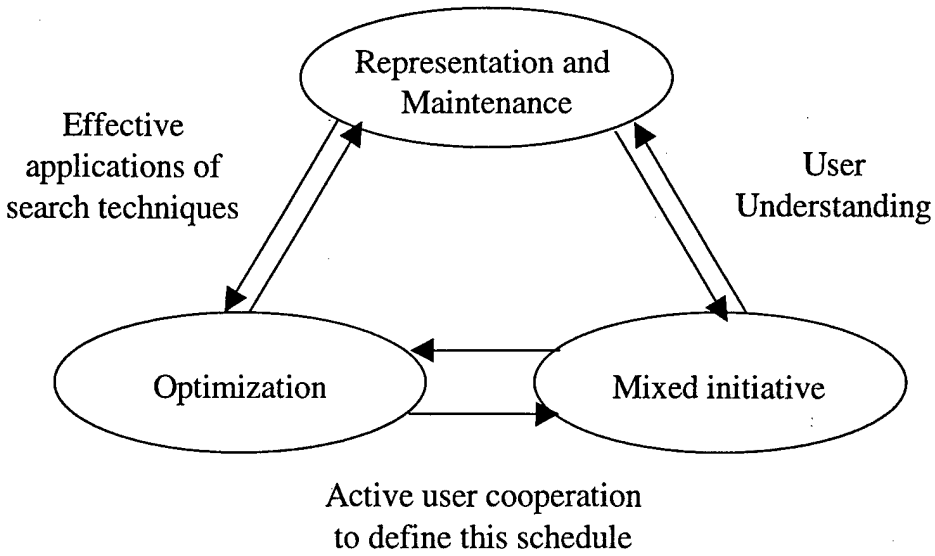


Figure 9: O-OSCAR A I Approach

Consider two points regarding domain representation. First a dynamic representation supports the physical changes (i.e., resource availability) that may occur in the domain and

supplies an incremental building of a schedule. Second a symbolic domain representation allows the user a high-level of understanding.

The main objective of the project is to keep the user inside the scheduling process. It is important to study the schedule environment to define the functions that need automation. For each type of user in the system, a personalized set of instruments to manipulate the schedule is defined. This requires user profile definitions. The user is in control of each schedule building step with these instruments and profiles in place.

The block diagram of the software system is presented in Figure 11. The modularity of the system allows the definition of new heuristics and/or new user profiles without modifying the main structure of the system. The knowledge representation module is responsible for the representation of the domain and for the maintenance of the current solution.

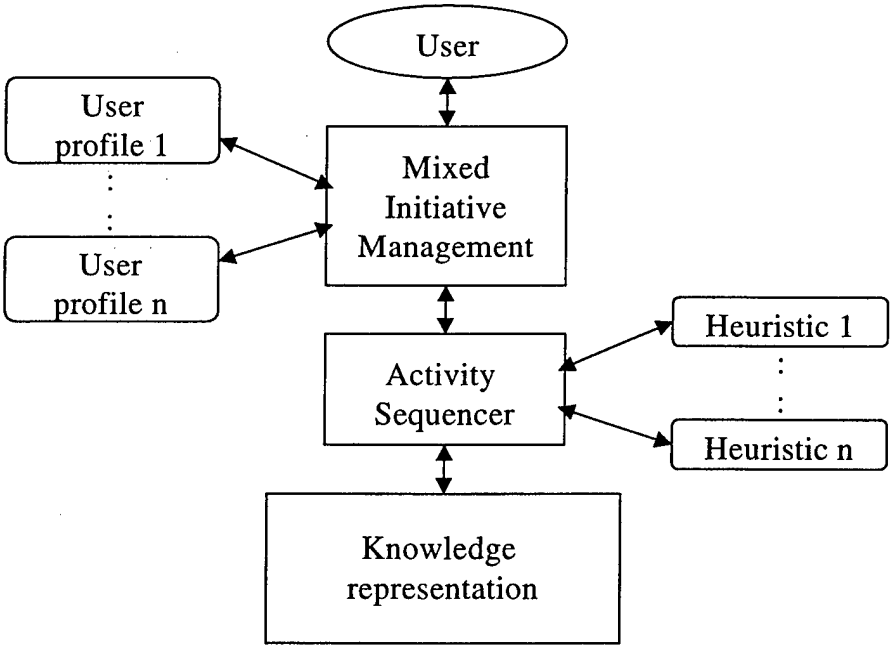


Figure 10: O-OSCAR Architecture

As the user acts on the schedule, the system interprets each action as an attempt to constrain the final aspect of the schedule and so checks the feasibility of the proposed modification. Each user contributes to the final aspect of the schedule by proposing allocations of activities and reacting to the schedule changes induced by the actions of other users. All the users share a common, object-oriented vision of the scheduling domain.

2.2.3 Schedule Construction

Three points mentioned in previous sections need to be reemphasized regarding schedule construction.

- The set of outstanding demands at any point determines the current scheduling problem.
- Efficient use of resources in support of multiple, competing activities is the crux of the scheduling problem.
- Synchronizing the use of resources by activities to satisfy demands over time is at the center of the scheduling problem.

These three statements give a clear sense of purpose when tackling the problem of schedule construction. It is the very nature of the resources and resource constraints that drive a scheduling system towards the development of a schedule that satisfies the outstanding demands. The earliest work on scheduling recognized the power of constraint reasoning to solve the scheduling problem [MF94]. The result of this research was constraint-directed scheduling. Constraint-directed scheduling is the successful identification of constraint types in a domain and the selection of appropriate methods for pruning the search space. It is an approach to problem solving that explores the problem space under the guidance of relationships, limitations, and dependencies among problem objects [JCB98]. These relationships, limitations and dependencies together are known as constraints. The approach requires that these constraints are first

represented, and second, represented in such a way that search techniques can make use of them for guidance [JCB98].

Two general forms of schedule construction exist: *constructive scheduling* and *repair-based scheduling* [CL94]. Constructive scheduling attempts to extend a partial schedule until it is complete. It checks along the way to ensure that the current constructed schedule is valid. This check guarantees that the final schedule is complete and valid. Repair-based methods attempt to iteratively modify a complete, but possibly flawed, schedule to remove conflicts or further optimize a solution. An example repair-based method is *iterative repair.* Iterative repair incrementally reschedules in a manner that minimizes changes to the previous schedule. *Iterative refinement* is a constructive scheduling algorithm that schedules activities in an iterative fashion using various techniques to sequence activities based on, for example, criticality or frequency.

2.2.3.1 The Constraint Satisfaction Problem

The simplest application of Constraint-directed Search is the *finite constraint satisfaction search problem* (CSP) [ET93]. A CSP is defined as follows:

Given:

- *A set of* n *variables* $Z=\{x_1,\ldots,x_n\}$ *with discrete, finite domains* $D=\{D_1,\ldots D_n\}$
- *A set of* m *constraints* $C=\{c_1,\ldots c_n\}$ *which are predicates* $c_k(x_i,\ldots,x_j)$ *defined on the Cartesian product* $D_i \times \ldots \times D_j$. *If* c_k *is TRUE, the valuation of the variables is said to be* consistent *with respect to* c_k *or, equivalently,* c_k *is satisfied.*

Find:

- *An assignment of a value to each variable, from its respective domain, such that all constraints are satisfied.*

An instance of a CSP (Z,D,C) is conceptualized as a constraint graph, G={V,E}. For every variable, $v \in Z$, there is a corresponding node, $n \in V$. For every set of variables connected by a constraint, $c \in C$, there is a corresponding hyper-edge, $e \in E$. A *consistent assignment* of a set of CSP variables, S, is the assignment of a value to each variable in S such that all constraints in the subgraph induced by the variables in S are satisfied.

A graph or map coloring problem is commonly represented as a CSP. Each variable in the CSP is represented by a node in the graph to be colored. Each variable (node) has a domain of three values {red, green, blue} and each constraint (edge) expresses a "not equals" relationship. This problem is depicted in Figure 11.

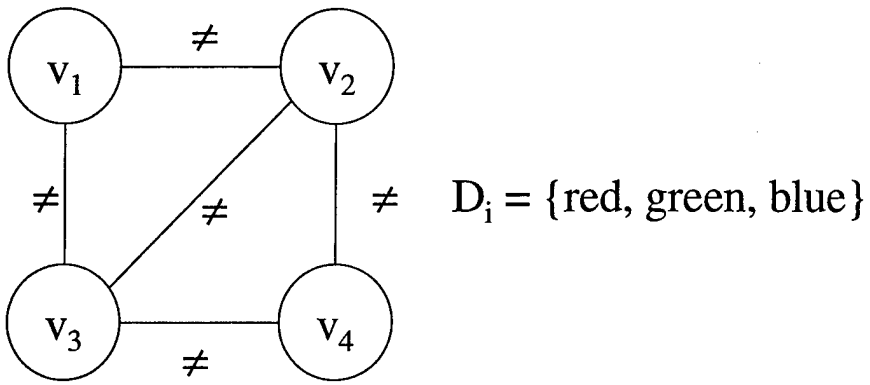


Figure 11: Graph Coloring Problem represented as a CSP

The search for a solution to a CSP can be viewed as a traversal of the problem space consisting of all combinations of variable domain subsets. A solution is a state with a single value remaining in the domain for each variable and no unsatisfied constraints. The mechanism for the

traversal of the problem space is the modification of the constraint graph by the addition and removal of constraints. The constraint graph, therefore, is an evolving representation of the search state.

2.2.3.2 Search Algorithms for Constraint Satisfaction Problems

Backtracking and *constraint propagation* are two basic types of constraint satisfaction search algorithms. Backtracking is synonymous with retraction techniques described in Section 2.2.2.2. Backtracking will always find a solution, but may not be very efficient. Constraint propagation is similar to the propagators also mentioned in Section 2.2.2.2. Constraint propagation is often viewed as a *pre-processing* algorithm in that it can reduce the problem space of a constraint satisfaction problem before conducting search. This reduction occurs by eliminating inconsistent domain values between variables resulting in *consistency sets*. Depending on the problem domain one or more of these consistency sets may actually be a solution to the problem. The need to search for a solution is eliminated. Unfortunately constraint propagation does not guarantee a solution will be found even if one exists. For sufficiently complex problems constraint propagation is often not enough. In these cases it can be applied to the problem prior to conducting backtracking to eliminate inconsistent domain variables prior to searching. These algorithms are described in more detail in the following sections.

2.2.3.2.1 Backtracking

There are many forms of backtracking, but the most popular and simplest algorithm is *dependency backtracking*. This algorithm tries to instantiate each variable, and for each instantiation a consistency check is done to ensure all constraints are satisfied. If all the checks succeed, the next variable is instantiated, otherwise, another instantiation with the next value is

done. If all possible instantiations for a variable fail then a backtrack is done to the most recently instantiated variable.

This technique can be used to solve the graph coloring problem previously presented. The algorithm starts by instantiating variable V_1 with the value red. It then proceeds to instantiate V_2 with red. It identifies that the constraint between V_1 and V_2 (i.e., $V_1 \neq V_2$) is violated. As a result it discards the value red as a possible choice and selects green. All constraints are satisfied so it selects V_3 instantiate next. The algorithm selects value red for V_3 and checks that all constraints are consistent. The final variable is selected and given the value red. Again a constraint is violated (i.e., $V_1 \neq V_4$) so the next value is selected. This continues until V_4 is assigned blue. At this point all constraints are consistent and a solution is found. In this example the backtracking is used to select a different value for a variable (i.e., remove red and try green). It is often the case that the backtracking algorithm needs to go back to a previous variable because no consistent values for the current variable exist. This is where the inefficiency of backtracking may manifest itself. In the end if a solution exists backtracking will find it.

2.2.3.2.2 Constraint Propagation

Constraint propagation is rarely, if ever, used by itself to solve a CSP. Instead the main objective of constraint propagation is to remove *redundant values* from the domain of the variables and remove redundant *compound labels* from the constraints. A value of a variable is a redundant value if its removal does not affect the solution of the CSP. A compound label is the simultaneous assignment of values to a set of variables [ET93]. That is $(<x_1,v_1>,<x_2,v_2>\ldots<x_n,v_n>)$ denotes a compound label of assigning v_1, v_2, $\ldots$,v_n to x_1, x_2,$\ldots$,x_n respectively. The most common constraint propagation algorithm is *arc-consistency*. An arc-

consistent CSP is a CSP in which for all assignments <X,a> there is at least one assignment <Y,b> for each variable such that all constraints are satisfied [ET93].

A common constraint satisfaction problem is *resource allocation*. Resource allocation attempts to assign resources to various tasks. This assignment is affected by the capacity of the resources in the domain. The meeting problem is a good example of resource allocation. Here the "resource" to be allocated is a meeting time. The various "tasks" are the people required to attend the meeting. If Sean, Steve, and David are required to attend and the days each can meet are {1,3,4,7}, {1,3,4,9} and {3,4,9,11} respectively, the goal is to find a day they are all available. Backtracking can be used to solve this, but much iteration may be required. Instead this problem can be preprocessed using arc-consistency. Since a constraint exists that they all must meet on the same day, only days they have in common are considered. Therefore days they do not have in common are removed. The result is the set {3,4} of days that are possible solutions. If one of the people did not have any days in common with the others, there would be no solution to this CSP. This eliminates the computation required to search for a solution.

2.3 Mixed Initiative Scheduling

Chien recognizes there are obstacles hampering the application of scheduling technology to real world problems [SC96]. One of the ways to solve these obstacles is through human involvement in the scheduling process. Scheduling systems need to fit into a wide range of operational contexts. Most scheduling tasks cannot be completely automated. Since this is true resulting schedules need to be easily understood so the user can modify them. In some cases the user required intimate involvement in the schedule construction process. By involving humans the scheduling process can be quicker, higher quality, and easily adaptable to dynamic changes.

A significant need exists for a natural mode of interaction between the user and the system. It is necessary to have a clear and convenient division of labor and control between the user and the system. Commonly the solution provided by the system must be verified and applied with some human intervention. Therefore an interactive, mixed-initiative schedule construction process is needed. Other desirable system characteristics include support for iterative refinement, subjective adjustments by human experts, and planning for interaction points. Fully functional systems are able to integrate scheduling and re-scheduling to support mixed-initiative interactions.

2.3.1 Desired Characteristics of Mixed Initiative Systems

Humans and machines collaborate in the construction and modification of schedules. Humans are better at formulating the scheduling tasks, collecting and circumscribing the relevant information, supplying estimates for uncertain factors, and various forms of visual and spatial reasoning critical to many scheduling tasks [FA94]. Machines, on the other hand, are better at systematic searches of the spaces of possible schedules for well-defined tasks and in solving problems governed by large numbers of interacting constraints [FA94]. Machines are also better at managing and communicating about large amounts of data.

Mixed-initiative scheduling strives to explore the productive syntheses of these strengths of men and machines and build effective schedules more quickly and with greater reliability. To support this synthesis the human must be able to dictate during the scheduling process where and how much to search, while at other times, an automated process may search problem spaces under its own control. This falls under the guise of *search control management*. Burstein and McDermott identify several factors to consider in this regard [BM94].

1. control dialogue
- system support for a variety of styles of graphical user interfaces
- user can express search constraints of many kinds
- users can ask for clarification or elaboration

2. context registration
- system needs to convey where in the problem space the team of human/computer is currently working and who is performing what tasks

3. schedule analysis
- system must provide the user with a set of tools for analyzing fragments of schedules and comparing versions

4. intent recognition
- system can ask for clarification

Before a true synthesis between human and machine is achieved, they must learn from each other. In this way measures such as schedule efficiency and quality can improve over the lifetime of the interaction. This is accomplished by finding opportunities for automated systems to do useful learning. Some of these areas are presented below [BM94].

1. user preferences
- automatically anticipate actions and execute them or inquire whether or not the user wants the action to be done

2. prior schedules and their effects
- users may want to generate new activities by modifying old ones
- case-based reasoning may be used to get schedule level learning into systems
- system could help by indexing and retrieving stored schedules as similar goals are stated for new problems and by recording failures and the conditions that led to them, so that they can be brought to the attention of users if similar schedules are constructed

3. general and domain-specific scheduling knowledge or heuristics
- interactions with the users of the system result in knowledge update and maintenance
- may motivate additional (maybe off-line) clarification dialogues so system can learn from user directives about such things as searching through the problem space, activity parameter preferences under different conditions, etc.

It is advantageous to capture schedules in usable electronic form. Flexible, interactive visualizations of schedules and support information from different perspectives is an effective way to convey information to the user [BM94]. Graphical contexts enhance a human's ability to communicate with the machine.

2.3.2 User-centered Scheduling

Perry was tasked with the job of specifying a standard approach to scheduling military airspace usage [POP92]. Through study of the various locations responsible for conducting this scheduling, several facts were identified. First each area had its own policies and processes for scheduling their respective airspace. Second no standard representation of the scheduling process existed. Most scheduling was conducted with pencil and paper and little auxiliary automation. Rather than trying to capture all of the various policies and procedures that each airspace management location used, emphasis on user interface design of the scheduling system would be the primary means of producing deconflicted schedules. The intent was to add more complex reasoning to the interface as the system matured.

Human schedulers typically resolve conflicts by generating alternatives, assigning priorities, or trying to negotiate mutually acceptable solutions. Because of the distributed, widely differing scheduling strategies inherent in the overall problem, an aid was developed where the user has an explicit role in the scheduling process. This was preferable to the development of a highly complex, fully automated scheduling system. Due to the dynamic rescheduling nature of airspace management, it seemed more effective to provide necessary tools via better user interface mechanisms rather than incorporate explicit knowledge of numerous considerations of the scheduling process. Therefore effort centered on providing useful interface components that facilitate forming and maintaining a schedule regardless of local practices or procedures.

The user interface is the scheduler's primary means of establishing a deconflicted schedule. It is modeled as an interactive Gantt Chart where horizontal areas represent resources being scheduled and the window is divided left to right by time. This allows the scheduler to

focus on a specific time period, yet gain access to distant areas of the schedule as needed. By clicking on an activity duration bar, the scheduler can gain detailed information about the activity.

The human scheduler needs support in maintaining temporal relationships and resource constraints. This is where the system aids the scheduling process. The system maintains a point-based representation of time for a single activity and a symbolic representation of time for links between activities. It was determined that users need to represent time as *before*, *meets*, and *equals* relations. These temporal relations are used when a complex linked mission is planned with multiple activities taking place using multiple resources according to an interdependent sequence of events. The system scheduler defines and maintains these relationships.

Any conflicting activities on a given schedule are highlighted in red. Conflict identification is performed by the system each time an activity state changes (i.e., is either scheduled or moved interactively). The system graphically provides an explanation of why two or more activities are considered in conflict by associating them with connected lines. To aid in manual resolution the user is provided with a pop-up window containing scrollable list of conflicting missions with conflicting field titles in red. Rather than maintaining continuously changing knowledge in the application, the human scheduler is left to resolve those aspects of the schedule that require human judgment. The system's responsibility is to maintain consistency in the schedule while managing a large set of scheduling data.

The system presented here is simple in terms of automation, but powerful in the way it communicates the state of the schedule and how it lets the user interact with the schedule. This is more of a human factors exercise with regard to mixed initiative systems, but it should not be underemphasized or neglected. This system is a good example of exploiting the strengths of the human scheduler and the system. Here the scheduler knows the preferences, policies, and

guidelines of his/her respective airspace and can act with those things in mind. There is no need to capture these rules early on in the system. Likewise the system is good at recognizing conflicts and managing the large set of scheduling data, actions that most humans can not do effectively. This proves that a scheduling system is effective as long as the user is kept in mind.

