Representing Spacecraft Mission Planning Knowledge in ASPEN

Ben Smith **Rob Sherwood** **Anita Govindjee** **David Yan**

Gregg Rabideau **Steve Chien** **Alex Fukunaga**

Jet Propulsion Laboratory
Pasadena, CA 91109-8099
firstname.lastname@jpl.nasa.gov

Abstract

Automated planning technologies show great promise of reducing operations costs by automating the spacecraft mission planning process. However, one of the bottlenecks is acquiring spacecraft operations knowledge from operations personnel and expressing it in a plan model. One of the primary design goals of the ASPEN planning language is to eliminate the knowledge acquisition bottleneck by making it easy for operations personnel to build plan models themselves.

The ASPEN language is designed to be used by domain experts that have no knowledge of automated planning technology. Spacecraft operations knowledge can be expressed in ways that are natural to operations personnel. The language is intended to be intuitive and require almost no knowledge of how the planning algorithm works. ASPEN itself provides hooks for interfacing with existing planning support tools, and has a mixed-initiative planning mode and GUI that allow operations personnel to easily edit automatically generated plans if desired. A non-AI expert with an operations background was able to construct ASPEN models that are scheduled for use on two missions, EO-1 and UFO-1.

Introduction

Automated planning/scheduling technologies show great promise of reducing operations costs by automating the spacecraft mission planning process. The Artificial Intelligence Group at the Jet Propulsion Laboratory has been developing a system called ASPEN (Automated Scheduling and Planning Environment). ASPEN (Fukunaga *et al.* 1997) is a modular, reconfigurable application framework based on AI techniques (e.g., (Allen, Hendler, & Tate 1990; Fox & Zweben 1994)), that can support a variety of planning and scheduling applications (similar to (Smith, Lassila, & Becker 1996)). The primary application area for ASPEN is the spacecraft operations domain.

Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals. The ASPEN planning model encodes spacecraft operability constraints, flight rules, and operations procedures to allow for automated generation of spacecraft sequences. By automating the command sequence generation process and by encapsulating the operations-specific knowledge, ASPEN will enable spacecraft to be commanded by much smaller operations teams and thereby reduce operations costs.

The ASPEN planning model encodes knowledge that is in the domain of mission operations personnel. Ideally, it is these experts who should build the planning models. This removes a major knowledge acquisition step, which is often costly and a source of errors. The operational constraints, flight rules, and other planning knowledge often change up to launch, and to a lesser extent even through operations. The mission planners and operations personnel are in the best position to know when and how this knowledge is changing, and keep the ASPEN models consistent with that knowledge.

The ASPEN modeling language is designed to allow operations personnel, who generally do not have an AI planning background, to develop ASPEN models quickly and easily. The ASPEN modeling language was designed to be intuitive, require almost no knowledge of AI planning technology, and to naturally express planning knowledge for common aspects of spacecraft operations.

The remainder of this paper describes the ASPEN modeling language and how it facilitates expression of planning knowledge for spacecraft operations. We first describe the modeling language and demonstrate how it can express planning knowledge for common aspects of spacecraft operations. We then discuss how ASPEN supports modular design of models and how it allows the modeler to be ignorant of the underlying planning

engine. Finally, we describe an ASPEN model that is scheduled to generate activity plans and command sequences for the EO-1 mission. This model was developed by a JPL engineer with a background in spacecraft operations (TOPEX/Poseidon and Mars Observer) but not in AI planning.

Overview of the ASPEN Language

In order to build an ASPEN model, a domain expert needs to know a few things about the modeling language, but can remain almost completely ignorant of the ASPEN planning engine. The engine can be treated as a black box that produces a plan that is consistent with the model and achieves the goals from the initial state of the spacecraft, both of which are specified in the initial state file.

The modeler needs to know the planning language, the elements that comprise a plan, and what it means for a plan to be consistent with the model. The main ASPEN plan elements are *activities*, *resources*, *states*, *temporal constraints* and *reservations*. All of these elements are specified in a *plan model* expressed in the ASPEN planning language. The model stores all of the domain-specific information. The following subsections describe each of these elements, how they are specified in the model, and what it means for a plan to be consistent with a given specification.

Activities

Activities are the plan elements that change the state of the world when executed. An activity type has a type name and zero or more parameters. An activity type can have several instantiations in the plan and initial state file (similar to types and objects of those types in C or C++). Parameters are the "local variables" of an activity.

All activities have a start and end timepoint and a duration. Activities in a plan can overlap, and there can be times when no activities are scheduled. A temporal constraint enforces a temporal relation between activity timepoints. Temporal constraints can only exist between activities; they cannot exist between any other plan elements.

Timepoints are expressed as intervals in the model, but are narrowed to a fixed time in the plan. The engine can reason about the interval when deciding how to schedule the activity, but this flexibility does not need to be maintained in the plan itself. That is, ASPEN has a *committed* plan representation (e.g., (Smith 1994; Zweben *et al.* 1994)). This is consistent with how most operations personnel think about planning. They know what the intervals are for each activity, but narrow those to a specific time on the mission plan.

```
Activity catbed_heater_on {
   reservation =
     cat_htr_sv change_to "on";
};

Activity fire_engine {
   constraint =
     starts_after end_of catbed_htr_on
        by [1800, infinity];
   reservation = engine_sv change_to "firing";
};
```

Figure 1: A Science Observation Activity

This committed approach is also consistent with the spacecraft operations environment. Traditional spacecraft execution engines cannot deal with intervals; they must be given fixed times at which to execute each activity. Planning support systems, with which ASPEN may need to communicate, usually require fixed times rather than intervals. (Communication with external planning support systems will be discussed later).