2.4 Software System Analysis and Design

System level analysis is concerned with what *architectural style* is best suited for a given process and what *design method* is appropriate for designing and implementing a given architecture. Architectural style is defined as the "specific organizational principles and structures" chosen to develop a software system [SG96]. Architectures consider the public details or external properties of *components* in the system. They consider how the components use, are used by, relate to, and interact with other components [BCK98]. Therefore an architecture is an abstraction of a system that suppresses the private details of the components. Multi-agent, client-server and layered architectures are good examples of commonly used organizational principles. Design methods such as object-orientation and agent-orientation are popular forms of design approaches.

The architecture of a system is a collection of computational components together with a description of the interactions among these components. The interactions between components are known as *connectors* [SG96]. Examples of components include agents, clients, servers, and databases. Examples of connectors include agent conversations, procedure calls, event broadcasts, and database protocols.

2.4.1 Architectural Styles

Selecting an architectural style determines a specification for the types of components and connectors to use as well as a set of "combination rules." Bass, Clements, and Katzman assert that several design choices determine an architectural software style [BCK98].

1. A set of component types (e.g., data repository, a process, a procedure) that perform some function at runtime
2. A topological layout of these components indicating their runtime interrelationships
3. A set of semantic constraints (for example, a data repository is not allowed to change the values stored in it)
4. A set of connectors (e.g., subroutine call, remote procedure call, data streams, sockets) that mediate communication, coordination, or cooperation among components

Bass, Clements, and Katzman also present several structural "rules of thumb" for good architectures [BCK98].

- well-defined modules supporting information hiding and separation of concerns
- modules producing data separate from those consuming data
- feature small number of simple interaction patterns

Table 2 provides a listing of common architectural styles.

Table 2: Common Architectural Styles [SG96]

Architecture	Examples
Data-centered systems (repositories)	Databases Hypertext systems Blackboards
Dataflow systems	Batch Sequential Pipes and Filters
Virtual Machines	Interpreters Rule-based systems
Call-and-return systems	Main program and subroutine OO systems Hierarchical layers
Independent components	Communicating processes Event systems

Data-centered architectures are characterized by data stores that are widely accessed by many clients. *Data-flow* architectures strive to achieve reuse and modifiability [BCK98]. This architecture style defines the system as a series of data transformations on successive pieces of input data. Data flow through the system is a dominant characteristic of the architecture. *Batch-sequential* systems, a substyle of data-flow architectures, have processing steps that are independent programs. Each independent program runs sequentially to completion; a batch of data is transmitted to the next program; and the next program begins execution.

Virtual machine architectures are the province of systems designed primarily for portability. A popular example is the Java programming language. It is built on top of the Java Virtual Machine. The Java Virtual Machine allows the language to be platform independent. Another architectural style is the *call-and-return* architecture. This style is used significantly in large-scale software systems and is useful for achieving modifiability and scalability [BCK98]. Four main substyles of this architectural style are in use: *main program-and-subroutine, remote procedure call, layered,* and *object-oriented.*

Independent component architectures consist of a number of independent processes or objects that communicate through messages [BCK98]. Components typically send data to one another, but do not directly control each other. Communication patterns among independent--usually concurrent--processes is central to this style [BCK98]. *Event systems,* a substyle of this architecture, consist of individual components that register with a message manager. The registration describes the information types that they are willing to provide and the types they wish to receive. One benefit of this architecture is that components do not need to know the names and locations of other components in the system. Additionally components can run in parallel and only interact when data is exchanged. The other substyle is the *communicating processes* style. Multiprocess systems such as client-server follow this style. Scalability is the

central goal of this style as it is possible to take advantage of multiple, distributed computing components to share computation, data, or other resources.

2.4.2 Feature-based Classification of Architectural Styles

In conjunction with the general characterization of architectural styles presented in the previous section Bass, Clements, and Kazman present a *feature-based classification* of architectural styles that discriminate styles along certain dimensions: *constituent parts* (i.e., components and connectors); *control issues; data issues; control/data interaction;* and *type of reasoning* .

Control issues describe how control passes from one component to the next and how the components work together temporally. They subdivide control issues into *topology*, *synchronicity*, and *binding time* considerations. The form the control flow of the system takes and the direction the control flows characterize the control topology. A server system has a *star* control topology for example. Synchronicity refers to the dependence one component's actions may have on another components control state. For example *asynchronous* components are unpredictable in their interaction with each other. Binding time determines when the identity of a partner in a transfer-of-control operation is established.

Data issues describe how data move around the system. Like control issues data issues are subdivided into topology, *continuity*, *mode*, and binding time. Data continuity is concerned with the frequency and volume of the data flow in the system. Is the date flow sporadic or continuous? Is it low or high volume? Data mode refers to the manner in which data is made available to components in the system. Examples include objects passing data or agents broadcasting messages.

Control/data interaction issues describe the relationship between certain control and data issues. Control and data topologies are compared to see if they have similar shapes. Control and data *directionality* is considered as well. Whether or not they flow in the same direction (e.g., a pipe-and-filter data flow system) or in opposite directions (e.g., client-server style) is of interest.

Different architectural styles are good for different computational approaches. Some are better suited for local reasoning (e.g., call and return) or nondeterministic reasoning (e.g., independent components). The nature of the system analysis will drive the architectural style selection.

2.4.3 Design Methods

Design methods are associated with the means for designing the components and connectors in a software system. The *object-oriented* methods commonly associated with modern software systems are a good example of a design methodology. Objects provide a powerful and natural abstraction for system development. Recently researchers have advocated *agent-oriented* modeling and design approaches proposing the use of software agents in the construction of complex systems [EK98] [WJK99] [SD99]. Software agents are an extension of objects that encompass all the features of objects (e.g., information hiding, encapsulation), but also include *autonomous*, *proactive*, *social*, *reactive*, and *intelligent* behavior [EK98]. Kendall proposes the use of roles which focus on the "position and responsibilities of an object within an overall structure or system" [EK98]. Furthermore Kendall suggests that a role be played by an agent if "it is to be automated and exhibits proactive, autonomous, and social behavior" [EK98]. The other roles in the system are allocated to people, objects, systems, processes, or organizations.

Design methods do not typically account for the cases where humans interact with the system. It is in the design phase of system analysis and design where these *interaction points* are

considered. In the case where the system designer chooses agent-oriented methods, an approach that considers agents acting in a mixed-initiative context is useful. Hartrum and DeLoach describe such an approach that addresses mixed-initiative concerns as they relate to multi-agent systems [HD99]. They define a *mixed-initiative agent system* as one in which "some of the agents are pure software, frequently interfaced to a database or software tool resource, and some of the agents are effectively human in the loop experts" [HD99]. Central to this definition is the concept of *human/agents*. A human/agent is a team approach that pairs a human expert to an interface agent allowing the human expert to interact with the rest of the system. The human becomes a local resource to the interface agent [HD99].

Multiagent systems are viewed from a domain level and an individual agent level. Mixed--initiative issues are not a factor at the domain level as the human/agent looks to other agents like any other agent interfaced to a tool. The focus on mixed-initiative issues in a multiagent system resides solely at the individual agent level. Therefore the only patterns of interaction occur at the human/agent level.

Hartrum and DeLoach describe three models of interaction between human/agents and the multi-agent system [HD99]. The first model is a *client-server model* with the human as the client. This can be either a *query-based* or *tasked-based model*. In a tasked-based interaction, the human initiates the interaction by submitting a task to its interface agent. The interface agent coordinates with other agents to accomplish the task on behalf of the user and upon completion of the task provides a response to the user.

A second model of interaction is the client-server model with the human as the server. In this model the human receives a query from an agent representing another agent or human in the

system. Although Hartrum and DeLoach do not state that this interaction may be tasked-based, it seems clear that this model supports this notion.

The final model of human/agent interaction is *peer-to-peer*. In this model agents and human/agents cooperate to solve a problem. The human participates in client and/or server roles throughout the cooperative dialogue.

Hartrum and DeLoach identify two main considerations related to individual human/agent design. The first design issue concerns defining the architecture and behavior of the human/agent. Listed below are some typical behaviors a human/agent must exhibit.

1. respond to *ad hoc* inputs from the human
2. accept and process asynchronous communication from other system agents
3. accept and process human responses to queries from other agents
4. send queries on behalf of the human

The second design issue corresponds to the design of the human-computer interface used to facilitate communication between the human/agent and the human. Richly representing the problem domain and allowing a high degree of user flexibility by way of interface manipulation and menu commands are key considerations.

Hartrum and DeLoach also present a list of questions that provide insight into the type of problem support and user preferences that are considered during human/agent design [HD99]. These questions are answered during the system design phase to correctly design a framework for the allowable message protocols in the system.

1. What are the allowable queries and tasks that can be sent to an agent?
2. What input information needs to be supplied with each query or task?
3. What is the syntax of the query and task messages?
4. What information is needed to answer the query or perform the task?
5. What is the syntax of the response messages?
6. For each query and task, what is the appropriate form of information exchange with the human?

2.5 Summary

This chapter addressed several aspects that are crucial to understanding the scheduling problem. Section 2.2 focused on the essential elements necessary for designing scheduling systems. Common characteristics of scheduling problems and methodologies in the large were presented. Section 2.3 addressed the requirements for and benefits of mixed-initiative scheduling. The final section presented some considerations for conducting software system analysis and design. All of these issues are valuable topics when tackling the problem of designing a decision support system for satellite operations scheduling.

III. Methodology

3.1 Overview

As stated in Section 1.3, the goal of this research is to define a methodology for designing a decision support scheduling system for satellite operations. For emphasis the objectives from Section 1.4 are restated as they are critical considerations for developing the methodology.

1. Automate interactions and information sharing between groups
2. Present a visual representation of the mission schedule
3. Schedule recurring operations and maintenance activities automatically
4. Assist human users in submission of non-standard or anomalous activities
5. Resolve conflicting schedule activities automatically or with the assistance of a human scheduler

This chapter outlines a methodology for designing a decision support scheduling system for satellite operations. The first issue the methodology addresses is specifying a general approach for translating the satellite operations scheduling problem into a general schedule problem representation. A formal analysis of the satellite operations domain is required first. The components identified in the domain analysis step are then classified using a general scheduling problem representation. A useful representation is already provided by the systems surveyed in Section 2.2.2. The conclusion of this step organizes the components with respect to the participants involved in the scheduling process. The result is a description of the scheduling process in terms of the essential concrete objects identified in the domain analysis and classification phases. The next issue is the analysis and design of the software system. The selection of an appropriate architectural style and design method for implementing the scheduling system occurs in this step. Section 3.3 describes the criteria used in the selection process. The third issue in the methodology is the selection and implementation of the *problem solving technique(s)* used to produce a schedule. Section 3.4 presents the general properties of problem

solving techniques and provides criteria for selecting the techniques most relevant to the satellite operations scheduling domain. Finally all systems surveyed consider it important that the human scheduler interact intimately with the system. The issues surrounding the presentation of the scheduling process and the process results (i.e., a completed schedule) to the user are addressed in Section 3.5. Figure 12 depicts the basic steps of the overall methodology presented.

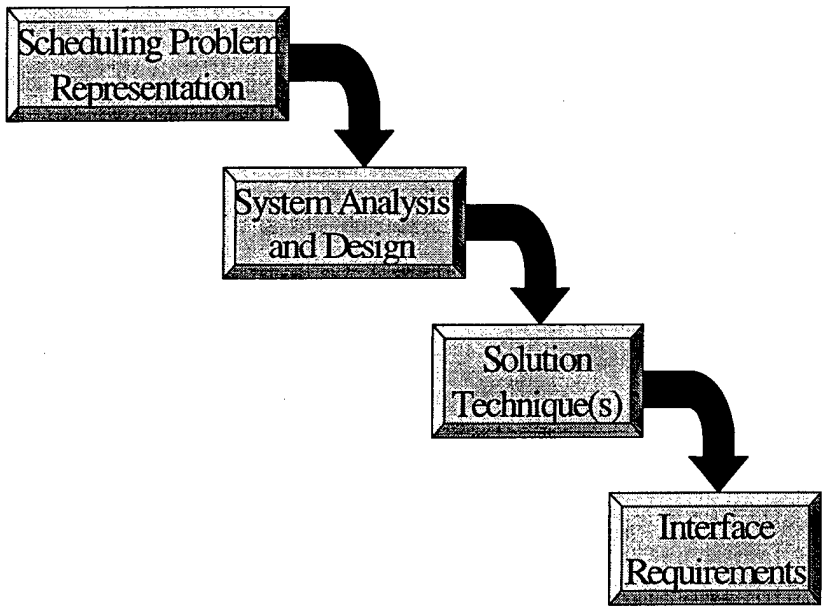


Figure 12: Satellite Operations Scheduling System Construction Methodology

3.2 Scheduling Problem Representation

The scheduling frameworks discussed in Section 2.2.2 take advantage of the large body of theoretical work in scheduling to identify abstract objects common to all scheduling problems. Smith and Becker represented these abstract objects in their Abstract Scheduling Domain Model depicted in Figure 5, page 17 [SB97]. In their work this general representation facilitated the design of object-oriented class libraries that are extended to build domain specific systems.

The schedule problem representation process requires as input the Abstract Schedule Domain Model and domain expertise. The Abstract Scheduling Domain Model is used as the representation of choice because it clearly depicts the abstract objects and their relations. In addition other frameworks surveyed support similar forms of this model. Expert interviews, process analysis, and other methods of gathering domain expertise are good sources of information when conducting domain analysis.

The first phase in this step is a domain analysis of satellite operations. Domain analysis results in the identification of objects and operations in the domain. The second phase uses the abstract objects (e.g., activity, resource) in the Abstract Scheduling Domain Model as *classification categories* to organize the objects and operations. The result of this classification is a specification of the concrete satellite operations objects and operations necessary for completely and accurately representing the satellite operations scheduling problem. The final phase organizes these classified concrete objects with regard to the participants in the system. The result is a process level description of the satellite operations scheduling problem. The overall output of this step is a complete representation of the satellite operations scheduling domain at the concrete object level and at the process level. Figure 13 shows the phases in this step.

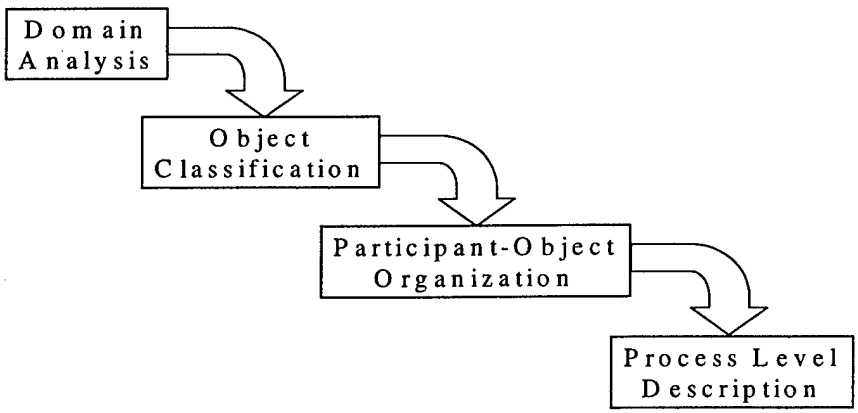


Figure 13: Scheduling Problem Representation

The following steps summarize the approach taken in this section.

1. Identify inputs and outputs of the current process (e.g., schedule request)
2. Select a corresponding abstract object from the Abstract Scheduling Domain Model
3. Compare inputs and outputs to an abstract object's properties and capabilities and find matches (e.g., schedule request = DEMAND)
4. Follow a relation link (e.g., ACTIVITY produces PRODUCT) from the abstract object selected in Step 2 and repeat Step 3 until all relevant concrete objects are identified and classified.
5. Organize concrete objects by process participant (e.g., mission scheduling is responsible for all resources)

3.2.1 Domain Analysis Process

The domain analysis process encompasses the first four steps listed above. Prieto-Diaz describes a process for domain analysis that takes place in three phases: identification of objects and operations, abstraction, and classification.

The result of the domain analysis process is a "knowledge base where objects and operations related to a particular domain are encapsulated and can be used to explain or to talk about events in that domain" [RPD87]. Therefore the goal of this step is to define and discuss the concrete objects and operations related to the satellite operations scheduling domain.

3.2.1.1 Identification of Objects and Operations

Identifying sources of knowledge and information about the domain is a key step in this phase of domain analysis. By analyzing the inputs to and the outputs from the satellite operations scheduling process, domain-dependent concepts related to the Abstract Schedule Domain Model become clear.

One way of identifying process inputs and outputs is through a simple process flow diagram constructed with expert help. As in the example from Chapter 1, mission scheduling receives schedule requests from satellite engineering and maintenance. Mission scheduling in turn produces various schedule products for different internal and external organizations. A simple representation of this is shown in Figure 14.

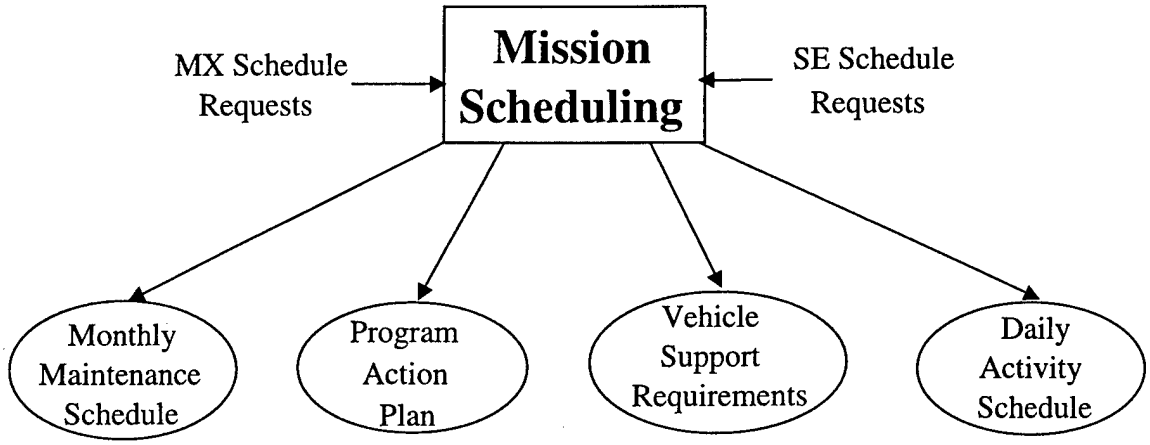


Figure 14: Simple process flow diagram for satellite operations scheduling

There are four outputs in the current scheduling process as depicted by the oval shapes in Figure 14. Each output is a schedule in its own right, but each depicts only a subset of the activities scheduled. The only exception to this is the Daily Activity Schedule. It includes all operations and maintenance activities for the current 24-hour period. With the most relevant inputs and outputs identified, the *intent* (e.g., does it impose a demand) and *content* (e.g., does it list resources, activities) of each is analyzed to determine where they fit in the abstract classification.

Using the schedule request in Figure 16 as an example, the intent of a schedule request is to specify an input goal of the process. The schedule request shares this intent with the abstract DEMAND object and is therefore classified as a DEMAND. The content of the schedule request

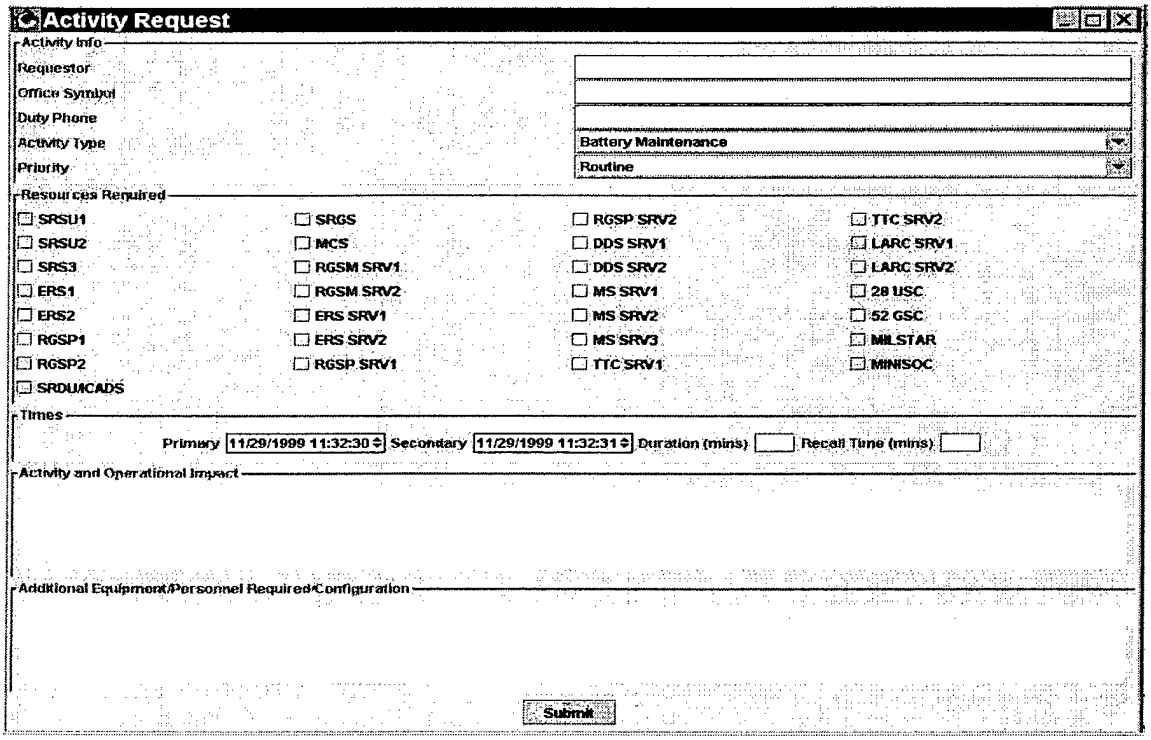


Figure 15: Schedule request

includes a "Resources Required" section that lists all of the possible resources needed for the schedule request. These are obviously specializations of the abstract RESOURCE object. Note that a DEMAND does not have a resources required property. In contrast the abstract ACTIVITY object does have a resources required property. Therefore the schedule request has properties of both a DEMAND and an ACTIVITY. This conflict is a result of the current process implementation. To change the schedule request representation or the model of the process itself involves detailed design decisions that are left to Section 4.2. In the meantime a schedule request is treated as an ACTIVITY or DEMAND where convenient for the purpose of example.

It is possible to determine all specific types of activities in the satellite operations domain by identifying the sources of the schedule requests. In the simple example presented in Chapter 1, both satellite engineering and maintenance submit schedule requests. A mission scheduler can confirm whether or not other sources of schedule requests exist. Satellite operations activities and

maintenance activities are the two basic classes of activities in the example. It is necessary to identify all potential activities in these classes. The example references battery reconditioning and antenna servo maintenance as two specific instances of activities. Therefore battery reconditioning is a *specialization* of ACTIVITY. Figure 16 shows the relationships between the abstract object ACTIVITY and its specializations.

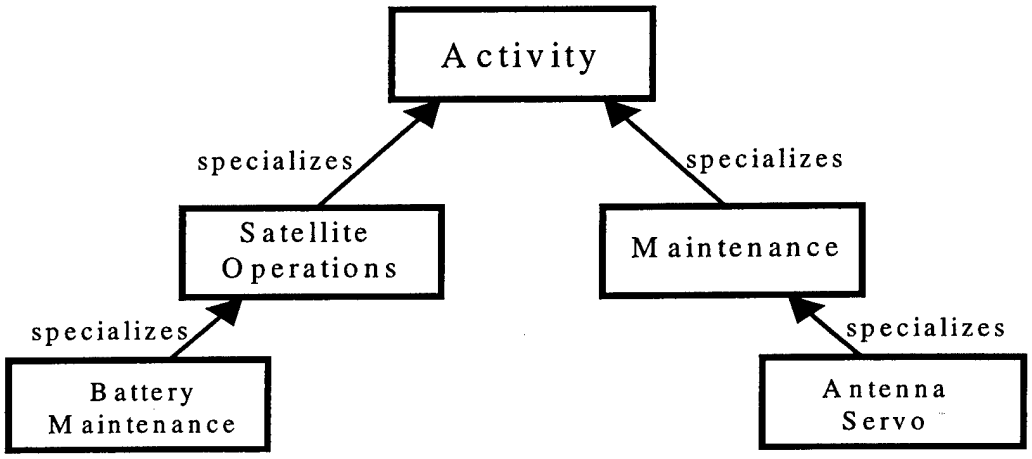


Figure 16: Preliminary Activity Model for satellite operations scheduling

With the sources and the classes of activities identified the next step is to follow a relationship link in the Abstract Schedule Domain Model. Continuing the example, the *requires* link is followed from ACTIVITY to RESOURCE. Following the relationship from one abstract object to another imposes a simple, structured approach for identifying all resources that are required by activities.

An ACTIVITY requires one or more RESOURCE(S). Figure 15 shows all the resources considered important under the heading "Resources Required." The submitter indicates all resources that are required by the activity. In the running example the battery reconditioning activity requires Antenna A. Therefore Antenna A is classified as a resource. Antenna A is represented as a RESOURCE specialization in Figure 17.

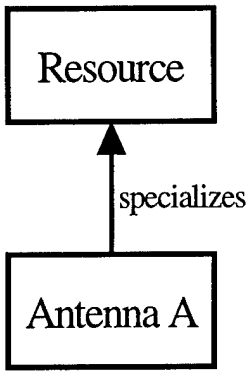


Figure 17: Preliminary Resource Model for satellite operations scheduling

Continue the method of following the relation links from one abstract object to another, identifying concrete objects from the satellite operations scheduling domain, until all abstract objects have corresponding concrete objects in the satellite operations schedule problem representation. This ensures the satellite operations scheduling problem is properly represented and inline with current scheduling theory.

3.2.1.2 Abstraction and Classification

This step is required when trying to find general concepts from many examples in the same domain. General concepts from many sample scheduling domains are already identified by earlier researchers. This step was already conducted by the systems surveyed. Thus the Abstract Scheduling Domain Model provides the abstraction and classification framework needed to categorize the concrete objects and operations identified in the previous step. In this sense a *reverse domain analysis* is conducted to find objects particular to the satellite operations scheduling domain that relate to the abstract objects in the Abstract Scheduling Domain Model. In each step of the previous section the properties and relationships of each abstract object leads to the identification and classification of each concrete satellite operations scheduling object.

3.2.2 Process Level Organization of Scheduling Objects

This section describes a process for organizing and representing the concrete satellite operations scheduling objects identified in the previous phase with respect to the participants in the process. This phase is conducted in two parts. First determine which participants introduce instances of the scheduling objects into the system. Second determine which participants are responsible for managing and processing the scheduling objects in the system. The focus shifts away from the interrelationships between the individual scheduling objects and towards the process participants' responsibilities associated with handling the concrete objects. The result of this analysis is a *process level description* of the satellite operations scheduling domain in terms of specific scheduling objects. This step is necessary for selecting an appropriate architectural style in the next step.

3.2.2.1 Scheduling Objects Introduced into the System

Schedule requests are the only scheduling objects introduced to the process. Schedule requests specify the input goals that drive the system along with any constraints that result from attempting to achieve them. It was shown that satellite engineering and maintenance are sources of schedule requests in the system. Therefore satellite engineering and maintenance require the capability to specify schedule requests and submit them to mission scheduling. Figure 18 depicts a modified version of the process diagram.

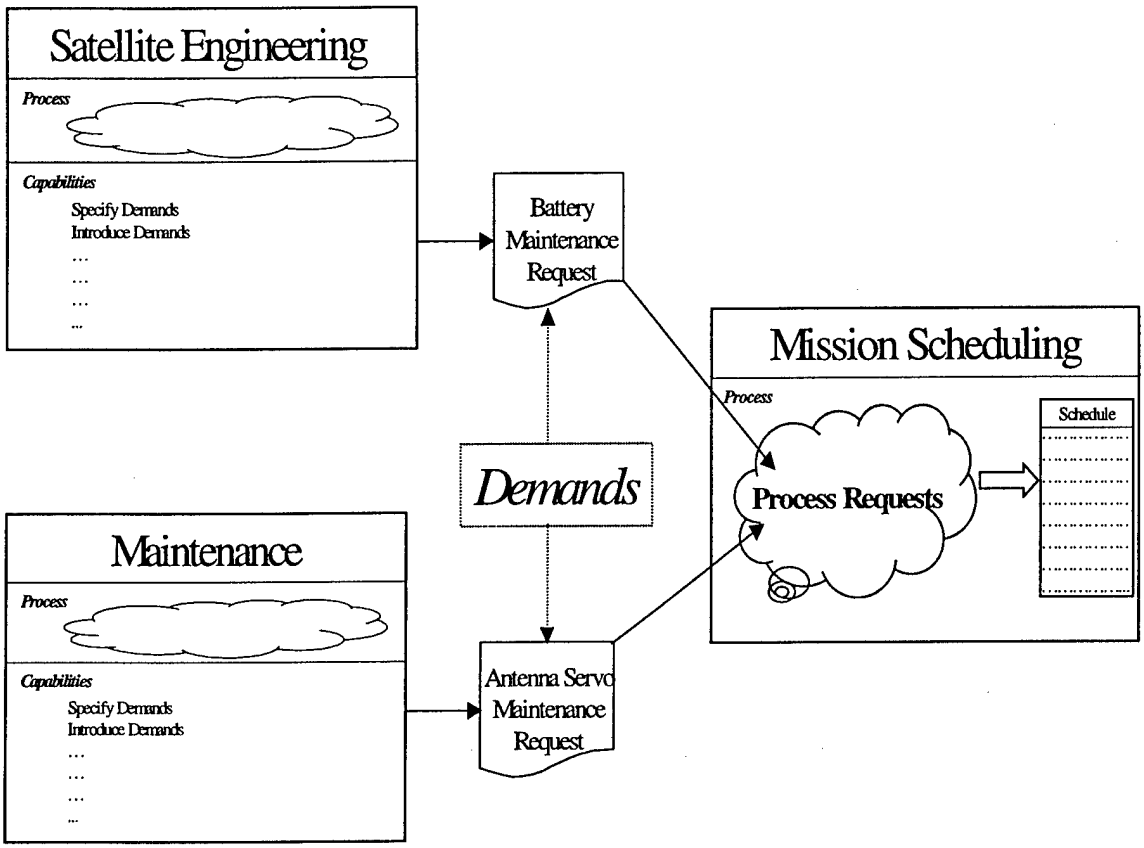


Figure 18: Process Level Diagram modified to include demands and capabilities

3.2.2.2 Schedule Objects Processed and Managed in the System

Schedule object processing refers to the manipulation of a scheduling object with the intent to create downstream scheduling objects. Demands and products are the only scheduling objects that require processing. The result of processing a demand is the selection of a product or products that satisfy the given demand. The product is then processed to select an activity or set of activities that produce the given product.

Scheduling object management refers to the addition, deletion, or modification of scheduling object properties. Products, activities, resources, and constraints are managed. Figure 19 depicts the management of activities, resources, and constraints.

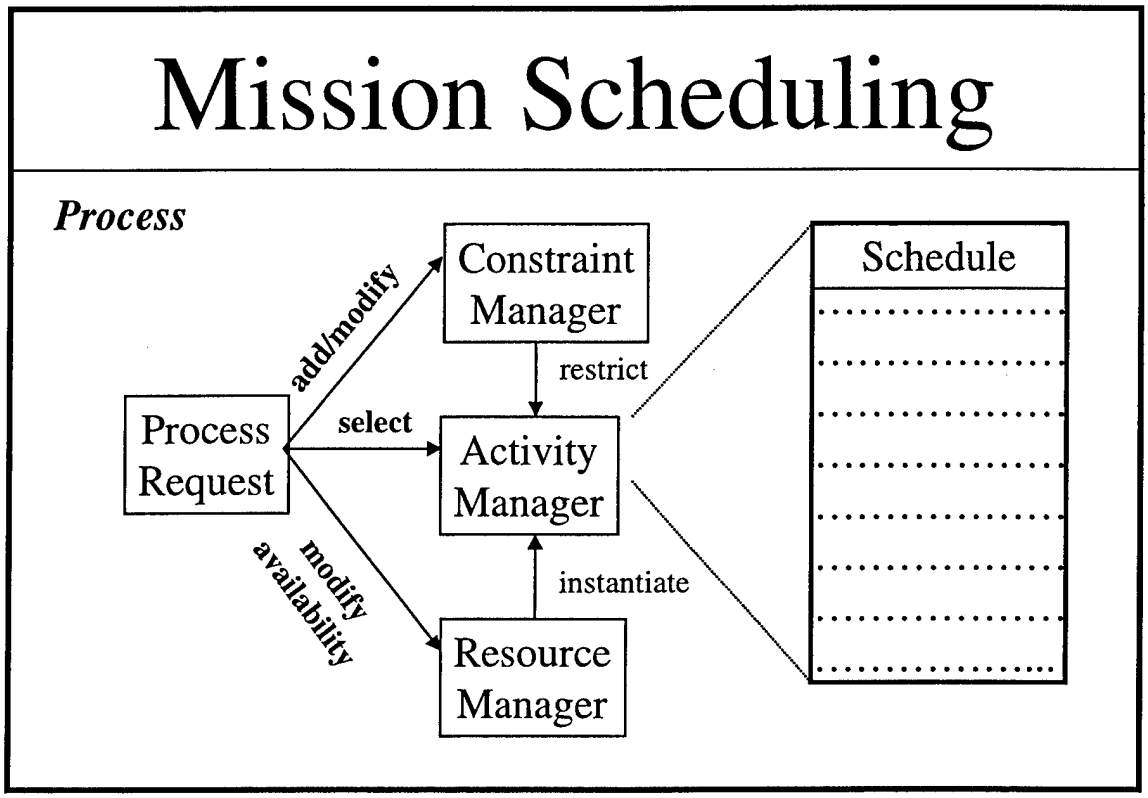


Figure 19: Managing activities, resources, and constraints

Referring back to Figure 18, mission scheduling receives demands from satellite engineering and maintenance. Mission scheduling must in turn process these demands. In the recurring example maintenance submitted an antenna servo maintenance schedule request requiring Antenna A, a start time of 0245 for any activities needed to satisfy the request, and duration of one hour by which time all activities must complete. In this example the start time, the duration, and the resource requirements impose constraints on the activity chosen to satisfy the demand. Figure 20 shows the results of mission scheduling managing the activities, resources, and constraints associated with this demand.

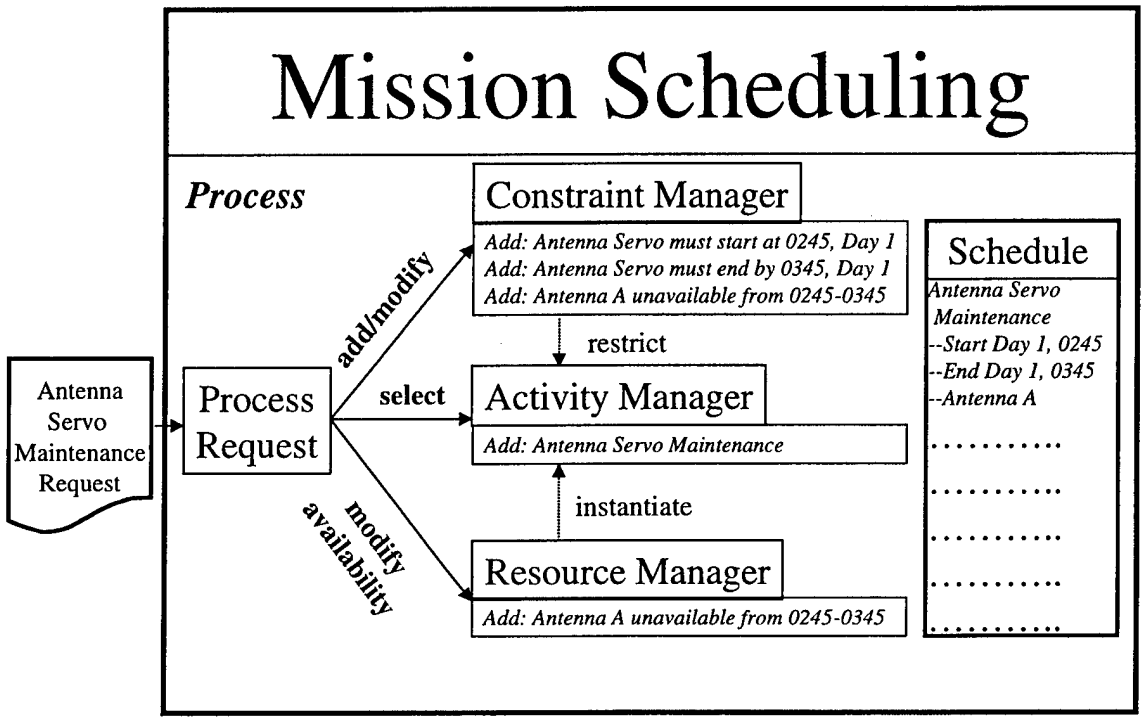


Figure 20: Modified Process Diagram including activities, constraints, resources

The result of the Schedule Problem Representation step is a process level description of satellite operations scheduling. This process level description must be translated into an appropriate software system. This software system specification is a result of the system analysis and design process presented in the next section.

3.3 System Analysis and Design

System level analysis is concerned with selecting an architectural style that is best suited for a given process. For the purpose of this research the system architecture is treated as a collection of computational components together with a description of the interactions among these components [SG96]. The interactions between components are known as connectors. Examples of components include agents, clients, servers, and databases. Examples of connectors include agent conversations, procedure calls, event broadcasts, and database protocols.

Conducting system analysis and design early in the methodology is critical for one particular reason--an architecture manifests the earliest set of design decisions. Bass, Clements, and Katzman suggest several design decisions of interest [BKC98]:

1. An architecture defines constraints on an implementation. The implementation must be divided into prescribed components, the components must interact with each other in the prescribed fashion, and each component must fulfill its responsibility to the other components.
2. An architecture dictates organizational structure.
3. An architecture helps in evolutionary prototyping. The identification of performance problems can be accomplished early and the system is executable sooner in its lifecycle. This may prove beneficial in systems where there is a high-level of user interaction.

The goal of this step is to establish a process for selecting the most appropriate architecture style and design method for the satellite operations scheduling domain.

3.3.1 Selecting an Architectural Style

The process level design described in Section 3.2.2 aids in the selection of an architectural style. It gives a clear organization of the scheduling problem and supports the architecture "rules of thumb" presented in Section 2.4.1. First the process level design provides insight into the "separation of concerns" in the scheduling process. Satellite engineering and maintenance are concerned with submitting demands to the system. Mission scheduling is concerned with processing those demands to produce a mission schedule. Second the process level design represents the "simple interaction patterns" that exist between participants in the scheduling system. By comparing the characteristics of each architectural style presented in Section 2.4.1 to the process level design, certain architectural styles are eliminated and others are chosen as possible candidates.

In conjunction with the simple comparison presented above it is helpful to use a feature-based classification of architectural styles presented by Bass, Clements, and Kazman [BCK98]. They discriminate among styles base on certain dimensions: constituent parts (i.e. components and connectors), control issues, data issues, and control/data interaction. Here is a list of questions that need to be answered satisfactorily to ensure the right architectural style is selected:

1. Would a mission scheduler, satellite engineer, or maintainer be best represented as an agent, client, server or some other component?
2. Should a satellite engineer communicate with a mission scheduler via conversations, procedure calls, event broadcasts, or some other connector?
3. Is control passed from one a satellite engineer to a mission scheduler? Does a maintainer's actions depend on the state of mission scheduler? Or is the control in the system asynchronous?
4. How does data move around the system? What is the frequency and volume of data in the system? How does satellite engineering make data available to mission scheduling?
5. How is the control and data related in the system? Do they flow in the same direction (e.g., a pipe-and-filter data flow system) or in different directions (e.g., client-server style)?