Activities are the source of all constraints in the model. There are four kinds of constraints that activities can impose on other plan elements: temporal constraints, functional dependencies, resource reservations, and state reservations.

Temporal Constraints

A temporal constraint is a temporal relation between a source activity type and a target activity type. The relation must be satisfied by every pair of activity instances of those types in the plan. The ASPEN language defines six temporal relations: starts_before, starts_after, ends_before, ends_after, contains, and contained by.

The first four relations are between a timepoint (start or end) of the source activity and a timepoint of the target activity. The last two relations, "contains" and "contained_by" are defined on the start and end timepoints of the activities. The relation "A contains B" is equivalent to a "A starts_before start of B" and an "A ends_after end of B" relation. The relation "A contained_by B" relation is defined symmetrically.

A temporal relation can be modified by an optional interval. For example, the temporal relation in Figure 1 has an interval of [1800,infinity]. This means the engine burn activity must occur at least 1800 seconds (half an hour) after the catbed_heater_on activity. That is, the engine should not be fired unless the catbed has been heating for at least half an hour. This is a common constraint on most spacecraft. If a temporal interval is specified as [0,0], the timepoints must coincide exactly.

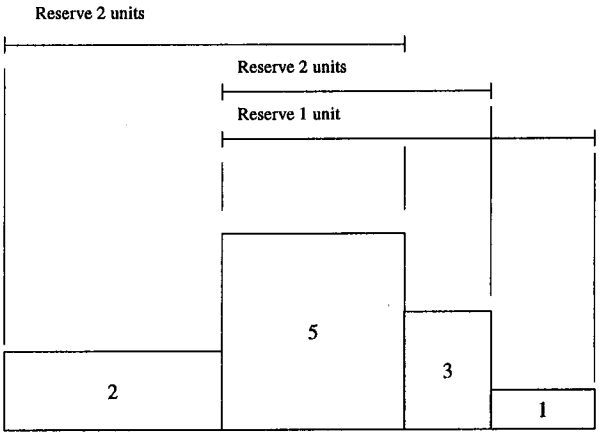

Figure 2: Resource Timeline with Overlapping Reservations

Resources and Resource Reservations

ASPEN has two kinds of resources: aggregate and atomic. An aggregate resource has some finite capacity, measured in number of units. Its units can be allocated among multiple activities at any time. Aggregate resources can be either depletable or non-depletable. Depletable resources are consumed by the activities that use them, whereas non-depletable resources are only removed from availability during the activity and become available again when the activity terminates. An atomic resource can be used by only a single user (activity) at a time, and is not consumed by the user.

Activities state their resource requirements via resource *reservations*. A reservation for an aggregate resource specifies the resource and the number of units. An atomic resource reservation only specifies the resource, since the number of units is always one.

The plan generated by ASPEN must satisfy the resource reservations of all the activity instances in the plan. Each resource is represented in the plan as a timeline. The timeline is segmented into units, each with a start and end timepoint. The timepoints correspond to the start and end timepoints of the activity to which the resource was allocated. If there are multiple overlapping reservations for the resource, the timeline is split into units for each segment of the overlap (reservation A, reservation A and B, reservation B) as shown in Figure 2. Both aggregate and atomic resources are represented this way. Atomic resources are essentially an aggregate resource with a capacity of one. In a valid plan, the reservations for all segments must not exceed the resource capacity.

```
Parameter string ALI_Mode {
  domain = ("data","idle","standby","off");
};
Parameter int warprange {
  domain = [1,40960];
};
```

Figure 3: Parameter Type Definitions

State Timelines and Reservations

State timelines represent the evolution of some aspect of the world (spacecraft) over time. A state timeline has an enumerated set of discrete state values that it can take on, and a list of legal state transitions. The timeline must have exactly one value at any given time.

Activities can impose two kinds of state reservations: a "must_be" reservation that requires the state have a specific value for the duration of the activity, and a "change_to" reservation that changes the state to a specific value at the beginning of the activity. As with resources, the timeline is divided into segments according to the timepoints of the "change_to" reservations, and each segment has one value. If there are multiple change_to reservations for a segment, one of them is satisfied and the others are marked as conflicts. The must_be reservations do not segment the timeline. They are merely checked against the value of the timeline over the relevant time period, and are marked as satisfied if the timeline is in the specified value for the entire period, and marked as a conflict otherwise.

Parameters and Functional Dependencies

An activity can have zero or more parameters. The parameters are essentially local variables of the activity and modify the behavior of its constraints. This allows the user to define one parameterized activity type rather than several types for each parameter combination.

Each parameter has a type. The ASPEN primitive types are integer, string, real, and list. ASPEN also supports user-defined types that are constructed from the primitive types. Some parameter type definitions from the EO-1 model are shown in Figure 3.

Parameters can modify an activity's temporal constraints, resource reservations, and state reservations. They are used as variables in the resource and state reservations to specify the number of units or state value as shown in lines 15 and 21 of Figure 4.