3.3.2 Design Methods

Section 2.4.3 described two of the more common design methods used today. Object-oriented methods are a powerful and natural abstraction for system development. This view is supported by the surveyed systems' choices to use object-oriented methods.

The other design approach mentioned in Section 2.4.3 uses software agents. Agents are autonomous, proactive, social, reactive, and intelligent. Active, goal-directed behavior is an attractive feature for entities in a distributed scheduling system and furthermore supports the central objectives of this research.

Kendall suggests that agents are useful for roles that are "automated and exhibit proactive, autonomous, and social behavior" [EK98]. The other roles in the system are allocated to people, objects, systems, processes, or organizations. The process level design depicted in Figure 18 on page 60 gives a notion of the types of roles that exist in the satellite operations scheduling process. These role types include those played by the satellite engineer, maintainer, and mission scheduler agents. An agent-oriented analysis and design method is an excellent candidate for designing and implementing a decision support system for scheduling satellite operations.

3.3.3 Mixed-initiative System Design Considerations

The manner in which a user expects or needs to interact with the system is a critical system design consideration. The patterns of interaction between the user and the system must be defined. These patterns occur between the automated and manually executed portions of the system. It is important to remember that the user is a critical factor in the success of the overall scheduling system for the reasons elucidated in Section 2.3. This is the part of the system design process where user/system interaction points are identified and designed into the system.

In an agent-oriented context, user/system interaction points are those points where the human interacts with a software agent. Section 2.4.3 presented Hartrum's and DeLoach's approach to the design of human/agents in a multi-agent systems. Following their suggestion the interaction models and design of the allowable message protocols are defined at this point.

Hartrum and DeLoach describe three models of interaction between human/agents and the multi-agent system [HD99]. The first model is a *client-server model* with the human as the client. The submission of a schedule request by a satellite engineer is an excellent example of this model. The task of satisfying the schedule request is accepted by the interface agent and passed

along to the mission scheduler agent. The mission scheduler agent responds to the interface agent with an acknowledgement who in turn notifies the human that the demand was accepted and processed.

A second model of interaction is the client-server model with the human as the server. Suppose two classes of demands exist in the satellite operations domain. The mission scheduler agent automatically schedules one class of demands upon receipt. The other class of demands requires human intervention before they are processed. For this last class the mission scheduler agent accepts the demand from the satellite engineer agent and notifies the human that a demand has arrived that requires attention. The human manually reviews the request and provides proper authentication before it is processed further. Once the human handles the request the demand is processed.

The final model of human/agent interaction is peer-to-peer. In this model agents and human/agents cooperate to solve a problem. The human may participate in client and/or server roles throughout the cooperative dialogue. This type of interaction exists in extended conflict resolution dialogues where the user is querying the details of activities in conflict (i.e., the client role) and changing activity attributes to resolve the conflict (i.e., the server role).

The nature and types of human/agent interactions affect the architecture and behavior of the agent. It is clear from the previous interaction model examples that the human/agent architecture must support both the client and server roles in the satellite operations domain.

The interactions that occur at the agent level are governed by the allowable message protocols in the multi-agent system. Hartrum and DeLoach present a list of questions to answer during the system design phase that ensure correct design of the allowable message protocol framework.

1. What are the allowable queries and tasks that can be sent to an agent?
2. What input information needs to be supplied with each query or task?
3. What is the syntax of the query and task messages?
4. What information is needed to answer the query or perform the task?
5. What is the syntax of the response messages?
6. For each query and task, what is the appropriate form of information exchange with the human?

3.4 Solution Techniques

The decision to design a mixed-initiative scheduling system significantly impacts the choice of solution techniques. It is not the intent of this step to specify an exact solution technique, rather to describe the characteristics of the general solution technique (which may include several algorithms) to use. Having a human-in-the-loop influences the problem solution approach.

Two types of scheduling techniques exist. Constructive techniques start with an empty schedule and incrementally extend a partial schedule until the schedule is complete. Constraint satisfaction techniques fall in this category. Repair-based techniques start with a complete, but possibly flawed, schedule and iteratively modify it by removing conflicts or further optimizing the schedule. A major weakness of the constructive approaches, especially in complex environments, is the inability to incrementally reschedule due to changes in the environment. Instead constructive approaches essentially throw out the previously computed schedule and start over. Incremental rescheduling is a major advantage of repair based methods because it minimizes the change to the original schedule and adapts well to unexpected scheduling conditions.

Lasilla and Smith suggest that mixed-initiative systems support incremental schedule revision during schedule construction and repair [LS94]. In this way scheduling functionality

closely parallels the "inherently reactive nature of scheduling in complex domains" [LS93]. They view schedule construction as a reactive, incremental, constraint-based process. In this sense reactive refers to the system's ability to respond quickly to user interactions. As the user makes changes to the schedule environment (e.g., removes a resource, moves a scheduled activity's start time) appropriate rescheduling procedures are applied to impose changes to the schedule. Iteratively interacting with a schedule enables the user to understand the current problem's requirements, constraints, and other properties [SLB95]. It is the nature of this *solution stability* that supports flexible control and localizes schedule changes from one iteration to the next. The result of this iterative process is that critical tradeoffs (e.g., interactively reassigning resources) are recognized, explored, and resolved [SLB95].

In the case of a conflicting mission schedule, mission schedulers never produce an entirely new schedule because the process is too costly. Instead they rely on expert-based repair techniques. This makes an automated iterative repair approach a promising technique for satellite operations scheduling systems. Unfortunately automated, repair-based methods are domain-specific and are extremely complex. Therefore designing an automated, iterative repair algorithm is very difficult. Two possible approaches to this problem are presented by Prietula, *et. al. and* Numao. Prietula, *et. al.*states that "understanding and supporting scheduling from the perspective of the human expert scheduler" is a sufficiently powerful enough approach to provide significant gains in scheduling efficiency [PHOT94]. They suggest designing a solution technique that produces a schedule sufficiently close to what the human expert can produce. The expert then corrects the schedule. They viewed this as *expert-based iterative refinement.* Numao suggests a *cooperative scheduling* approach whereby the user tells the system what constraints are violated by the current schedule iteration or why one schedule is better than another [MN94]. The system

helps the user produce a schedule by keeping constraints consistent as the user modifies the schedule.

Although designing an iterative repair-based technique is very difficult, designing a construction-based algorithm is not as difficult. The constraint satisfaction techniques presented in Section 2.2.3 are good approaches for constructing mission scheduled from scratch. Of course this approach works at the cost of loosing solution stability.

To summarize, a good solution technique should have the following properties:

- support incremental rescheduling (guarantees solution stability)
- localize solution changes where possible (reduce wide swings from one iteration to the next)
- user must understand the reasoning the problem technique(s) employs
- support manual and automated scheduling algorithms

3.5 Interface Requirements

The interface requirements for a system are a direct result of the interaction points identified during the analysis in Section 3.3.3. Interface design plays a crucial role as it is concerned with the ability of the user to view and modify the current system schedule. The manner in which the schedule is displayed and the manner by which the user interacts with the system to modify the schedule are addressed in this step. The main goal of this step is to make interface requirement and design decisions that allow the system to take advantage of human intelligence and allow the user insight and influence over the scheduling process. This type of interface is needed in addition to the use of traditional artificial intelligence techniques [CB91]

The choice of visual representation impacts the ease with which the user can manipulate the schedule as well as the user's understanding of the automated scheduling decisions that are made. A traditional Gantt Chart or process specific representation is used to display different information based on user needs. As an example satellite engineering has a *satellite-centric* focus

regarding activities. They only wish to view scheduled activities organized by satellite. In contrast maintenance is *resource-centric* in that they are only interested in viewing activities organized by resource.

Viewing schedule generation and modification processes as decision-support issues, enabling the user to directly manipulate specific decisions and problem constraints is fundamental. Extending the running example, satellite engineering decides that a subclass of activities known as *Ranging* needs to be scheduled automatically by the system. Satellite engineering establishes the parameters for this type of activity (e.g., frequency per day, duration per contact). Later satellite engineering decides to schedule an additional Ranging activity per day. This requires a modification to what is essentially a system maintained problem constraint. As these activities appear on the schedule, mission scheduling is curious about these new schedule activities. Mission scheduling displays information about the Ranging activity by double-clicking on the activity representation on the interface. A window displays informing mission scheduling that the activity was automatically generated upon request by satellite engineering. The window further displays all the problem constraints that were used in the decision process for scheduling this particular activity.

The system responds as a result of each manipulation by incrementally making changes consequent to each action [LS94]. The user action of changing an activity parameter is supported by a direct manipulation interface that emphasizes visualization and manipulation of schedules in terms of activities or resource capacity utilization over time [LS94]. Lassila and Smith describe this type of interaction as the "spreadsheet model" [LS94]. Much like a spreadsheet, the user is able to make a schedule change see the results of that change immediately. Continuing the example, satellite engineering originally requested Antenna A for its battery reconditioning activity. Due to unforeseen circumstances Antenna A is unavailable. Mission scheduling modifies

the resource problem constraint of the battery reconditioning activity from Antenna A to Antenna B. As a result of mission scheduling's change, a constraint violation is flagged by the system because Antenna B is already allocated to another activity at that time. The constraint violation identified by the system gives mission scheduling immediate feedback regarding their action.

A functional user interface provides the user a means for looking "behind the scenes" during system scheduling [SLB93]. This enables the user to apply his expertise to the scheduling problem. The ability of the interface to explain automated scheduling decisions, problem diagnosis resulting from user actions, and other human/system interactions enhances the user's ability to improve schedule quality [SLB93]. These concepts are similar to the previous example where mission scheduling checks the properties of the new ranging activity.

In summary, user interface requirements and design for mixed-initiative systems encompass many characteristics.

- allow the user insight and influence over the scheduling process
- display different information based on user needs
- enable the user to directly manipulate specific decisions and problem constraints
- enable user to immediately view results of posting changes to the schedule
- enable user to understand motivations for performing particular scheduling actions
- enable user to modify existing system rules (e.g., preferences, constraints)

3.6 Summary

This chapter presented a methodology that defines the essential components necessary to construct a mixed-initiative scheduling system. The specific design issues associated with each component in the scheduling system were addressed. Section 3.2 specified a general method for using a domain-independent scheduling problem representation to represent the satellite operations scheduling problem. This is accomplished by conducting a domain analysis and subsequently classifying relevant satellite operations scheduling concrete objects under appropriate abstract objects. Section 3.3 addressed architectural style and design methods for

designing a satellite operations scheduling system. The section further described common attributes and properties of architectural styles and design methods and the criteria for selecting the properties most appropriate for satellite operations. Section 3.4 analyzed the properties of common problem solving techniques used to produce schedules. It provided criteria for selecting the techniques most relevant to the satellite operations scheduling domain. Finally Section 3.5 addressed user interface requirements and design in terms of decision support and user/system interaction.

IV. Design Decisions

4.1 Overview

Chapter III presented a methodology for designing a decision support scheduling system for satellite operations. Following the example presented in Chapter 1 and expanding it as needed, this chapter describes each step of the approach in detail, highlighting the actual decisions used to implement a proof-of-concept scheduling system. Section 4.2 follows the general approach presented in Section 3.2.1. It shows the step-by-step process of translating relevant domain-dependent scheduling objects identified from the inputs and outputs of the current process to their representative abstract object counterparts. This step further relates these scheduling objects to the participants in the process resulting in a process level organization of the scheduling objects. Section 4.3 takes the process level organization resulting from Section 4.2 and translates it into an independent-component system architecture using a particular agent-oriented design method known as *Multiagent System Engineering* (MaSE) [SD99]. It describes the decisions made regarding the implementation of the agent-oriented, proof-of-concept system. Section 4.4 presents a problem solving framework that offers the user the ability to iteratively refine an existing schedule or to automatically generate a new schedule using constraint-based schedule construction. Finally Section 4.5 presents interface requirements and design decisions.

4.2 Scheduling Problem Representation

This section applies the general approach presented Section 3.2.1 to the satellite operations scheduling domain. This approach consists of five steps subdivided into domain analysis and process level organization. These procedures and their associated steps are detailed in the next sections.

4.2.1 Domain Analysis Process

The domain analysis process is comprised of the first four steps in the general approach. Step 1 identifies objects and operations. Steps 2 and 3 classify the identified objects and operations. Step 4 is an iterative step that guarantees representative scheduling objects are found for all the abstract objects shown in the Abstract Scheduling Domain Model. This ensures that the satellite operations scheduling problem is properly represented.

Step 1 consisted of an interview with mission scheduling to confirm the validity of simple process flow diagram presented in Figure 14, page 55. All inputs and outputs to the system are accurately represented.

Step 2 of the general approach states that an abstract object is selected next. Once a concept is selected, Step 3 states that the intent and content of each input and output is classified with respect to the properties and capabilities of the currently selected abstract object. The following sections relate the rationale used to classify the objects and operations in the satellite operations scheduling domain.

4.2.1.1 Demands

Demands specify the input goals that drive the system. Properties of a demand include the product that is the object of the demand, release date, priority, and due-date. Referring back to the process flow diagram, only schedule requests support the notion of providing input goals to the system. Figure 15 on page 56 shows a schedule request form. It requires primary and secondary start times (i.e., potential release dates), a duration, which in conjunction with the start time specifies the due date, and a priority (e.g., routine). These requirements are inline with the properties associated with a DEMAND. The other attributes specified on the schedule request form will come into question in later sections as they do not enforce the precise intent of a

demand (e.g., required resources is not a property of DEMAND). This point is also made in Section 3.2.1.1. Based on the process flow diagram and the analysis in this section a schedule request is classified as the only DEMAND specialization in the domain.

4.2.1.2 Products

Following the *satisfied by* link from DEMAND leads to the abstract object PRODUCT. A PRODUCT has as its primary property a list of activities that constitute the processing steps for producing the product. The satellite operations scheduling domain does not produce *tangible* products like in job shop or factory scheduling. Instead it is much like a transportation domain where *intangible* transport missions are produced. In this sense satellite operations scheduling produces *satellite missions* and *maintenance missions*. It is necessary to identify these missions.

Referring to Figure 15 there is a selection list labeled "activity type." In a theoretical sense this label is a misnomer. It should be labeled "demand type" or *mission demand.* This is true since an activity is not a property of a DEMAND. The schedule request is modified to contain only those properties that constitute a DEMAND. The requester only submits DEMANDS to the system, particular activities. Instead of specifying an activity, the requester specifies the mission demand that relates to product or products that represent the object of the demand. Whether a single product or a group of products satisfies the mission demand, let alone the individual activities required, should be no concern of the requester. The system automatically selects or computes the *mission types* (i.e., PRODUCTS) that satisfy the mission demand.

Unfortunately the current process has no concept of a mission type. Instead the process equates the submission of a schedule request to the scheduling of a singular activity. This is a problem for two reasons. First there is no specialization for a PRODUCT in this domain as it is currently represented. Second it is impossible to model a schedule request to schedule more than

one activity. The idea of a mission type is added to the domain and the schedule request is changed to identify a mission demand property for the request.

A mission type is defined as a PRODUCT that consists of set of activities. By modeling a mission type (e.g., battery reconditioning), the need to submit a schedule request for each activity is eliminated. A more precise problem representation results by enforcing the distinction between a demand, a product, and an activity in the satellite operations scheduling domain. This is best expressed by example.

Example Revisited: Satellite engineering has a reference known as a *command plan index*. A *command plan* is a sequence of satellite commands executed to configure a satellite for a desired state. There are several classes of command plans including *spacecraft bus command plans* and *payload command plans*. *Electrical power and distribution subsystem command plans* are a subclass of spacecraft bus command plans. Satellite engineering's request for battery reconditioning is really a further subtype of the electrical power and distribution subsystem command plans. Unfortunately users do not view battery reconditioning as a single mission type (although the simplified example makes it appear that they do). Instead the users view battery reconditioning as the execution of four distinct command plans.

Currently satellite engineering submits a schedule request for each of the four command plans to achieve the battery reconditioning goal. These command plans are listed below.

- Battery Reconditioning Part 1: Battery 1 Discharge
- Battery Reconditioning Part 2: Battery 1 Recharge
- Battery Reconditioning Part 3: Battery 1 Recharge
- Battery Reconditioning Part 4: Batteries 2 & 3 to Auto Charge

There are temporal relations associated with the execution of these command plans. The satellite engineer needs to know these relations in order to specify the schedule requests correctly. If the battery reconditioning mission type changes in the future requiring more, less, or different command plans, or the temporal relations between the command plans change, the satellite engineer needs to be aware of this. Each time battery reconditioning is required this process is repeated in its entirety. If instead battery reconditioning is viewed as a mission type (i.e., an end product) defined by the set of four command plans and temporal relations, then the satellite engineer could submit a battery reconditioning request (i.e. DEMAND) that is satisfied by the battery reconditioning mission type (i.e., PRODUCT). The battery reconditioning mission type is then processed to schedule the relevant command plans (i.e., ACTIVITIES).

Based on the analysis in the accompanying example, a couple of observations are in order. First the schedule request is renamed a *mission request.* This precisely relates the user's action to submitting a demand. Second the activity type label on the mission request is relabeled *mission demand.* This allows a mission demand to be represented by one or more mission types. It lets the satellite engineer focus on attaining a desired operational state rather than considering each action individually. Note that other changes to the mission request form are needed (e.g., no need for "Resources Required"), but the description of these changes is deferred to later sections. Finally all operations mission types are identified from the command plan index. Two exceptions are the *MOMEST* and *ranging* mission types that were identified by reviewing a Daily Activity Schedule. Any command plans that are temporally related are aggregated to form general mission types (e.g., battery reconditioning). Table 4, column 3, lists the operations mission types as Command Plan Subclasses. The focus of the research from this point forward is on satellite

operations rather than maintenance since no information is available to represent maintenance mission types.

Table 4: Satellite Mission Types (PRODUCTS)

Command Plan Index Categories	Command Plan Classes	Command Plan Subclasses
Spacecraft Bus Command Plans	Electrical Power & Distribution System	• Battery 1 Reconditioning • Battery 2 Reconditioning • Battery 3 Reconditioning • Eclipse Monitor • Post Eclipse Batt Commanding
	Attitude Control Subsystem	• Delta-V Repositioning • Delta-V Station Keeping • Spin Control • ACS RAM Dump • Variable Deadband Change • N-Register Change • Pre Eclipse Delta Register Load • Post Eclipse Delta Register Load • ACS ESR Register Retest • ACS Offset Register Reset
	Propulsion Subsystem	• GGA-A Enable/Disable
	Mission Data Message Subsystem	• MDM Coarse/Fine TOD Adjust • Load New MDM System Variables • MDM System to Frequency Hopping
Payload Command Plans		• IR Sensor Calibration • Noise Collect • TMU Readouts • Advanced RADEC I and II Weekly Calibration • Advanced RADEC II Monthly Calibration

4.2.1.3 Activities

The next concept to address is selected by following the *produced by* relation from PRODUCT to ACTIVITY. An activity has properties such as duration, assigned resources, temporal relations between other activities, and status (e.g., unscheduled, scheduled, complete, in

process). In the previous section all of the satellite mission types are identified as command plan subclasses. These subclasses are mission types that are produced by one or more command plans. This observation clearly points to a command plan as equivalent to an ACTIVITY. For instance a battery reconditioning mission type requires the execution of four distinct command plans. An *infrared sensor calibration* mission type requires the execution of the same command plan once a month, on the same day. The *TMU readouts* mission type requires the execution of a single command plan. The point to make is modeling the command plan subclasses as products allows existing relations to be better expressed and possible future relations among command plans to be identified. This leads to the aggregation of existing mission types to form more general classes of mission types. This is made clearer in the following example.

Example: While interviewing a satellite engineer it was discovered that three mission types are temporally related (i.e., they are related to a common goal): Pre Eclipse Delta Register Load, Eclipse Monitor, and Post Eclipse Delta Register Load. They are required, in the sequence listed, during eclipse events. Each of these mission types consists of exactly one command plan. By adding the concept of a mission type to the satellite scheduling problem representation rather than representing every schedule request as an atomic activity, it is now possible to identify a new mission type, call it *Eclipse Management*, that is an aggregation of the aforementioned mission types. At a finer level of granularity it is also an aggregation of the command plans that produce those mission types. This results in a more focused approach to representing what is transpiring in the system. Much like the battery reconditioning mission type example, this alleviates the satellite engineer from knowing all the mission types and their temporal relations during eclipse season.

Several observations are drawn from this discussion. First since "Required Resources" is a property of activities, it is not appropriate to specify the resources required on a mission request form. In practice satellite engineers do not specify resources at all. They expect the mission scheduler to know what resources are required for each mission type. Second duration is another property of activities, not demands. Although the duration can be used to calculate a demand's due date, it is more precise to calculate the due date based on the temporal relations and durations of the activities that ultimately satisfy the demand. This again supports the decision to represent a mission request by the properties of a demand only.

In any mixed-initiative system the designer is required to classify those actions that are automated and those that require user involvement. It is possible to take advantage of the defined mission types to simplify this classification process. Rather than classifying at the activity level it is possible to classify at the mission type level. This classification also requires the expertise of a satellite engineer. Table 5 on page 81 displays the results of this classification process.

All the mission types denoted by asterisks are aggregated mission types composed of other mission types and were identified as a result of this research.

4.2.1.4 Resources

A RESOURCE is *required by* an ACTIVITY. The availability of resources constrains when and how activities execute. One property of a resource is its capacity. A resource can be modeled as a numeric quantity that varies over time as a function of allocating the resource to various activities. It can also be modeled as a discrete-state entity.

The "Resources Required" section of the "old" schedule request form in Figure 15, page 56 is a good place to start identifying resources. Discussions with satellite engineering led to a

Table 5: Mission Type classification- Automated vs. Manual

Mission Type	Command Plans	Mode
Battery Reconditioning	• Battery Discharge • Battery Recharge • Battery Recharge Pt 2 • Batteries to Auto Charge	**Partially automated**: Battery Reconditioning *demand manually submitted* and all *command plans scheduled automatically*
Eclipse Management *	• Pre Eclipse Delta Register Load • Eclipse Monitor • Post Eclipse Delta Register Load	**Automated**
IR Sensor Management*	• IR Sensor Calibration • Noise Collects	**Automated**
Monthly RADEC Calibration*	• Advanced RADEC I and II Weekly Calibration • Advanced RADEC II Monthly Calibration	**Automated**
• Delta-V Station Keeping • Load New MDM System Variables • Advanced RADEC I and II Weekly Calibration	• Single Command Plans	**Automated**
• Delta-V Repositioning • Spin Control • ACS RAM Dump • Variable Deadband Change • Post Eclipse Battery Commanding • ACS ESR Register Test • ACS Offset Register Reset • GGA-A Enable/Disable • MDM Coarse/Fine TOD Adjust • MDM System to Frequency Hopping • TMU Readouts	• Single Command Plans	**Manual**

classification scheme for a majority of the resources listed on the schedule request form. Example classifications include satellites, antennas, antenna strings, servers, workstations, data distribution systems, and several unique classes (e.g., 28 USC, 52 GSC). In addition to these resources other uncommon resources required by non-standard activities are identified from process outputs. Figures 21, 22, and 23 depict these resources and their relations to each other.

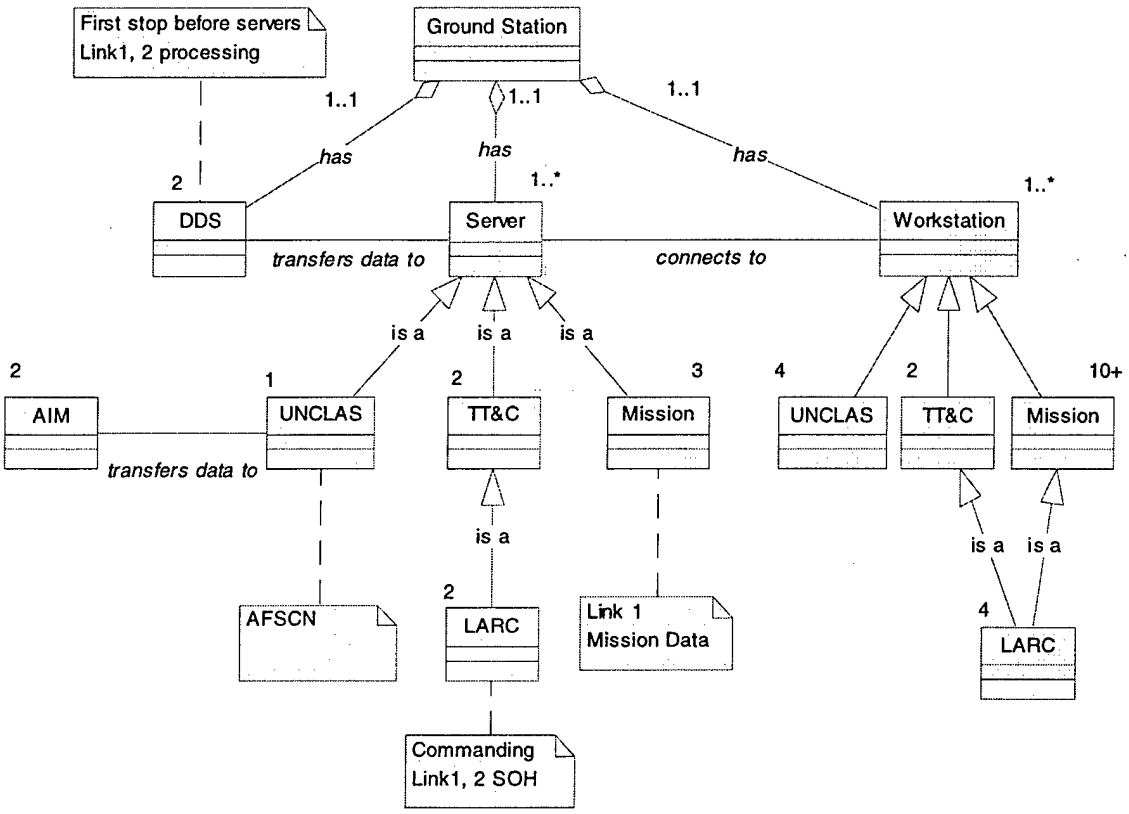


Figure 21: Satellite Operations Ground System

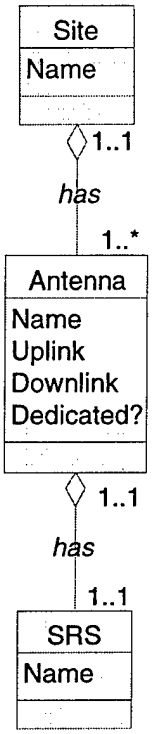


Figure 22: Satellite Operations Antenna

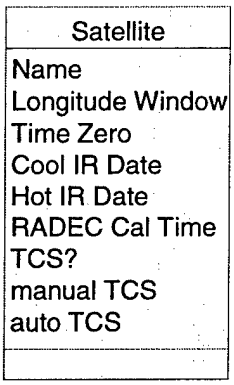


Figure 23: Satellite Operations Satellite

It is worthwhile to note that Figures 22 and 23 illustrate domain-specific properties of interest exist for different resources in the domain. Most of these properties were identified from the operational parameters that satellite engineering uses to decide the need for mission requests. For example a mission request for a *delta-V station keeping* mission type is submitted five to

seven days prior to a satellite crossing a ±1 degree longitudinal boundary as specified by a satellite's *longitude window* property. Other resource properties characterize a resource's functionality. The properties are needed to ensure that activities are instantiated with appropriate resources. For example most activities require an antenna that is capable of *uplink* and *downlink* communications while others only need downlink communications.

It is a requirement to model these resources from the standpoint of their availability and physical structure. All resources in the satellite operations domain are modeled as discrete-state resources. This means that each resource is characterized by a set of possible discrete values. In this case all resources are either *busy* or *idle*. In addition all resources are *reusable*. Once an activity is completed, all assigned resources are released for use by other activities. Although all satellite operations resources can be modeled as atomic resources, using an aggregate resource model captures the hierarchical structure of the resources. These aggregate classes are derived from the resource categories described earlier such as servers and antennas. These classes are well represented by the class diagrams in Figures 21, 22, and 23. A benefit of using aggregate resources to model the domain 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.

With the resources and their respective models identified it is possible to organize the activities by the resources they require. It is helpful from a system design perspective to classify those activities that have the same or similar resource requirements. This allows subclasses of activities to be modeled.

There are two general resource-centered classifications for activities. Both classifications are related to the type of antenna required by the activity. The first classification considers the

communication characteristics of an antenna. Some activities require uplink/downlink communication and others only need downlink communication. Activities associated with the ranging and MOMEST mission types require downlink only. All other activities require uplink/downlink communications. The second classification is related to whether an antenna is dedicated (available solely for squadron operations) or part of the Air Force Satellite Control Network (squadron competes with other users for time). Additional resource requirements are needed if an AFSCN antenna is to be used. An AIM communication device, UNCLAS server, and UNCLAS workstation are required in these cases. Note that all AFSCN antennas are uplink/downlink. The resource requirement templates are presented below:

- Downlink Only: satellite, antenna, antenna string, DDS, TT&C server, TT&C workstation
- Uplink/Downlink: same as Downlink Only, but with an uplink/downlink antenna
- Uplink/Downlink (AFSCN): same as Downlink Only, but with an uplink/downlink antenna, an AIM, an UNCLAS server, and an UNCLAS workstation

4.2.1.5 Constraints

A constraint restricts the set of values that are assigned to decision variables of an activity. Demands, products, and resources impose the constraints in the system. Example constraints were described in Section 3.2.2.2. Constraints of interest in this domain are temporal constraints related to the release date of the mission requests and the constraints between command plans in a mission type. Also resource-availability and resource-compatibility constraints are important. The point to make is no problem exists as long as there are available resources for a particular activity. The problem solving technique handles the case where one or more resources are unavailable. That is the topic of section 4.4.

4.2.2 Process Level Organization of Scheduling Objects

Up to this point design decisions were made regarding what satellite operations scheduling objects are central to representing the satellite operations scheduling problem. Specific instances of demands (mission requests), products (mission types), activities (command plans), and resources were identified. Appropriate methods for classifying activities by mode (i.e., automatic or manual) and resources required were presented. Modeling resources as aggregates of discrete state resources was also addressed. These decisions aid later steps in implementing these activities and resources in a proof-of-concept system. What remains to be accomplished in the Schedule Representation Problem step is to relate these concrete scheduling objects in terms of the participants in the system. The result of this analysis is a process level description of the satellite operations scheduling domain in terms of specific satellite operations scheduling objects.

4.2.2.1 Scheduling Objects Introduced into the System

It was stated in Section 3.2.3.1 that only satellite engineering and maintenance introduce demands into the system. Satellite engineering and maintenance must have the abilities to specify mission requests and submit them to mission scheduling. Satellite engineering is responsible for submitting mission requests that are satisfied by the manual mission types identified in Table 4. The mission requests that are satisfied by automatic mission types are generated as required by the mission scheduler agent. No human intervention is required.

4.2.2.2 Scheduling Objects Processed and Managed in the System

Mission scheduling processes mission requests and mission types (i.e., satellite missions and maintenance missions). Processing a mission request involves adding the constraints imposed by the mission request to the system. In the running example the release date specified for the

battery reconditioning mission imposes a constraint on the start time of the first command plan scheduled to satisfy it. Mission type processing involves adding any constraints that the mission type imposes on the system as well as identifying the command plans responsible for producing the mission type.

Mission scheduling is also responsible for managing command plans, resources, and constraints. Constraint and command plan management is implied above since a result of processing mission requests and mission types is the addition, modification, and deletion of system constraints and scheduled command plans. Once mission scheduling identifies the command plans required for the mission type, it instantiates the command plans' decision variables and assigns the necessary resources. The assignment of the resources to a command plan imposes constraints on the system by making those resources unavailable for the time they are assigned to the command plan. Finally the mission scheduler resolves any schedule conflicts. This scenario shows that the system's problem solving techniques reside centrally at the mission scheduler. Figure 19 on page 61 depicts the scenario described here.

4.3 System Analysis and Design

This step requires the selection and application of an architectural style and a design method to the design of a proof-of-concept decision support system for scheduling satellite operations. The result of this step is a fully designed system ready for implementation.

4.3.1 Selecting an Architectural Style

The comparison between the architectural styles presented in Section 2.4.1 and the process level design from Section 3.2.3 as well as the use of feature-based architecture

classification results in the following candidate architectural styles: data-flow and independent components. The justification for eliminating other architectural styles follows.

Data-centered architectures are eliminated since in the satellite operations scheduling process no multiple processes access a single data store. Call-and-return architectures are removed from consideration also. They are typically dominated by "order of computation" that is normally controlled by a single thread of control [BCK98]. Because mission requests are submitted *ad hoc*, there is no order of computation. In addition multiple processes (i.e., satellite engineering, maintenance, and mission scheduling) exist in a distributed setting so it is unlikely that a single thread of control can be used to handle all computation. Finally virtual machine architectures are useful when a computation is designed, but no machine exists on which it can run. The critical nature of satellite operations guarantees the machines used to execute scheduling computations are explicitly declared. There is no need to use an interpreter, for example, to ensure compatibility between the underlying machine and the scheduling system software.

The next step is to choose either the data-flow or the independent component architecture. Using feature-based classification of architectural styles, it is possible to make this selection. The following table represents the features of both the data-flow and independent component architectural styles.

Based on the classification listed in Table 3 the independent component architecture is the best choice for the proposed scheduling system. First the distributed nature of the participants involved allows the system to be structured as loosely coupled components. Second message passing is sufficient as an interaction mechanism given the simplicity of interaction depicted in Figure 18 on page 60.

Table 3: Feature-based Classification of Architectural Styles [BCK98]

	Data Flow	Independent Components
Constituent Parts		
Components	Transducer	Processes or Objects
Connectors	Data stream	Message protocols
Control Issues		
Topology	Arbitrary	Arbitrary
Synchronicity	Asynchronicity	Synchronous, Asynchronous, opportunistic
Data Issues		
Topology	Arbitrary	Arbitrary
Continuity	Continuous low volume or high volume	Sporadic low volume
Mode	Passed	Passed, shared, multicast
Control/Data Interaction		
Isomorphic Shapes	Yes	Possibly
Flow Directions	Same	If isomorphic, either

4.3.2 Design Methods

The decision to use agent-oriented design is due to the favorable characteristics that agents possess in addition to the standard object-oriented advantages. The autonomous, automated, and proactive behaviors are well suited for a decision support system. The choice to use MaSE is a result of past success using the methodology and the clear, succinct nature used to describe the design process. The steps in the MaSE methodology below domain level design are applied to one agent as an example of its use. The complete design history for the proof-of-concept system is not described here.

4.3.2.1 Domain Level Design

The first MASE step is a domain level design that identifies the agent types and the possible interactions between the agent types. The interactions between agents are known as *conversations* and they are described *by coordination protocols*. Coordination protocols describe the sequence of allowable actions that take place between agents during conversations. Role

models are used to identify agents and their conversations because roles emphasize interactive behavior and roles work together to accomplish goals. Users take on roles in the system, but this discussion concerns the agent level design step. From the examples and figures presented to this point (e.g., Figure 1), the system roles and interactions (or conversations) are identified.

1. Four agent types
   - Mission Scheduler Agent
   - Maintenance Agent
   - Satellite Engineering Agent
   - Orbital Analysis Agent
2. Three agent-to-agent conversations
   - submit (i.e., submit a mission request)
   - retrieve (e.g., get antennas with visibility from orbital analysis)
   - update (e.g., update maintenance agent when schedule changes)

MaSE uses three *Agent Modeling Language* (AgML) diagrams to represent the agents, conversations, and protocols in the system. Agents are described using *agent diagrams*. Agent diagrams list the services an agent provides as well as the goals of the agent. Figure 24 shows the agent class diagrams for the mission scheduler and satellite engineer agents. It depicts the valid conversations between these agents and identifies the agent that initiates the conversation as well as the agent that responds.

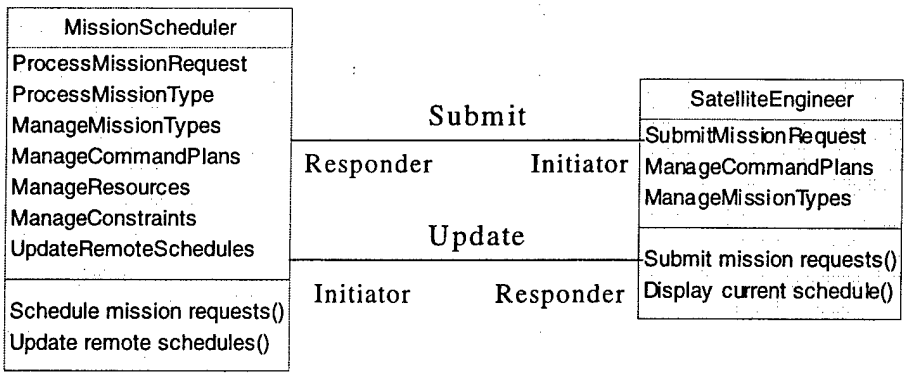


Figure 24: Agent class diagrams with conversations

AgML also represents the types of allowable conversations that exist in the system as *communication hierarchy diagrams*. It describes the base classes of conversations and the specializations of the base class conversations. The *submit* and *update* conversations from Figure 24 and their associated subclasses are represented in Figure 25.

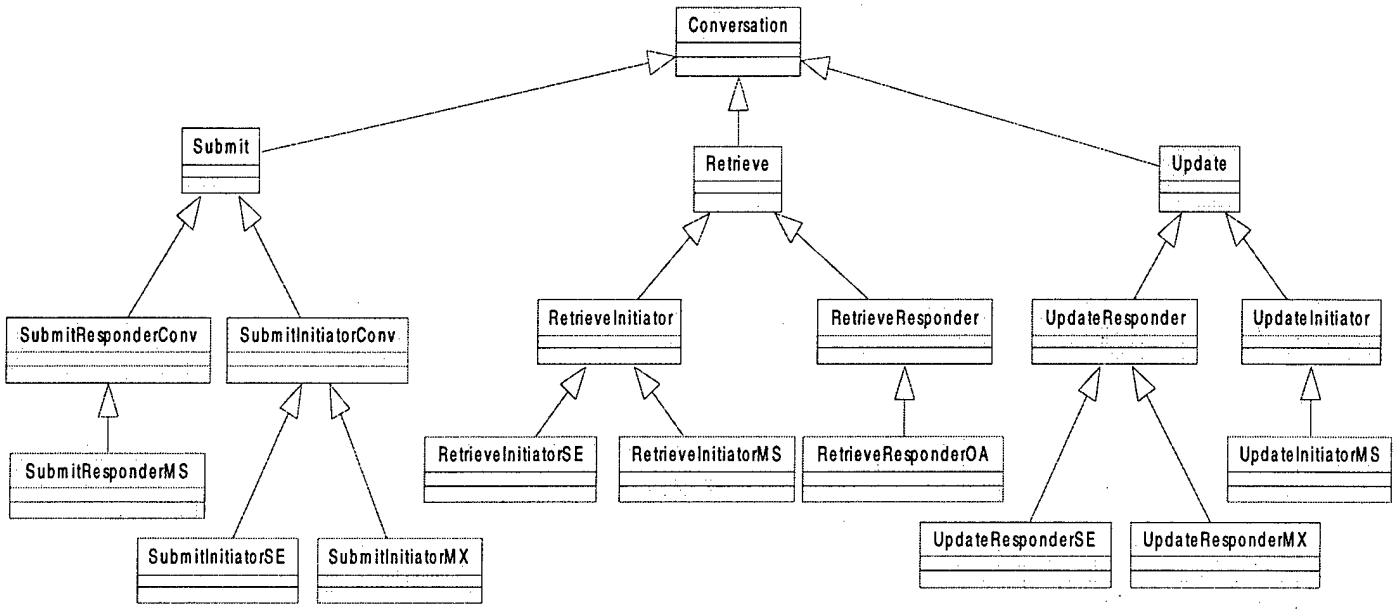


Figure 25: Communication Hierarchy Diagram

The final AgML diagram used during domain level design is the *communication class diagram*. The communication class diagram is a finite state machine that represents all the states an agent is in during a conversation. It embodies the communication protocols that govern behavior during a conversation. There is a class diagram for each agent role. For example all the states that the mission scheduler agent is in during the submit conversation is shown in Figure 26. The intent is to show a simple implementation of an agent's conversation states.