Temporal constraints have an optional "with" clause that specifies the parameter values of the relation's target activity. The specified value can either be a constant or the value of one of the source activity's parameters. Lines 9 through 11 of Figure 4 show a tem-

```
1 Activity Take_Image {
2  int image_size;
3
4  Duration = [1,60];
5  Dependencies =
6    image_size <- size_from_dur(duration);
7
8  Constraint =
9    starts_after end of SAD_Changer
10       with (new_mode<-"fixed") by [100,300],
11    ends_before start of SAD_Changer
12       with ("tracking"->new_mode) by [16,16],
13    ...;
14  reservations =
15    use warp_storage image_size;
16 };
17
18 // change mode of solar array drive (SAD)
19 Activity SAD_Changer {
20  SAD_Mode new_mode;
21  reservations = SAD change_to new_mode;
22 };
```

Figure 4: Activity with Parameter Dependencies

```
int size_from_dur (int duration) {
   return 1000 * duration; /* 1K per sec */
};
```

Figure 5: 'C' code for `size_from_dur`

poral constraint between a Take_Image activity and a SAD_Changer activity. The with clause constrains the new_mode parameter of the SAD_Changer activity to have the value "fixed." The "with" constraints are called *functional dependencies* since they define one parameter value to be a function of some other parameter value. In this case, the function is simple equivalence.

More complex functional dependencies are possible among the parameters of an activity. A parameter can be defined as a user-defined function of one or more other parameters of the same activity. This is useful for defining constraints that would otherwise be difficult or impossible to express in ASPEN. It is also useful for linking with existing planning support systems, as will be discussed later.

The dependency functions are written in 'C' and linked into the ASPEN runtime image. Line 6 of Figure 4 shows a functional dependency that computes the size of an image as a function of the image duration. The function definition is shown in Figure 5. Other programming languages can be supported with foreign function interfaces from 'C'.

Modeling Common Mission Operations

The ASPEN modeling language is designed to be used by domain experts, such as mission operations personnel, who generally are not familiar with AI planning techniques or the ASPEN planning algorithms.

The ASPEN planning concepts map onto many of the mission planning concepts with which operations personnel are already familiar. This is intended to make it easier for them to build ASPEN models, since they do not have to learn many new concepts. Furthermore, the ASPEN language makes it easy to express that knowledge. Activities, states, and resources are distinct planning elements, which as we will discuss later is how operations personnel refer to them. Likewise, temporal constraints, resource reservations, and state reservations are distinct entities, and obey clear semantics. States and resources are declared using simple terms familiar to operations personnel. Finally, many aspects of spacecraft activity planning can be expressed in a straightforward and natural manner in ASPEN.

ASPEN also facilitates modular design of plan models, and requires no knowledge of the underlying plan engine. These issues will be discussed in the two following sections. The remainder of this section describes the ASPEN language in more detail, and demonstrates how ASPEN can model many common aspects of spacecraft activity planning in a straightforward and intuitive manner.

Resources

The driving constraints on most spacecraft missions are the resources, since most spacecraft are resource limited. A large portion of the planning problem is scheduling activities in a way that avoids resource conflicts. Typical spacecraft resources are propellant (fuel), power, data storage, and battery capacity. These are *aggregate* resources: they have several units that can be allocated among multiple users (activities) at any time. The ability to reason about aggregate resources is a primary requirement of the mission activity planning domain. Spacecraft devices that can only be used for one activity at a time, such as a science camera, are also resources. These are *atomic* resources: they have a capacity of one and can be used by at most one activity at a time.

Operations personnel are concerned with certain information about resources for planning purposes. They need to know the resource capacities, usage profiles, and availability profiles. Each resource has some maximum capacity, and may have a non-zero minimum capacity. For example, batteries loose their effectiveness below a certain discharge level, and so have a effective

```
Resource Propellant { type = depletable;
                      capacity = 1500; //grams}
Resource Power      { type = non_depletable;
                      capacity = 300; //watts}
Resource Data_Store { type = depletable;
                      capacity = 3,000;//KB}
Resource Battery    { type = depletable;
                      capacity = 15; //Ah
                      min_capacity = 7;}
Resource camera_A   { type = atomic; }
Resource camera_B   { type = atomic; }
```

Figure 6: ASPEN Resource Definitions

minimum capacity that is non-zero . Minimum levels may also be assigned for operational or spacecraft safety reasons (e.g., do not use let the propellant level drop below ten kilograms during this mission phase).

Resources usage and availability profiles are very important for planning. Operations personnel need to know how each activity will consume resources over time, and whether the availability of that resource is constant or varies with respect to time or some aspect of the spacecraft state. For example, propellant usage is roughly linear in the duration of a burn. Solar power availability depends on the spacecraft attitude and position with respect to the Sun.

Let us consider how this resource knowledge would be represented in ASPEN. First, each resource is declared with a simple expression that specifies its name, type (depletable, non-depletable, or atomic), and minimum and maximum capacities (if aggregate). These declarations are very straightforward, as can be seen in Figure 6. Data storage and battery capacity are depletable but renewable resources (batteries can be recharged and data can be deleted after it is downlinked). Fuel is depletable but non-renewable, and power is a non-depletable resource. This representation is sufficient to express all of the spacecraft resources we have encountered.

Resource usage profiles depend on the activities that use them. Each activity in ASPEN has an optional resource *reservation*. This states the resource name and the number of units that the activity will use. For atomic resources, the number of units is omitted (it is unnecessary, since there is only one unit).