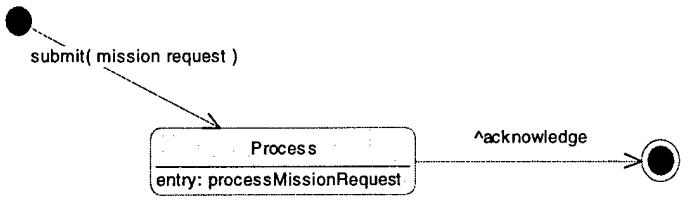


Figure 26: Communication Class Diagram for submit responder

4.3.2.2 Agent Level Design

Agent level design is concerned with the architecture for each agent in the system. Here it is appropriate to consider the human/agent paradigm in designing the architecture for agents that have human resources. This does not affect the orbital analysis agent as it is an information agent and does not require interaction from a human.

The first step in this level of design is to map actions identified in agent conversations to internal components. In Figure 26 the mission scheduler agent has a *process* state that requires the agent to call the *processMissionRequest* method upon entry. This method interacts with several data structures internal to the mission scheduler agent. The mission request is transformed into its corresponding mission type. The *processMissionType* method then transforms the mission type into its corresponding command plans. This is accomplished by a *mission type manager* that is a data structure containing the command plans and temporal relations associated with each mission type in the system. These command plans are then scheduled by the *manageCommandPlan* method which works in coordination with the *manageResources* method to instantiate each command plan with the appropriate resources. It is the *resource manager* data structure that keeps record of the resources in the system and stores the times each resource is assigned. This same logic is applied to the other conversation subclasses listed in Figure 25.

The second step in this level of design is the definition of data structures used to represent input and output from the agents. Only one message object is defined for all conversations. It can

carry all conversation results. The agents have knowledge of the message structure and only access the data structures appropriate for the given conversation type.

The final step defines additional data structures internal to the agent. These data structures represent the data flow between components internal to the agent. The only data structures recognized in this domain are mission requests, mission types, command plans, and resources. These are passed from component to component based on the level of processing taking place in the agent.

4.3.2.3 Component Level Design

The mission scheduler has the ability to process and manage scheduling objects because of the internal components it possesses. These components were alluded to in previous sections. For example the agent has a mission type manager that is responsible for decomposing mission types into corresponding command plans. It has a resource manager that allows the user to add resources to the system or modify the availability of resources in the system. Each agent in the system has internal components based on the operations (i.e., introducing, processing, and managing) performed on the domain scheduling objects.

4.3.2.4 System Design

System Design requires specifying the type and number of agents as well as the agents' physical locations and conversations in which they participate. Four agents are required in this system:

- mission scheduler agent
- satellite engineer agent
- maintenance agent
- orbital analysis agent

Physical locations are arbitrary as they may be placed anywhere that is most useful and easy to manage. The implementation and successful execution of the system are not dependent on agent location.

4.3.3 Mixed-initiative System Considerations

Section 3.3.3 states that the system designer needs to identify the interaction points between the human and the interface agent. Interaction points occur between the user and the agent. The MaSE process identified four agents in the system, but the orbital analysis agent is an information agent and is not designed for human interaction. This step declares the specific interaction models and interaction points for the mission scheduler agent and the satellite engineer agent.

4.3.3.1 Mission Scheduler Interactions

The human mission scheduler has four primary interactions with the mission scheduler agent. First the human can query the details of a scheduled or unscheduled activity. This interaction follows the client-server model with the human as the client. Second the human scheduler can resolve a schedule conflict. This interaction follows the previous model as well. The agent requests that the human modify the properties of a scheduled activity to resolve a conflict. If this process requires the successive modification of more than one activity then this model of interaction is more like the peer-to-peer model where the human acts as both a client and a server over the course of the extended dialogue. Another interaction point exists where the human scheduler submits a mission request. This interaction follows the client-server model with the human as the client. This could occur in the case that a maintainer or satellite engineer agent is unavailable. Finally the mission scheduler can add or delete resources from the domain. Again the user is taking the role of the client in the client-server model.

4.3.3.2 Satellite Engineer Interactions

The human satellite engineer has four primary interactions with the satellite engineer agent. First, like the mission scheduler, the satellite engineer can submit a mission request. The same respective interaction model applies. Second the user can query the details of a scheduled activity. This too is like the mission scheduler. Third the user can modify the command plan library and the mission plan library. This is the client-server model with the user as the server. This interaction requires the user to supply information regarding the addition, deletion, or modification of command plans and or mission types in the libraries. Finally the satellite engineer is responsible for updating the operational parameters of the satellites in the constellation (e.g., longitudinal window). Although the mission scheduler is responsible for maintaining the resources in the domain, the satellite engineer is responsible for anything related to the individual satellites. The satellite engineer acts as a client in that it adds, deletes, and modifies the satellites in the constellation. These changes are communicated by the satellite engineering agent to the mission scheduler agent's resource manager.

4.4 Solution Techniques

Section 3.4 identified four characteristics that are desirable for designing a mixed-initiative scheduling solution technique.

- support incremental rescheduling (guarantees solution stability)
- localize solution changes where possible (reduce wide swings from one iteration to the next)
- user must understand the reasoning the problem technique(s) employs
- support manual and automated scheduling algorithms

Two elements are a part of the proposed solution technique. The first element is human-centered. It places the responsibility on the user to incrementally reschedule in situations where repair is required (i.e., schedule conflicts). This method is attractive because the design of

automated, repair-based algorithms is highly complex and domain specific and it gives the user some control over the solution. It gives the system (i.e., the human/agent team) the ability to incrementally schedule and localize solution changes. The only changes that are made are a result of user action. This implies that the user will understand the reasoning behind the system's actions because they are a direct result of the user's actions. This approach satisfies the requirement for the system to support incremental rescheduling and local solution changes. Human expertise is used to compute a valid mission schedule and system "expertise" (i.e., maintaining consistency) is used to present the result of each user action.

The second element of the problem solution technique is computer-centered, constraint satisfaction based. It is possible for scenarios to occur where the schedule revision dialogue between the human and the agent might be extensive. Unless the agent can offer suggestions that direct the user to select the actions that will reduce or eliminate conflicts in the schedule (usually a characteristic of more mature systems), it may take the user many iterations to "get it right." At that point the user might wish the system to automatically generate a new schedule. The cost of this approach is the loss of localized scheduling changes, solution stability, and user understanding regarding the form of the final schedule, but a solution *is* highly probable. Remember that one of the characteristics of mixed-initiative systems is the ability for the user to determine how involved the system should be in finding a solution (i.e., purely automated vs. user-directed only). The tradeoff is a valid solution instead of user control over the solution. A mixed-initiative, iterative approach coupled with a fully automated constraint satisfaction approach captures the desirable characteristics (in one way or another) that best suits the nature of a mixed initiative, decision support system.

4.5 Interface Requirements

The interaction points identified in Section 4.3.3 are the basis for determining the user interfaces required for the system. There were eight interactions points identified between humans and agents in the system. Two of the interaction points are related to submitting mission requests. Only one interface is required to support this type of interaction since both the satellite engineer and mission scheduler are required to provide the same type of information. The interface accepts all data associated with scheduling demands (e.g., priority, release-date, product-type).

Another class of interactions allows the mission scheduler and satellite engineer to query activity details. The query returns the details of the activity and the reason it was generated. The result is slightly different though depending upon the user. The satellite engineer views a display of activity information that can not be edited, whereas, the mission scheduler can edit the start time, priority, and assigned resources of the activity. This query ability supports several of the characteristics desirable of mixed-initiative interfaces. First it gives the users insight into why the activity was scheduled. Second it gives the mission scheduler influence over the scheduling process. Finally it displays information differently based on user needs and enables the mission scheduler to manipulate specific decisions and problem constraints.

Conflict resolution is an interaction that is solely the domain of the mission scheduler. The interface displays the results of the scheduling process in a manner that highlights any conflicts that exist. A Gantt Chart is used to display scheduled activities and graphical links between conflicting activities are used to identify conflicts. Like the activity query interaction the mission scheduler modifies activity parameters to eliminate the conflict. As soon as the activity's parameters are changed, the display is updated to reflect the result of the user's action. This

enables the user to immediately view the result of posting changes to the schedule. Again it allows the user to directly manipulate the specific decisions and problem constraints that exist in the current schedule. In this scenario the interface also supports the choice of the mission scheduler to allow the system to automatically resolve any conflicts in the schedule using its constraint satisfaction problem solution techniques.

The final interaction point of interest for the mission scheduler is the ability to add and delete resources in the domain. The resource manager component of the mission scheduler agent is responsible for updating its resource lists based on the user's action. The ability to update resources is necessary to ensure the problem constraints (i.e., the availability and types of resources) represent real world operations. This is akin to the user modifying existing system rules.

The satellite engineer has the ability to update the command plan and mission type libraries. The command plan library contains all atomic activities that can be scheduled. The mission type library contains all mission types that are defined in the system. The command plan and mission type definitions are a subset of the system rules used for scheduling. They contain the greatest amount of preference and constraint information in the system. An interface is required that enables the satellite engineer to add, modify, and delete command plans and mission types in the system. Like the resource manager interaction this interaction ensures the user has control over existing system rules.

Updating satellites in the constellation is the last satellite engineering interaction. Much like the resource manager interaction of the mission scheduler, the system reflects the current real world state of the satellites and the constellation. It must correctly represent the operational

parameters of the satellites as well as the number of satellites in the constellation. An interface that supports the users' changes to the system's rules is needed.

The interaction points are not the only properties that drive the system's interface requirements. In addition to the aforementioned interfaces, one is needed to communicate the system actions to the user as they happen. These actions include automatic mission scheduling, conflict resolution, mission and command plan updates, and other issues relevant to scheduling operations. An event log is used to aid the user's understanding of system operations.

4.6 Summary

This chapter detailed some of the major design decisions that were used to design and implement a proof-of-concept, decision support system for satellite operations scheduling. Section 4.2 identified the domain scheduling objects and precisely modeled the properties of each object. It also organized them with respect to the process participants in a way that clearly specified the responsibilities of each process participant. Section 4.3 used the process level description from Section 4.2 to choose an independent component system architecture. The choice was made to use agent-oriented design methods for the components. MaSE was used to analyze and design the system. The proper type and number of agents was identified as well as the allowable conversations between the agents. In addition agent and component level design was conducted for the mission scheduler and satellite engineering agents. Interactions between the human and agent were identified next. This affected the number and type of components needed in each agent and the interfaces used to interact with the agents. Section 4.4 showed that a combination of human-centered, incremental schedule revision and computer-centered, constraint satisfaction schedule generation meets the desirable characteristics of mixed-initiative scheduling solutions. Finally Section 4.5 discussed in detail the interface requirements identified by the

interaction models described in Section 4.3.3. The user's ability to visualize the scheduling process, modify system rules, and other desirable characteristics was also addressed. These design decisions result in the user-centered design of a decision support system for scheduling satellite operations.

V. Implementation

5.1 Overview

The goal of this research is to define a methodology for designing a decision support system for scheduling satellite operations. Chapter 3 presented this methodology and Chapter 4 identified the key design decisions necessary to implement a proof-of-concept system. The resulting proof-of-concept system presented here embodies the disciplined approach presented in the methodology. This ensures that the system is capable of representing the satellite operations scheduling domain and can work in concert with the human in all aspects of the mission scheduling process.

This chapter describes some key implementation details. First extensions to the Abstract Scheduling Domain Model are presented. The model is extended to support automatic demand generation and active resources. Second the structure of the mission scheduling agent and satellite engineering agent is presented. Architectural characteristics influenced by mixed-initiative considerations are highlighted. Third the interfaces used to interact with the agents are described. These interfaces are essential elements of the way the humans and agents work together to solve the scheduling problem. Finally the implementation of the problem solution techniques is presented.

5.2 Abstract Scheduling Domain Model Extensions

A property is added to each demand that states whether or not it can be scheduled automatically by the system. If it can, it has the further property that it is either a *time specific* demand or a *location specific* demand. Time specific demands in the satellite operations domain are characterized by frequency of occurrence. For example the frequency of a ranging demand is

daily. Other demands may occur monthly or in one case, every 40 days. Location specific activities are generated from data provided by the orbital analysis agent. The release-date for this demand type is calculated based on the time the orbital analysis agents predicts that the satellite will be in a specific location. For example at a certain *lunar attitude angle* a satellite is in *lunar eclipse* and requires an eclipse monitor mission. The system can automatically generate a demand that is satisfied by an eclipse monitor mission. This level of decision support is directly attributable to the extension of the abstract DEMAND object.

A further extension of the Abstract Scheduling Domain Model is the extension of RESOURCE to include properties that aid automatic demand generation. Resources are viewed as passive objects in the standard domain model. They are added or deleted from a domain and allocated to or deallocated from an activity. Instead the satellite operations scheduling problem represents satellites as active resources notifying the system when they require demands to be generated on their behalf. Each satellite has properties associated with the frequency that certain demands are required. When the frequency criteria are satisfied, the satellite requests the system to generate demands. Once the user specifies the criteria for the satellite, the user is no longer required to track when it is necessary to submit satellite specific demands.

5.3 Agent Implementation

The mission scheduling and satellite engineering functions in this system are represented by human/agents. Each human/agent is characterized by the components internal to the agent, the user interfaces designed to interact with the internal components, and the user that interacts with the agent via the user interfaces. This representation gives a clear understanding as to what user interactions are supported, how they are supported, and the components used to record the results of the user interactions. Figure 27 depicts the mission scheduler human/agent.

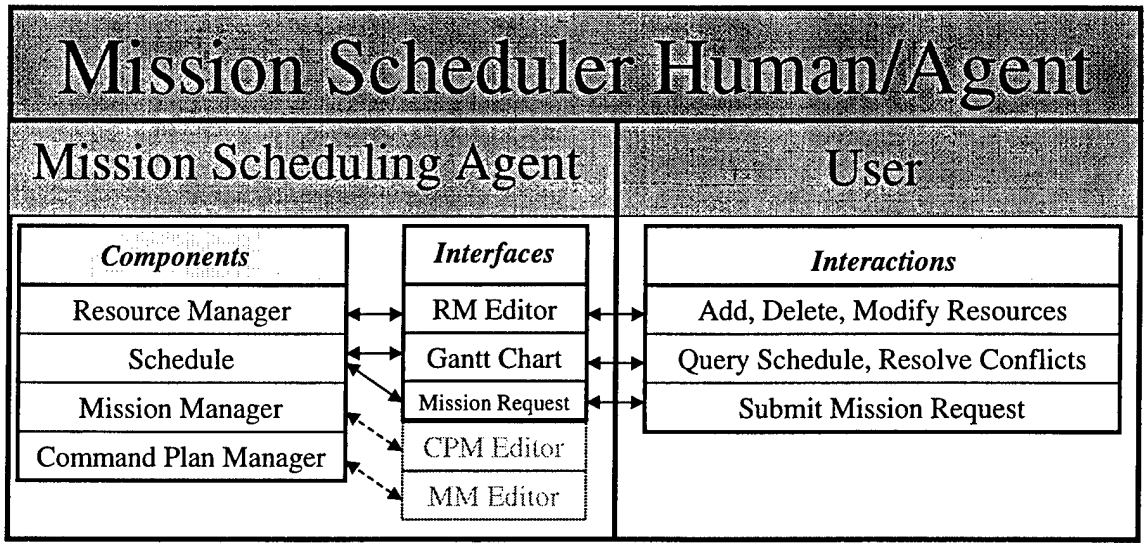


Figure 27: Mission Scheduler human/agent

Figure 27 shows that there are four components that comprise the mission scheduling agent. There are five interfaces that coordinate with the components. Three of these are local and support the interactions required by the human mission scheduler. The two interfaces that are grayed out refer to remote interfaces that support actions by the human satellite engineer. The dashed arrows reflect that a remote operation is responsible for updating these components (i.e., agent conversations are required). It was a design decision to include the mission manager and the command plan manager in the mission scheduler agent rather than the satellite engineer agent because they contain the system rules necessary for carrying out the scheduling process. If they were part of the satellite engineer, the mission scheduler agent would be required to retrieve them from the satellite engineer agent every time scheduling needed to be accomplished. Since the update frequency of the mission demand, mission type, and command plan libraries is less than the frequency of scheduling operations, this architecture reduces network traffic.

The resource manager of the mission scheduler agent is a critical component in the schedule process and deserves special attention. The resource manager's primary responsibilities are to track the availability of domain resources and assign available resources to command plans. It supports adding, deleting, and modifying resources. The resource manager is implemented as an aggregate manager. That is the resource manager has internal components that are themselves resource managers responsible for resource aggregates. For example the resource manager's site manager component is responsible for tracking the availability of the antenna sites in the domain. Each site manager is composed of sites that are further characterized by individual antennas and antenna strings. Figure 28 shows the components of the resource manager.

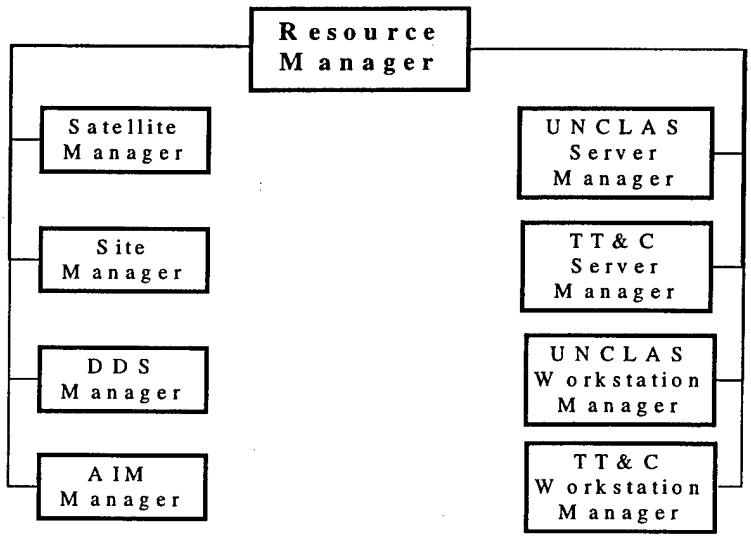


Figure 28: Resource Manager components

The satellite engineer human/agent has only one internal component. It maintains its own copy of the schedule. This enables the human satellite engineer to avoid submitting mission requests that would result in an inconsistent schedule (i.e., conflict avoidance) and to query scheduled activities. Other interactions in which the human satellite engineer can participate are supported by special local interfaces that coordinate with remote components in the mission scheduler human/agent. These remote components are subdued in the diagram. As in the mission scheduler human/agent diagram, the dashed arrows reflect that the operations of the satellite engineer agent interfaces are carried out remotely

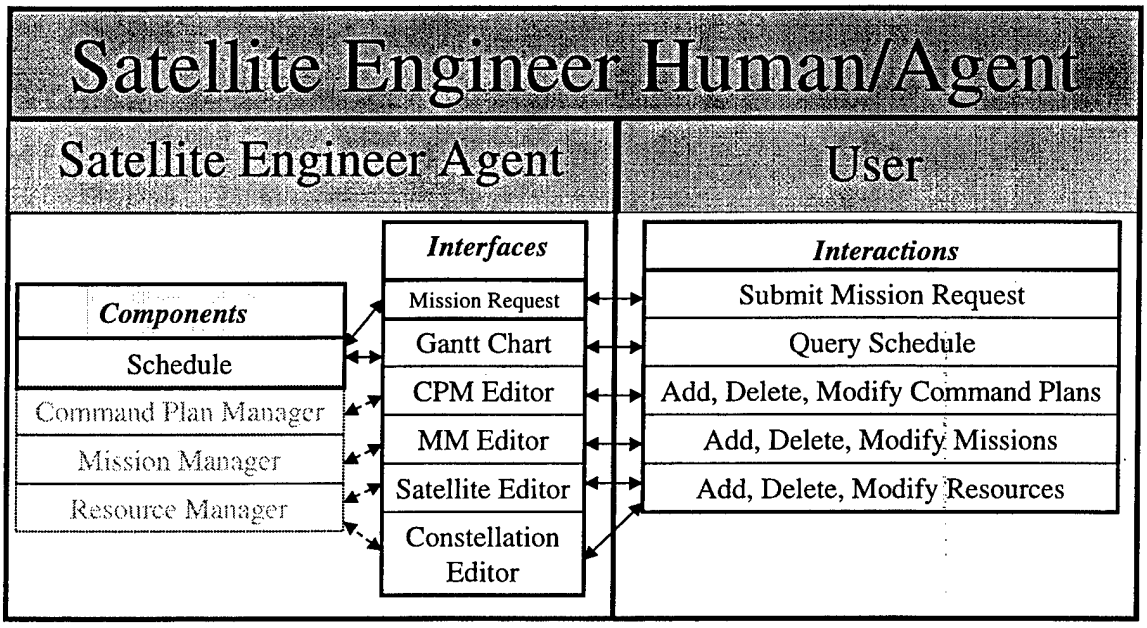


Figure 29: Satellite Engineer human/agent

5.4 User Interfaces

When the system is first started, the user is presented with a Gantt Chart representing the current activities scheduled in the system (if any are currently scheduled). The chart has a tab feature that supports multiple representations of the schedule dependent on a user's needs. The

mission scheduler is interested in viewing activities as either scheduled or unscheduled. Therefore the mission scheduler human/agent Gantt Chart has a *scheduled tab* and an *unscheduled tab*. Conversely the satellite engineer is only concerned with viewing the scheduled activities for each satellite. Therefore each tab in the Gantt Chart represents a satellite.

Parent diagrams further divide the Gantt Chart. Parent diagrams consist of *child diagrams* that represent individual activities. For the purposes of this problem, each parent diagram represents a site and all activities scheduled at that site are child diagrams in the Gantt Chart. This allows the satellite engineer, for example, to look for all activities for Satellite 1 at site AFSCN by selecting the AFSCN parent diagram on the Satellite 1 tab. Figure 30 depicts the satellite engineer Gantt Chart.

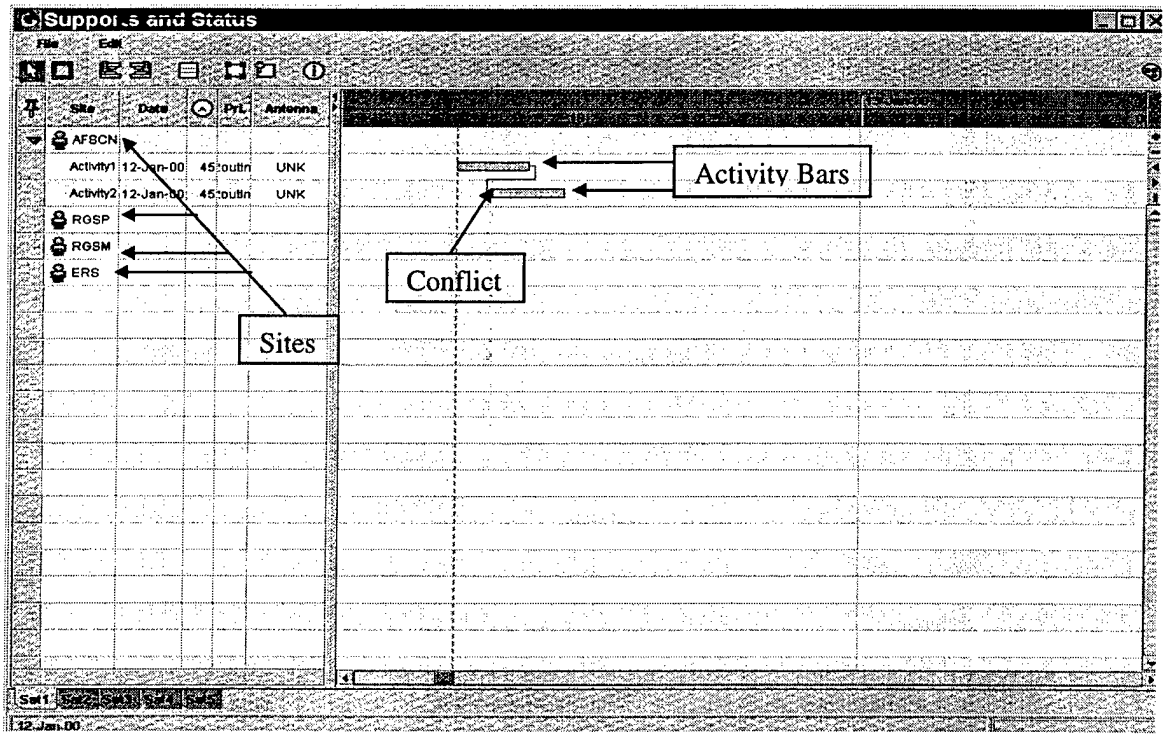


Figure 30: Satellite Engineer Gantt Chart Interface

Clicking on the arrow beside AFSCN displays any activities scheduled on the AFSCN. In Figure 30 Activity 1 and Activity 2 are scheduled on the AFSCN. The chart displays the date and duration for each activity as well as its priority and the antenna on which it is scheduled. The red connector line between the activities indicates there is a resource conflict between the two. The blue-dotted line represents the current system time and shows that Activity 1 is about to start execution. Activities are color-coded based on their priority (here both activities are routine).

One of the interactions requires the human mission scheduler and the human satellite engineer to query scheduled activities. By double clicking on the activity bar an activity interface is displayed that shows the details of the given activity. In addition if there are any conflicting activities with the queried activity they are listed along with the reasons for the conflicts. The satellite engineer does not have the authority to update scheduled activities so the activity query interface displayed for this user is static. In contrast the mission scheduler is required to resolve conflicts so its activity query interface is editable.

Each human/agent has the ability to submit a mission request to the system. The mission request interface is opened by selecting the Gantt Chart's edit menu option *Submit Mission Request*. Figure 31 shows a mission request. Note that the mission request interface is significantly different than the original activity request shown in Figure 15, page 56. No longer does the user specify resources or an activity type. The user selects a mission demand instead. The mission demand (defined in the mission demand library) defines the mission types that satisfy it. It is the mission types that determine the resources required for their constituent command plans.

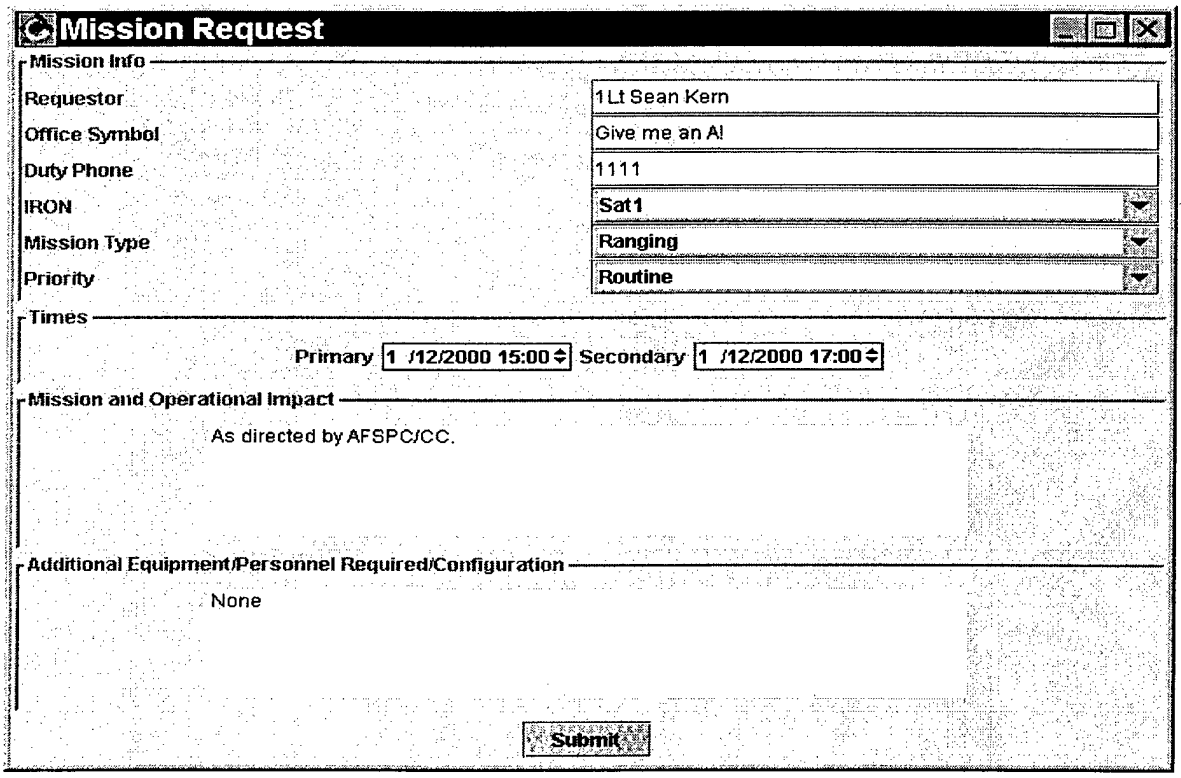


Figure 31: Mission Request Interface

As part of the resource management interactions in the system the human satellite engineer has the responsibility to maintain the state of the satellite constellation. This is accomplished in two ways. First there is a constellation editor interface that allows the human satellite engineer to add and delete satellites from the constellation. This interface is activated from the edit menu on the Gantt Chart. The satellite engineer must specify all of the operational characteristics of a new satellite before it can be added to the constellation. Adding and deleting satellites results in the addition and deletion of Gantt Chart tabs. Therefore the display is always a "real-time" representation of the current scheduling domain. The second way the user maintains the constellation state is by editing the operational parameters of an existing satellite. This too can be accomplished from the edit menu. These parameters are used by the scheduling system to automatically determine the necessary missions demands to generate and therefore represent a

subset of the system rules that govern scheduling. Figures 32 and 33 depict the constellation editor and the satellite editor respectively.

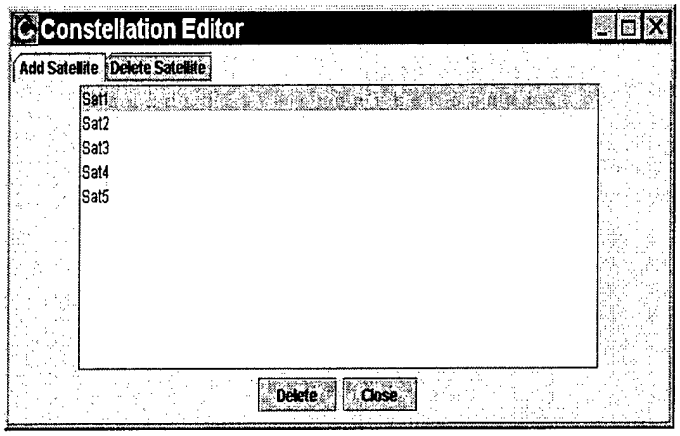


Figure 32: Constellation Editor Interface

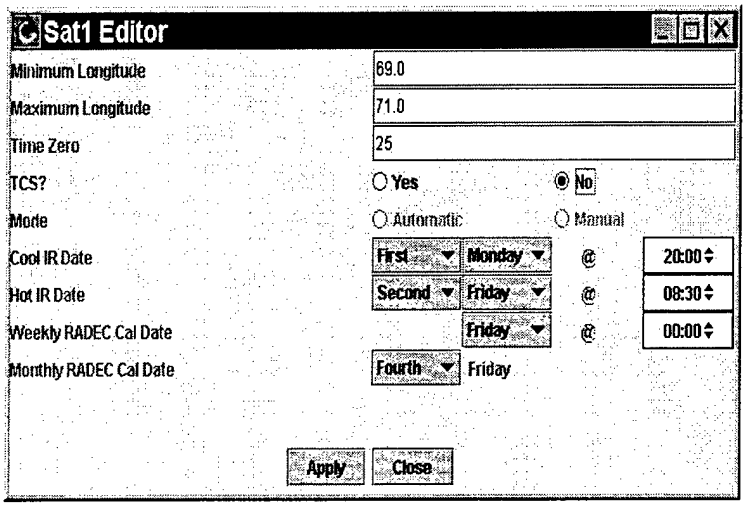


Figure 33: Satellite Editor

Figure 32 shows an add tab and a delete tab. The add tab is much like the satellite editor of Figure 33. Although the interfaces are similar, these interactions are separated in order to represent precisely the human/agent interactions.

The mission scheduler human/agent is responsible for managing the remaining resources in the scheduling domain. These include all of the sub-resource managers described in Figure 28 except for the satellite manager. This interaction is necessary in the event that new resources are added to the system or deleted from the system. The system needs to be aware of the changing circumstances since the number and type of resources affect the scheduling process. This interface is not implemented in the system due to time constraints and also resources rarely change.

The final interactions in the system are the addition, deletion, and modification of the command plans and mission types maintained by the command plan manager and mission type manager respectively. The command plan editor interface lists all the details of the command plan currently selected in the command plan window. Figure 34 shows the command plan editor interface.

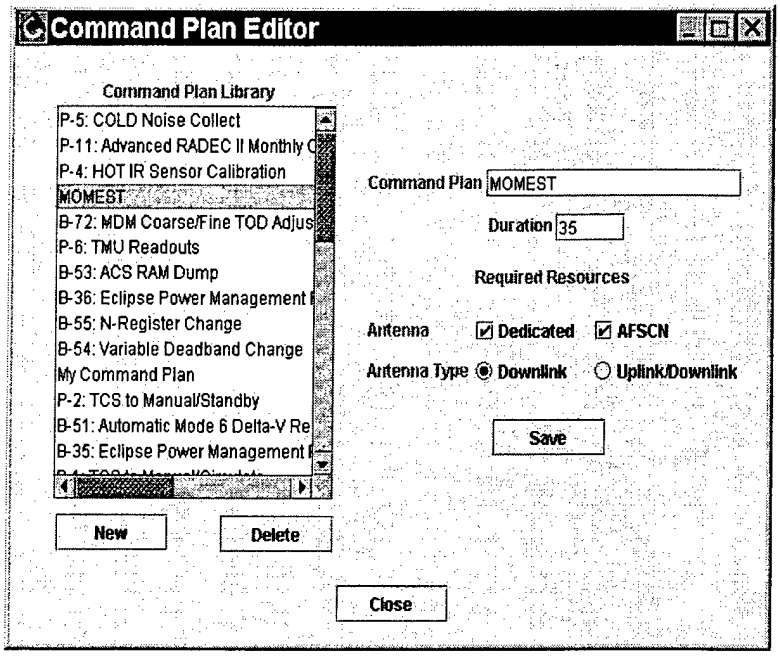


Figure 34: Command Plan Editor

The user can add, delete, or modify command plans using this interface. Note that each command plan has a name and a duration as well as required resources. Only three types of resource requirement scenarios are identified; these are represented by the antenna checkboxes and the antenna type radio buttons. First if the user selects the AFSCN checkbox only, then the user is specifying that the command plan may only be scheduled on an AFSCN antenna. All AFSCN antennas are uplink/downlink types and therefore the Uplink/Downlink radio button is selected automatically. The command plan is assigned the AFSCN resource template. Second if the user selects any other combination of antenna types the default preference is to use a dedicated antenna and the Uplink/Downlink radio button is selected. This is the Dedicated Uplink Downlink resource template. Finally if the Downlink radio button is selected then the Dedicated Downlink Only resource template is assigned. If neither antenna checkbox is selected (it is possible) the system assigns defaults to either Dedicated resource template based on the antenna type selected.

The mission type editor interface lists all the details of the mission type currently selected as well as displays all the command plans available in the system. This allows the user to add command plans from the command plan library. Figure 35 shows the mission type editor interface.

In the figure the Ranging mission type is selected. It consists of six ranging activities each separated by four hours. In a scheduling scenario if a mission demand was submitted that required this mission type, the first ranging activity would be scheduled at the mission demand's release-date and the following activities would be scheduled in subsequent four hour increments. All of the temporal relations are *starts after* relations because that is the nature of mission types in this domain. If the user wanted to model a *starts-after-end* relation the relation would be the tuple

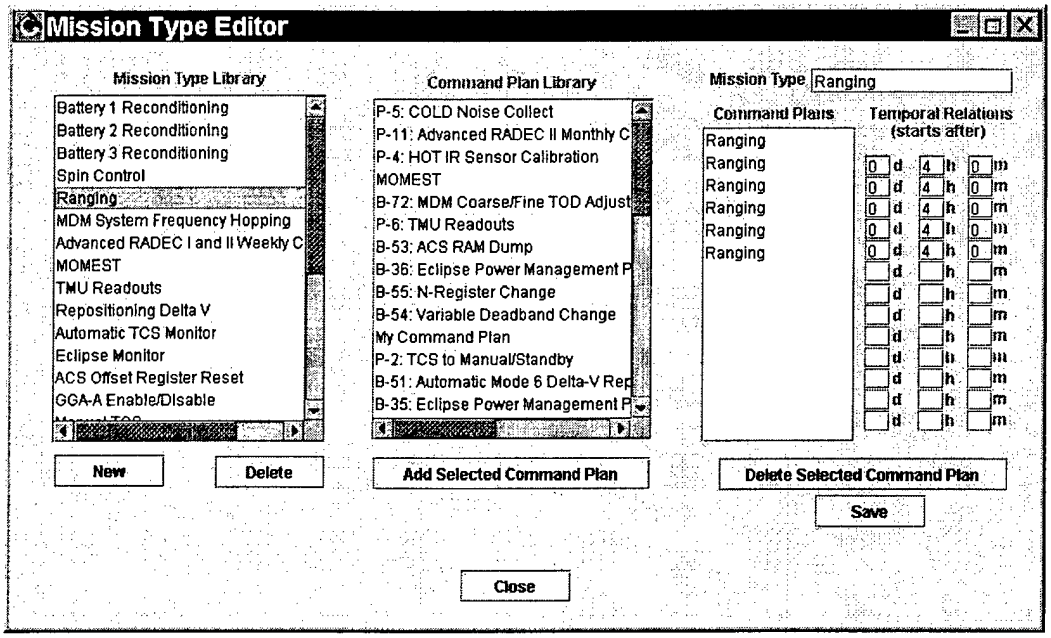


Figure 35: Mission Type Editor

(0,0,0) representing that the activity would take place 0 days, 0 hours, and 0 minutes after the end of the previous activity. There are three ways the user can handle a *starts-before* relation. First a command plan, call it cp1, has to be deleted, a new command plan (cp2) added, and then cp1 is added after cp2. Second the interface supports the insertion of a command plan at any point in the mission type specification (i.e., third command plan to execute). This insertion moves any command plans below that point down and zeroes the temporal intervals. The user must specify new temporal relations if desired. Finally the user could use negative values in the temporal relations, but this is highly discouraged since it makes it difficult to understand. Other relations such as *starts-at-start, starts-after-begin,* and *contained-by* are not allowed since only one activity can occur on a satellite at time. No other relations are required to represent mission typs.