The number of units can be a constant, or a user-defined function of the activity's parameters, including the implied parameters of start time, end time, and duration. For example, a thruster burn uses a number of fuel units that is linear in the duration of the burn.

ASPEN assumes that all the resource units are consumed at the beginning of the activity, even if the actual usage varies over time. This is a conservative assumption that simplifies the ASPEN engine. It is conservative in that it permits a subset of the schedules that would be legal if it did more detailed resource profiling. More detailed profiling would allow some activities to overlap that would not be allowed to overlap with simpler modeling. We are considering more detailed resource profiling for future versions of ASPEN.

At present, ASPEN can only represent constant resource availability profiles. That is, the resource has some constant capacity, and does not vary with time or other state variables. This is sufficient for most resources, so it is not a strong limitation. For the few places where it is necessary to have varying availability profiles, this can be implemented using activities that create or allocate the resource at appropriate times. We are considering extensions to ASPEN that would allow resource capacity to be modeled directly as a function of state or time. This extension would make the model clearer, but not extend the representational power of ASPEN.

States

Operations personnel must know about the spacecraft states relevant for planning purposes, and how they interact with the spacecraft activities. They are primarily concerned with states that are preconditions of activities or are affected by those activities. Other states do not affect planning and do not need to be considered.

Typical states that operations personnel are concerned with are device states, and global spacecraft state. Device states are things like "valve open or closed" and "engine is off, firing, or warming up." Devices are often limited to certain state transitions. For example, the engine can go from "off" to "warming up" to "on", but not from "off" to "on" directly. There are often duration constraints on the device states as well. For example, the engine must warm up for at least thirty minutes before firing.

Global spacecraft states are things like spacecraft attitude and vibrational level. They do not correspond to any one device, and can be affected by several activities. Global states are often preconditions of activities as well (e.g., the vibrational level must be low in order to take a picture).

This state information is represented in ASPEN as state timelines. State timelines in ASPEN describe the evolution over time of some aspect of the spacecraft. Each state timeline is specified in the model by a simple declaration that specifies its name, states, and legal state transitions. The allowable state transitions are indicated using the **transitions** keyword with a forward arrow ($\rightarrow$), a bi-directional arrow ($\leftrightarrow$), or with

```
State_Variable camera_health {
  states = ("ok", "degraded", "broken");
  default_state = "ok";
  transitions = all;
};

State_Variable camera_mode {
  states = ( "on", "off", "warming_up");
  default_state = "on";
  transitions=("off"<->"warming_up",
               "warming_up"->"on", "on"->"off");
};

State_Variable vibration {
  states = ( "low", "high" );
  transitions = all;
};
```

Figure 7: Common Spacecraft States

```
Activity instr_on {
   constraints =
     ends_before start_of instr_off by [10,3600];
   reservation = instr_sv change_to "on";
};

Activity instr_off {
  reservation = instr_sv change_to "off";
};
```

Figure 8: Representing State Duration Constraints

the `all` keyword. If the transitions are not specified, ASPEN assumes that all transitions are legal. The initial value for a state timeline can be specified by the optional `default_state` keyword. Declarations for some common spacecraft states are shown in Figure 7.

Constraints on the duration that the spacecraft can be in a given state are represented in ASPEN by temporal constraints on the activities that change the state. An activity that changes a state variable to one of its duration-limited values must be followed within a given duration by an activity that changes it to a different value. This is represented by temporal constraints on the first activity, as shown in Figure 8. These constraints say that the once the instrument enters the "on" state, it must stay in that for at least 10 seconds and no more than an hour.

We are considering extensions to ASPEN that would allow state duration constraints to be specified within the state variable declaration. This will not extend the expressiveness of ASPEN, but may make the modeling a little more intuitive.

Activity Type:
Title:
Description:

Used in following activities: *subactivity of*
Implements following activities: *parent of*

COMMANDING SCENARIO

Parameters:
activity parameters

Overview:
English description of activity

Commands:
Relative time Activity/Cmd Parameters

PRECONDITIONS
i.e., must_be state reservations

CONSTRAINTS/RESOURCES
i.e., temporal constraints and resource reservations

POST CONDITIONS
i.e., change_to state reservations

NOTES:

Figure 9: Template for an Activity Type.

Activities

Activities are events or actions that the spacecraft performs. Operations personnel begin by defining all the spacecraft activities using an "activity type template" similar to the one shown in Figure 9. The template describes the operational constraints, resources and effects of the activity. It also specifies the low-level command sequence that implements the activity.

The main components of the template are the activity name and parameters, the preconditions for performing the activity (spacecraft state and resources), the effect of performing the activity, and the low-level command sequence that implements the activity. The activity type is used in the mission plan for human consumption, and the sequence is used for commanding the spacecraft.

The operations personnel in conjunction with the subsystem engineers determine what the relevant activities are for each subsystem and what their operational constraints are. This information is captured in activity type definitions and used by the operations personnel for planning purposes.

Activity Type:	**Science_Obs_Act**
Title:	Science Observation
Description:	Take image of science target

Used in following parent activities: N/A
Implements following activities: N/A

COMMANDING SCENARIO

Parameters:
target_attitude
camera_filter

Overview:
Select filter
Take image

Commands:

Relative time	Activity/Cmd	Parameters
0T00:00:00	FILTER_SLCT	camera_filter
0T00:00:10	TAKE_IMAGE	

PRECONDITIONS
a) pointing camera at target_attitude
b) low vibration

CONSTRAINTS/RESOURCES
a) 10 KB of data storage
b) 15 Watts of power

POST CONDITIONS
a) Image is in data storage
b) filter is selected

NOTES: None

Figure 10: Science Observation Activity as Operations Personnel Would Define It.

An ASPEN activity definition is very similar conceptually to the activity type definitions familiar to operations personnel. Consider how a science observation activity would be defined in ASPEN and with an activity type template. The activity type definition is shown in Figure 10, and the corresponding ASPEN definition is shown in Figure 11.

An ASPEN activity type has a name, zero or more parameters, a duration, an optional sequence expansion, and optional lists of temporal constraints, resource reservations, and state reservations. These all correspond to parts of the mission operations activity type definition. The preconditions and postconditions of the operations activity type correspond to must_be and change_to state reservations in the ASPEN activity definition. The constraints section corresponds to the temporal constraints in ASPEN. The operations activity type does not have an explicit duration, but the

```
1 Activity Science_Obs_Act {
2   //parameters
3   attitude target_attitude;
4   filter_id filter;
5
6   reservations =
7     // preconditions
8     attitude_sv must_be target_attitude,
9     vibration_sv must_be low,
10
11    // resources
12    use data_storage 10, //KB
13    use power 15; // Watts
14
15 // (temporal) constraints: none
16 // Constraint = ... ;
17 };
```

Figure 11: Science Observation Activity as Defined by ASPEN.

duration is implicit in the command sequence. The other elements of the operations activity type correspond to obvious parts of the ASPEN activity type definition. These elements are discussed in more detail in the following subsections.

Sequence Expansion. An activity type template includes a section for defining the sequence that implements the activity. ASPEN has a similar capability. Each activity has an optional **command** keyword that specifies the sequence for that activity. This directive is used in a post-processing phase to convert the plan into a sequence that can be executed on the spacecraft.

Duration. The duration of an activity can be specified as a range $[x, y]$ or a constant duration. If not specified, it defaults to [1, infinity]. The duration can be extracted from the command sequence in the operations activity type definition.

Reservations. A resource reservation allocates a resource to an activity for the duration of that activity. State reservations either change the value of a state variable or reserve a state value for the duration of an activity. State changes correspond to effects of activities as defined in the activity type template, and state reservations correspond to preconditions. A state change is represented in ASPEN by a "change_to" state reservation, and a state requirement is represented by a "must_be" reservation. See Lines 8 and 9 of Figure 11, respectively.

Temporal Constraints. These specify temporal relations that must hold between pairs of activities. The

```
Activity ScienceObs {
  subactivities =
    select_filter a with (filter_id<-"A"),
    take_image b,
    select_filter c with (filter_id<-"B"),
    take_image d
  where
    a starts_before end_of b by [0,0],
    b starts_before end_of c by [0,0],
    c starts_before end_of d by [0,0];
};
```

Figure 12: Activity with subactivities.

ASPEN modeling philosophy urges the use of reservations instead of temporal constraints wherever possible in order to increase the modularity and readability of the model. However, it is necessary and appropriate to use temporal relations to represent temporal constraints on states.

One class of situations where this occurs is when an activity requires that some state has been established for some duration. The requirement that the catbed heater be on for at least half an hour before firing the main engine is a good example of this, and has been discussed.

Another class is states that have some constraint on their duration. For example, some instruments should not be left on too long; other devices should be left on for a minimum duration before being turned off to avoid damage. An effective way to do this is to place temporal constraints between the activities that change the state, as was discussed in the section on states.

Subactivities. ASPEN allows activities to have subactivities. These correspond to the "used in the following activities" and "implements following activities" sections in the activity type template. Subactivities are useful for representing the steps of high-level activities that are always performed the same way. For example, suppose each science observation consists of selecting filter A, taking an image with filter A, selecting filter B, and taking an image with filter B. This would be defined as a "science_observation" parent activity with three subactivities as shown in Figure 12. The parent activity declaration can include temporal relations among the subactivities. In the example of Figure 12, they force the four subactivities to be performed in the correct order.

It is often convenient to think of activities in this way, and it can simplify the planning search. Subactivities are implemented as regular activities, except that ASPEN maintains a link between the subactivities and their parent. This allows the planning engine to treat parents and subactivities as a block where appropriate. For example, it can schedule the parent and its subactivities all at once, or move them as a unit to resolve conflicts. This is often more efficient than reasoning about each activity in isolation.

```
Activity science_obs {
   target t;
   dependencies =
     score <- f(t,start_time);
};
```

Figure 13: Activity Preference Score

Optimization Criteria

Spacecraft mission are almost always oversubscribed. There are more science requests and other objectives than can possibly be met. It is up to the operations personnel to devise an operations plan that achieves as many goals as possible (weighted by priority). The operations personnel must optimize the plan to eke out every last bit of performance.

ASPEN can optimize plans, removing some of the optimization burden from the operators. ASPEN cannot optimize plans as well as experienced operators, though. ASPEN has a mixed-initiative mode and a GUI that facilitates hand-optimization, if desired.

In order for ASPEN to optimize a plan, the optimization criteria must be specified in the model. There are two ways to specify preference criteria in ASPEN. The first is to define a score function within an activity. This is a user-defined function that computes a preference score for the activity from the activity parameters. An example is shown in Figure 13. The user-defined function (f in the example) is an arbitrary 'C' function.

The second way is to use the ASPEN preference language. This allows the user to score a partial plan as a function of activity parameters (including start time, end time, and duration), resource utilization, number of state transitions, or number of activity instances of a particular type. The functions are limited to a predefined set, namely a linear function or an exponential function. Examples are shown in Figure 14.

Both of these methods allow preferences to be stated in strictly domain-specific terms. The user does not need to know anything about the planning engine. The preference language is sufficient to express common spacecraft optimization criteria. Typical criteria are maximizing activity instances (e.g., the number of science observations), optimizing activity parameters

```
// maximize number science observations
prefer linearly more science_obs activities;

// maximize propellant remaining at end of plan
prefer exponentially more propellant at end;

// minimize number of times camera A
// changes state
prefer linearly fewer cam_a_sv units;
```

Figure 14: Preference Examples

(e.g., take the most interesting observations), and minimizing resource utilization.

External Interfaces

A practical mission planning system must interface with other mission-planning support systems. Typical systems perform computations that cannot be easily done within a planner. Typical support systems compute trajectories, attitudes needed to take science observations as a function of spacecraft position and how the target is moving, determine when Earth tracking stations can communicate with the spacecraft, etc.

ASPEN provides two interfaces to such tools. One is the initial state file, and the other is dependency functions. The initial state file is what defines the initial schedule. This file typically contains the initial state of the spacecraft and activities (goals) that the user wants to be achieved. External systems often have a hand in defining ASPEN's goals or computing resource levels. Systems that can reason about orbital information are often needed to compute science targets, mosaics, and trajectories, which can then be communicated to ASPEN as activity parameters in the initial state file. The initial state file is also a good way to inform ASPEN of events such as communication passes and occultation periods (when the spacecraft is in shadow, for Earth orbiters). These are often determined by systems or procedures external to ASPEN.

The other interface is through parameter dependency functions. As discussed earlier, an activity parameter can be a user-defined function of one or more of the other parameters. The user-defined function can be any 'C' function, and can therefore call external support systems or simulators.

For example, the duration of a slew (change of spacecraft attitude) depends on the start and end attitudes, the start time of the slew, and the turn rate. This function can be quite complex. Spacecraft are usually subject to pointing exclusion constraints–for example, they cannot point instruments and radiators too near the sun and other bright objects. The spacecraft may have to "walk around" these exclusion zones in order to reach the target. The zones change with time, since the spacecraft is moving with respect to the Sun and other bright bodies (e.g., Jupiter). An example is shown in Figure 15.

```
Activity slew {
   attitude from;
   attitude to;
   real turn_rate;

   dependencies =
     duration<-slew_dur(from,to,
                        start_time,
                        turn_rate);
   ...
};
```

in functions.c:

```
int slew_dur (char* from, char* to,
              int start_time,
              float turn_rate)
{ ... };
```

Figure 15: Dependency Function

Modularity

The ASPEN language facilitates modular design of models. This is important, since it allows experts in different subsystems to create their part of the model without having to coordinate strongly with other experts. This reduces the knowledge acquisition bottleneck. It also localizes the knowledge for each subsystem, which should improve maintenance and verification of the model.

What makes this modularity possible is ASPEN's philosophy that activities should interact through states and resources. The activities can be ignorant of each other, which allows activities to be added, deleted, and modified without having to change the other activities. Of course, if changes are made to the states or resources, then all the activities will have to be modified accordingly.

To see how activities interact through states and resources, consider a science observation activity that needs to be pointing at Jupiter, have low spacecraft vibration, have access to the camera, and uses three megabytes of data storage. The non-modular way to enforce these conditions is to impose temporal relations with activities that establish those conditions. That is, the observation activity would require a "point to Jupiter" and "turn camera on" activity some time before it, and disallow during the observation all other activities that use the camera.

Resource and state reservations enforce these conditions in a modular way without using temporal constraints. The ASPEN planning engine will ensure that these states and resource requirements are met. It finds activities that establish the desired states, and inserts them at appropriate places in the plan.

Heuristics

ASPEN has facilities for instrumenting every choice point in the search with heuristics. However, heuristics are not required for most models (all of the ASPEN models developed so far use only a standard set of six domain-independent heuristics). This independence is made possible by the ASPEN engine design, which compensates for weak search knowledge by searching more. ASPEN uses an iterative repair search, in which each repair operation is relatively inexpensive computationally.

The ASPEN design is predicated on this principle of fast, inexpensive search. This principle is reflected in many design decisions, but a large portion of the speed gains come from ASPEN's committed approach to planning. Although the plan model represents activity timepoints as intervals, those timepoints are narrowed to fixed times when the activities are added to the schedule. This simplifies the plan representation, conflict detection algorithms, and plan modification operations, and enables faster plan operations (the design goal is to average 100 plan operations per second.)

The benefit of faster search is that ASPEN models do not require domain-specific heuristics. It is much easier to develop plan models when heuristics are not required. First, heuristics require that the modeler have some understanding of the planning algorithm. Mission operations personnel generally do not have this understanding, nor should they have to. Second, domain-specific heuristics often break modularity. They must know what activities and goals are in the model, and how they interact during the planning search. When the heuristics need to be very strong and finely tuned in order to achieve acceptable planning performance, developing the heuristics can become the most difficult part of the modeling task (Smith, Rajan, & Muscettola 1997).

Future Extensions to ASPEN

The ASPEN modeling language is expressive and intuitive, but there are some modeling tasks that could be made even more intuitive by extending the modeling language. The extensions discussed here generally do not increase the expressiveness of ASPEN, they just allow things to be represented in a way that more naturally reflects how operations personnel perceive them.

ASPEN presently presumes that activities consume all of their aggregate resource reservations at the beginning of the activity. The actual resource usage profiles are often linear or decaying exponential functions over the duration of the activity. In the future, ASPEN may have such resource usage profiles, but for now it is a known limitation. This extension would increase the expressiveness of ASPEN.

Other possible extensions that have already been discussed are allowing states to have duration, and computing resource availability as a function of states. These extensions will not extend the representational capability of ASPEN, since all of these things can be represented in the current version of ASPEN by using activities in appropriate ways, but they may make the modeling language more intuitive.

Another problem that is difficult to express in ASPEN, or any other planning languages we are familiar with, is power interaction between the solar array and the battery. This interaction arises in EO-1 and many other spacecraft. When the EO-1 satellite is occulted by the Earth, activities in the plan which use solar array power must instead use battery power. The reverse is true when EO-1 is in direct sunlight. There are also periods where both solar array and batteries are used for power due to partial illumination of the solar array. Combining these effects with the complex charging and discharging cycles of the batteries creates a difficult problem to model.

One way to represent this problem, which ASPEN does not directly support, would be to define a "power" resource whose availability profile (capacity) is determined by a battery resource and a solar-power resource. The availability of the battery and solar-power resources can be determined as a function of the spacecraft position and attitude with respect to the sun as determined by a state timeline. The availability of the combined power resource would be a function of the battery and solar-power resource availability[1]

This does not increase the expressiveness of Aspen, since this can can be represented with parameter dependency functions in the activities that use power. Each activity would have one function to compute the solar power consumed and one for the battery power.

This representation would require adding a resource availability function to resources, and allowing resources to reserve other resources. The first would allow the battery and solar-power availabilities to be

[1]Simply adding the availabilities may not be sufficient, since solar and battery power often have non-linear interactions. The battery is recharged by solar power, and the battery compensates for drops in solar power output. This connection creates some complex interactions between the two.

computed from the sun-position timeline; the second would allow the combined power resource to reserve some combination of battery and solar power to satisfy reservations on the combined power resource.

For all of these extensions, some care must be taken before adding them. They all involve allowing states and resources to be the source of various constraints. This might confuse users by cluttering ASPEN's clean semantic, which only allows constraints to come from Activities. On the other hand, users may find this more intuitive and not perceive it as cluttering at all. Further study is needed to see if these will in fact improve the ease of modeling.

The EO1 Model

We now provide a description of the ASPEN planning model for the EO-1 mission. This model was developed by a one of us [Sherwood] who has a strong mission operations background, but no background in AI or automated planing. This description is intended to show that planning models are easy to express in ASPEN, and that they can be developed by operations personnel with no AI background.

EO-1 is an Earth imaging satellite to be launched in May 1999. It is part of the New Millennium Program of technology validation missions. The NASA Goddard Space Flight Center is responsible for project management. The purpose of EO-1 is to validate new technologies that can be used on future Landsat class Earth remote sensing missions. In fact, EO-1 will be flying in formation one minute behind Landsat-7, with the goal of imaging as many of the same targets as possible. EO-1 will be using the Landsat 7 daily scene list as an input file of potential EO-1 targets.

The science payload on EO-1 is an advanced multispectral imaging device. Mission operations on EO-1 consist of managing spacecraft operability constraints such as power, thermal, pointing, buffers, consumables, and telecommunications. EO-1 science goals involve imaging of specific targets within particular observation parameters. Managing EO-1 spacecraft downlink is particularly difficult because the amount of data generated by the imaging device is quite large and ground contacts are limited. In addition, because science targets for EO-1 are based on short-term cloud predictions, schedules must be generated daily.

The main activity in EO-1 operations is the Advanced Land Imager (ALI) data take. The ALI instrument contains six separate detectors that output data simultaneously. One image takes a total of 24 seconds and consumes about 19 gigabits of data on the solid state recorder (WARP). Because the capacity of the WARP is only 40 Gbits, it is important to plan the data takes and downlinks to maximize the amount of data returned. Due to a limited amount of available downlink time, only four data takes per day can be performed. Data takes can be prioritized based on the following parameters:

- Cloud cover over the region to be imaged
- Sun angle at the region to be imaged
- Ability to return the data before overflowing the WARP recorder
- Images coinciding with Landsat 7 images
- Imaging of scientifically interesting areas

Each EO-1 data take has several conditions that must be satisfied before and after the data take occurs. These conditions are listed below:

Before:

- Change the ACS mode to science
- Change the solar array to a fixed orientation
- Open the ALI aperture
- Change the data rate to high rate mode

After:

- Close the ALI aperture
- Take one second of calibration data
- Change the ACS, solar array, and data rate modes back to the previous values

Each of these conditions is modeled as temporal constraints in the ALI data take activity. The data take activity itself is only a 24-second activity. The constraints on the activity span a period of five minutes before and one minute after the bounds of the activity. The constraints on the activity could have been modeled as subactivities. The reason we chose to model these activities as constraints is because of their tight temporal constraints. The data take activity breaks down into 14 separate activities as listed in Table 1.

The ALI must be calibrated by viewing the sun or the moon regularly. The sun calibration involves pointing at the sun and changing the aperture filter several times. The moon calibration points at the lunar limb and pans across the moon using each of the detectors. Similar to the data take activities, the calibrations involve several constraints. The calibration activities and constraints are listed in Table 2.

EO-1 communication activities are modeled as follows:

```
1  State_variable ALI_sv {
2    states = ("data","standby","idle","off");
3    transitions = ( "standby"->"data",
                     "data"->"standby",
                     "idle"->"standby",
                     "standby"->"idle",
                     "off"->"idle",
                     "idle->off");
4    default_state = "idle";
5  };
6
7  State_variable aperture_sv {
8    states = ( "open", "closed");
9    transitions = ( "open"->"closed",
                     "closed"->"open" );
10   default_state = "closed";
11 };
```

Figure 16: State Variable Examples

ALI_data_take	aperture_changer
ALI_user_data	ALI_user_standby
ALI_changer	SAD_user
SAD_changer	aperture_user
engdata_user	engdata_changer
ACS_user	ACS_changer
cloud_cover_changer	sun_angle_changer

Table 1: EO-1 Science Activities

1. An input file gives the times at which the ground station is in view of the satellite.
2. The in view times are converted into a state variable with the value 'inview' or 'outview.'
3. The planner chooses communication links during these in view times.
4. The communication link is broken down into uplinks (if required) and downlinks.

The EO-1 model also includes initialization activities

ALI_sun_calibration
slew_to_sun
aper_test_changer
ALI_moon_calibration
moon_cal_ms_pan
slew_to_moon
ramp_up_pitch_slew
ramp_down_pitch_slew
roll_to_next_position

Table 2: EO-1 Calibration Activities

Non-Depletable	Depletable
ALI	Battery
S-band Receiver	Warp Storage
Transponders	Propellant
solar array	
ACDSE	
Warp	
Processor	
Bus_1773	
Cat_bed_heater	
WFF	
DSN	

Table 3: EO-1 Resources

for power, propellant, and memory. These activities are used to keep track of consumable resources from the previous planning period.

The **command** key-word is used for activities that represent an EO-1 spacecraft command. When the command keyword is included in the activity definition, along with the command name, the spacecraft command output file will include a time tagged command for that activity.

The EO-1 spacecraft resources are modeled as either depletable or non-depletable. It was not necessary to model every physical device on EO-1 because many devices consumed a constant power and did not interact with any spacecraft activities. The power of these devices is included in the power_init activity. The resources that are modeled are listed in Table 3.

The EO-1 ASPEN model has ten different state variables, which are listed in Table 4. Most of these state variables are used to represent the state of a spacecraft resource. The states are used in activities that require a resource to be in a particular state. These requirements are specified in the reservations of the activity. For example, the EO-1 data take activity requires the WARP state variable to be in record mode during the period of imaging. This requirement ensures that the data is being recorded during the imaging operation. Activities are defined that either change or use a particular state of a state variable. These activities usually contain a command keyword that corresponds to an EO-1 spacecraft command.

Creating the EO-1 Model

The modeling language has been designed such that it can model a physical spacecraft system directly. It is a descriptive language that allows an engineer to directly represent the physical spacecraft information in the model. In fact, the EO-1 model was created by one

Variable	States
ALI_sv	data, standby, idle, off
SAD_sv	off, tracking, fixed
aperture_sv	open, closed
aperture_test_sv	small, med, large, blank
engdata_sv	high, low
ACS_sv	nadir, low_jitter, standby, safe, orbit_adjust, WARP_sv off, idle, record, playback
Cloud_Cover_sv	low, med_low, med, med_high, high, none
Sun_Angle_sv	low, med, high, none
WFF_inview_sv	in, out

Table 4: EO-1 State Variables

of us (Sherwood) who had no knowledge of the software or its algorithms and procedures. He successfully created the model by simply taking the EO-1 spacecraft information and putting it into the modeling language syntax.

After building the EO-1 model, Sherwood built a model for another mission, called UFO-1. The UFO-1 model is comparable in complexity to EO-1, and will be generating all the command sequences for operating UFO-1 beginning October 1999. The modeling effort for both of these models is summarized in Table 5.

Although the UFO-1 model is larger than the EO-1 model, it took only a third as much effort to build. The modeler (Sherwood) attributes this improvement to increased familiarity with the ASPEN language after having built the EO-1 model. It would be interesting to see whether this trend holds for other modelers and missions.

The modeling language is flexible and allows for different ways of representing the same information. Therefore, there is no one correct model for a given spacecraft. The EO-1 model is constrained to have certain state and resource variables as determined by the mission, but, on the other hand, there are different ways of representing constraints among activities.

End-to-End Planning System

The goal of this EO-1 work is to produce an automated on-board planning system for spacecraft commanding of the EO-1 satellite. The system will be validated after launch on the ground. As a ground based planner, the inputs to ASPEN include:

- Landsat-7 cloud cover and sun angle predictions
- Current power, propellant, and memory levels
- Sun, moon, and sky calibration requests
- Ground station view files