5.5 Problem Solution Techniques

There are two elements to the problem solution technique chosen to solve the satellite operations scheduling problem. The first is the human-centered, expert iterative refinement

technique. The second is the computer-centered, constraint satisfaction technique. Both are integrated in a single framework that allows the user to control the schedule process. This control is exercised via several scheduling modes. The first mode is completely manual. The user is responsible for generating all mission demands. The second mode enables automatic mission demand generation. In both of these cases the user may conduct expert iterative refinement or select automatic resolution.

The basic scheduling algorithm is quite simple. It follows the same logic discussed throughout this thesis. First a mission demand is generated. The source of the mission demand is not an issue (i.e., user submitted or system generated). The mission demand library is queried to determine the mission type(s) that satisfies the mission demand. The mission type library is then queried to determine the command plans to schedule. Each command plan, based on temporal ordering, is then passed to the resource manager along with its required resources. The resource manager assigns available resources or in the case of conflicts, marks each conflicted resource and any competing activities. Finally the fully instantiated command plans are displayed by the interface agents.

The human-centered approach relies on the user's expertise and the information available from the system to resolve conflicts. The system has two responsibilities to support this technique. First the system notifies the user that conflicting activities exist. This helps the user identify the location of inconsistencies in the schedule. Second it changes the schedule to reflect the user's action. This allows the user to view the result of his action and determine whether or not other inconsistencies are introduced. This scenario can go on as long as the user wishes. The user always has the option to let the system automatically compute a new schedule by selecting the "automatic resolve" command in the schedule menu.

The computer-centered, constraint satisfaction approach models the satellite operations scheduling problem as a resource allocation problem and solves for each resource an activity requires. First the activities to be scheduled are pre-processed to identify all the activities that overlap. For all activities that overlap a constraint is posted that does not allow any of the resources of these activities to be the same. Once these constraints are posted, a constraint satisfaction representation is built for each resource and then either the arc-consistency or simple backtracking algorithm is used to solve the problem. An example specification for the antenna constraint satisfaction problem is presented next.

For every activity A_i a variable AV_i (i=1,...,n) is created and the domain of the variable is the set of possible antennas the activity A_i can use. There is an inequality constraint between two variables if their domains intersect in at least one value.

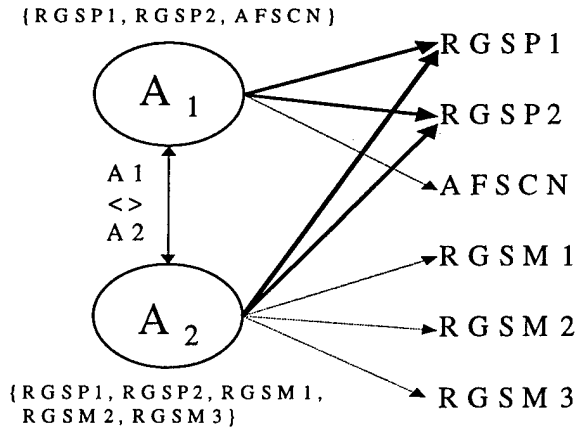


Figure 36: Antenna Constraint Satisfaction Problem

The bold arrows represent the intersection of the activity domains. This intersection results in the addition of the inequality constraint between the activities only if the activities overlap. If there is no overlap then the domain intersection between the activities does not matter.

Each resource in the domain is modeled in a similar way and each constraint satisfaction problem is executed. The benefit of this approach is that if a solution does not exist, it is clear what resource is in conflict. This is an essential point since returning no solution at all is unacceptable. Instead a partial solution is returned to the user and the user must make a decision to move an activity or in the worst case delete it from the schedule entirely.

5.6 Summary

This chapter discussed implementation details regarding the proof-of-concept system. It described the agent implementations and how the required user interactions affect the design of the agents. Discussion then centered on the implementation of the problem solution techniques. The human-centered, expert iterative refinement method is presented as well as the computer-centered constraint satisfaction problem. The implementation details presented here form the key concepts critical in the process of translating the design from the methodology to actual implementation.

VI. Results and Conclusion

6.1 Overview

The goal of this research is to define a methodology for designing a decision support system for scheduling satellite operations. This research presents a methodology that systematically decomposes the design problem into an ordered sequence of design decisions. The result of each design decision acts as input to the following design decision. The design decisions surrounding the scheduling problem representation step identify, classify, and organize scheduling objects by process participant. The system analysis and design step uses this process level organization to select an appropriate architecture and system design methodology. Based in part on the chosen architecture, the designer can choose the appropriate class of problem solution techniques. Finally the culmination of the previous steps results in a user interface requirements definition. The user interface requirements definition specifies the manner in which the user can interact with decision support system.

A common theme became evident as the research on this methodology progressed; a successful methodology is a human-centered methodology. It is inconceivable to design a decision support system without considering the end user, yet history shows this to be the norm. Representing the problem in a way that supports user intuition by using common terms and ensuring understanding is the critical first step. Choosing a mixed-initiative, agent-oriented design approach influences the underlying system architecture ensuring the mechanisms to support user interaction are available. User interfaces are mapped directly to these architectural components, guaranteeing the emergence of a tightly coupled human/agent. This tight coupling is at the heart of a well-designed decision support process and is central to the methodology this research presents.

6.2 Results

The overarching result of this research is that a human-centered design approach for decision support systems is essential. In addition the scope of the methodology presents some fascinating results in several specific areas including problem representation, problem visualization, and solution techniques.

6.2.1 Problem Representation

A key result of the scheduling problem representation step is the satellite operations domain has no concept of products. Instead satellite operations relies on manually scheduling individual activities. This forces the user to recognize the temporal relations that exist between activities and ensure that these relationships are enforced on the mission schedule. Modeling temporally related activities as a mission type (i.e., product) allows the system to aid the user by defining the temporal relations and enforcing them automatically.

A second key result of the scheduling problem representation step is the extension of the abstract DEMAND and RESOURCE objects in the Abstract Scheduling Domain Model to contain properties that aid in the automatic generation and scheduling of demands. The current literature does not address the ability of a scheduling system to automatically generate and subsequently schedule demands. The scheduling systems reviewed in the current literature rely on the user to specify the demand. A scheduling system uses the release-date property of the DEMAND to schedule its associated activities. This implies that all demands are *time specific*. In the satellite operations domain, some demands are *location specific*. The location of a satellite in relation to the sun and/or moon determines whether certain demands are required. The times of these location events are used as release-dates for automatically generated mission demands.

6.2.2 Problem Visualization

This research shows that conflict avoidance aids in maintaining a consistent mission schedule. For demands that have very flexible release-dates, the user looks for holes in the mission schedule that support the user's demand. Alternatively the system can return open time intervals based on certain user provided criteria (this feature is not implemented in the proof-of-concept system). These actions give the user access to all of the current scheduling decisions and aids the user in selecting a demand release-date that is consistent with the current mission schedule. Of course if there is no alternative release-date, then the user has no choice but to submit a demand that is inconsistent with the current schedule.

In the event that a conflict does occur, schedule visualization gives the human scheduler the ability to quickly identify conflicting activities and determine the reason(s) for conflict. If the user makes a change to an activity, the visual representation changes to reflect the result of the user's action. Incremental conflict resolution as a result of expert refinement may be all that is required to solve the conflict. In the event that it is not, the problem solution techniques can support further conflict resolution.

Aside from the visual representation of the schedule, there is the visual representation of the system rules used to schedule activities. The systems rules are the properties and capabilities of the mission demands, mission types, and command plans in the system. The user can add, modify, or delete these rules so that the operation of the system reflects the real world. The mission demand, mission type, and command plan editors support this visualization.

6.2.3 Problem Solution Techniques

This research presents a framework that integrates automated and user-directed problem solution techniques based on the user's preference. The goal of this solution framework is to

allow the user to control as much of the problem solution process as possible. This form of *solution control management* is recognized as an ideal trait in mixed-initiative systems [BM94]. The user has control over the incremental revision process by manually making changes to the conflicting activities' parameters. If a conflict still exists even after the user's attempt to incrementally change the schedule, the user can choose to automatically resolve the conflict by asking the system to compute a new schedule. The system accomplishes this task by building a constraint satisfaction problem from the activities to schedule and then executing an arc-consistency or simple backtracking algorithm. Although the satellite operations squadron that served as the example in this research has an undersubscribed problem and does not require the complexity of the constraint satisfaction search, it is advantageous to represent the problem in this manner since many squadrons do have many multiple competing activities and limited resources. This constraint satisfaction approach provides a general, scalable problem solving strategy for those satellite operations domains that are oversubscribed.

The ability to generate many valid solutions using automated techniques allows solution quality to play a role in satellite operations. Historically the satellite operations scheduling community was satisfied by the first conflict-free schedule it could devise. There is no concept of solution quality since generating alternatives is too costly. Now the user can control the scheduling process by choosing preferences that affect the resulting outcome. The user may prefer using only antennas with uplink and downlink capability first (even though an activity may only require downlink) since commanding can be done in case of an anomaly. The user may also prefer to use dedicated sites rather than AFSCN sites. The schedule solutions are processed by an evaluation function and the highest rated solution, based on the user's preference, is selected.

6.3 Future Work

Due to the methodology's broad nature there are many areas of interest for future work. The most interesting area of work is case-based reasoning. This could aid the decision support system dramatically. If the agent was able to recognize temporal relations between otherwise independent activities it could offer the user the ability to create new mission types. Also it could aid the problem resolution process by recording the user's actions and preferences during incremental revision scenarios. The agent could then support future incremental revision scenarios or apply the user's preferences directly thus resolving the conflict without user intervention. This agent ability is especially useful in the context of assisting novice users. Case-based reasoning requires the agent to understand the context in which the user is working. Research in context-based interaction could benefit this system. This could be in the form of formal discourse languages for use between the agent and the user.

Only the arc-consistency and simple backtracking algorithms are implemented in the current system. It would be possible to implement additional constraint satisfaction algorithms to generate schedule solutions. The solution quality functions could be applied to find the most successful algorithm(s).

The algorithms for recognizing location specific trends in the orbital analysis data were not implemented because not enough data was provided to cover the possible mission types. These need to be implemented to get the full benefit of automatic demand generation.

A domain language for representing resources, activities, mission types, and mission demands is needed to model different satellite operations squadrons' requirements. This domain specification could then be processed at system initialization time allowing the general class library to more easily represent each squadron's scheduling domain.

6.4 Final Comments

Many methodologies exist at different levels of abstraction. Some design methods are relevant to individual agent design, others for system level design. The goal of this research was to present a systematic design approach that encompassed the necessary design decisions at the appropriate levels of abstraction in order to design a decision support system for satellite operations. By following the approach described herein, a designer can design a system that takes advantage of and enhances the capabilities of the user. The end result is a system that produces schedules better than a human or computer can do alone.

BIBLIOGRAPHY

[BCK98] L. Bass, P. Clements, and R. Katzman. Software Architecture in Practice. Addison Wesley. 1998.

[BM94] M. Burstein and D. McDermott. "Mixed-Initiative Military Planning: Directions for Future Research and Development."

[CBC97] A. Cesta, P. Bazzica and G. Casonato. "An Obect-Oriented SchedulingArchitecture for Managing the Data Relay Satellite Requests."

[COS99] A. Cesta, A. Oddi, and A. Susi. "O-OSCAR: A Flexible Object-Oriented Architecture for Schedule Management in Space Applications" *Proc. of the 5th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space* (I-SAIRAS-99).

[CP94] C. Le Pape. "Scheduling as Intelligent Control of Decision-Making." In M. Zweben and M.S. Fox, eds., *Intelligent Scheduling*, chapter 3, 1994.

[CV97] M.T. Cox and M.M. Veloso. "Controlling for unexpected goals when planning in a mixed-initiative setting." In E. Costa & A. Cardoso (Eds.), *Progress in Artificial Intelligence: Eighth Portuguese Conference on Artificial Intelligence* (pp. 309-318). Berlin: Springer.

[DC87] D. Chapman. "Planning for Conjunctive Goals." *Artificial Intelligence*, 32:333-377

[DEJAVU99] "A Reusable Framework for the Construction of Intelligent Interactive Schedulers." http://www.dbai.tuwien.ac.at/proj/DejaVu/document/docu.htm

[EK98] E.A. Kendall. "Agent Roles and Role Models." *Intelligent Agents for Information and Process Management* (AIP'98).

[ET93] E. Tseng. Foundations of Constraint Satisfaction. University Press. 1993.

[FA94] G. Ferguson and J.F. Allen. "Arguing About Plans: Plan Representation for Mixed-Initiative Planning." *AIPS-94*. pp 43-48, 1994.

[FAM96] G. Ferguson, J.F. Allen, and B. Miller. "TRAINS-95: Towards a Mixed-Initiative Planning Assistant. In *Proceedings of the Third International Conference on AI Planning Systems*. Edinburgh, Scotland, May 29-31, 1996.

[HAS72] H.A. Simon. "On Reasoning about Actions." *Representation and Meaning: Experiments with Information Processing Systems*. pp 414-430, 1972.

[HD99a] T.C. Hartrum and S.A. DeLoach. "Design Issues for Mixed-Initiative Agent Systems." 1999. In *Proceedings of the AAAI-99 Workshop on Mixed-Initiative Intelligence, Orlando FL, July 1999.*

[JCB99] J.C. Beck. Texture Measurements as a Basis for Heuristic Commitment Techniques in Constraint Directed Scheduling. PhD Thesis, Enterprise Integration Laboratory, Department of Industrial Engineering, University of Toronto, 4 Taddle Creek Road, Toronto, Ontario, M 5S 3 G9, Canada.

[LS94] O. Lasilla, S.F. Smith. "Constructing Flexible Scheduling Systems for Decision Support." *Proceedings 1994 Finnish AI Symposium*, Turku (Finland), August 1994.

[MSF94] M. S. Fox. "ISIS: A Retrospective." In M. Zweben and M.S. Fox, eds., *Intelligent Scheduling*, chapter 1, 1994

[NWA96] H.S. Nwana. "Software Agents: An Overview." *Knowledge Engineering Review*, Vol. 11, No. 3, pp 1-40, Sept 1996. Cambridge University Press.

[OC94] T. Oates and P. R. Cohen. "Toward a plan steering agent: Experiments with schedule maintenance." In *Proceedings of the Second International Conference on Planning Systems* (AIPS-94) (pp. 134-139). Menlo Park, CA: AAAI Press

[PHOT94] M.J. Prietula, et. al."MACMERL: Mixed-Initiative Scheduling with Coincident Problem Spaces". In M. Zweben and M.S. Fox, eds., *Intelligent Scheduling*, chapter 3, 1994.

[POP92] P.O. Perry. "User-Centered Scheduling Support in the Military Airspace Management System Prototype." 1992

[RN95] S. Russell, P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall. 1995

[RPD87] R. Prieto-Diaz. "Domain Analysis for Reusability" *Proceedings of COMPSAC '87*, pp 23-29, 1987. IEEE Press

[SB97] S.F. Smith, M. Becker. "An Ontology for Constructing Scheduling Systems." *Working Notes of 1997 AAAI Symposium on Ontological Engineering*, Stanford, CA, March, 1997 (AAAI Press).

[SC96] S. Chien. et al. "Why Real World Planning is Difficult: A Tale of Two Applications." *New Directions in AI.* pp 287-298. 1997.

[SD99] S. DeLoach. "Mutliagent Systems Engineering: A Methodology And Language for Designing Agent Systems." *Agent-Oriented Information Systems* (AOIS) '99.

[SG96] M. Shaw, D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall. 1996.

[SHOH97] Y. Shoham. "An Overview of Agent-oriented Programming" J.M. Bradshaw (ed), *Software Agents*, AAAI Press, 1997.

[SLB96] S.F. Smith, O. Lassila, M. Becker. "Configurable, Mixed-Initiative Systems for Planning and Scheduling, in Advanced Planning Technology." A. Tate, ed. AAAI Press, Menlo Park, CA, May, 1996.

[SV83] S. Vere. "Planing in Time: Windows and Durations for Activities and Goals." In Allen, J.,et al, eds., *Readings in Planning*, pp 297-318, 1990

[WJ95] M. Woolridge and N. Jennings. "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review*, Vol. 10, No. 2, June 1995, Cambridge University Press.

[WJK99] M. Woodridge. N.R. Jennings, and D. Kinny. "A Methodology for Agent-Oriented Analysis and Design." *Agents '99*. Seattle, WA.

VITA

First Lieutenant Sean C. G. Kern was born on 21 October 1970 in Indianapolis, Indiana. He graduated from Dover Area High School in Dover, Pennsylvania in June 1988. He entered undergraduate studies at the Pennsylvania State University before enlisting in the Air Force in 1990. After assignments in Woomera, Australia and Patrick AFB FL, he graduated with a Bachelor of Science degree in Computer Science in December 1995 from Rollins College. He was commissioned through Officer Training School, Maxwell AFB AL, in June 1996.

His first assignment was with the 1st Space Operations Squadron at Schriever AFB CO as a Communications-Computer Systems Officer in July 1996. In August 1998, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Force Operational Test and Evaluation Center, Kirtland AFB NM. He is married to the former Angela Payne of Sydney, Australia and he has two daughters: Amber and Caitlin.

INDEX

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2000	Master's Thesis

4. TITLE AND SUBTITLE

Sean C. G. Kern, 1Lt, USAF

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 P Street, Building 640
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/00M-10

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

2 SWS/DOUE
Attn: Captain Nicholas Martin
18300 E. Crested Butte Avenue (STOP 69)
Buckley ANGB, CO 80011-9518 DSN 877-5439; COMM 303-677-5439

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Dr. Henry B. Potoczny, ENG, DSN: 7856565, ext 4280

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The choice to include the human in the decision process affects four key areas of system design: problem representation, system analysis and design, solution technique selection, and interface requirements specification. I introduce a design methodology that captures the necessary choices associated with each of these areas. In particular I show how this methodology is applied to the design of an actual decision support system for satellite operations scheduling.

Supporting the user's ability to monitor the actions of the system and to guide the decision process of the system are two key considerations in the successful design of a decision support system. Both of these points rely on the correct specification of human-computer interaction points. Traditional, computer-centered system design approaches do not do this well, if at all, and are insufficient for the design of decision support systems. These approaches typically leave the definition of human-computer interaction points till after the component and system level designs are complete. This is too late however since the component and system level design decisions can impose inflexible constraints on the choice of the human-computer interaction points. This often leads to the design of human-computer interaction points that are only "good enough." These approaches result in ill-conceived problem representations and poor user-system interaction points because the system lacks the underlying architecture to support these constructs efficiently. Decision support systems require a new, human-centered design approach rather than the traditional computer-centered approaches.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Agents, Object-oriented, Decision Support, Mixed-Initiative, Scheduling, Software Design, Satellite Operations	146
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL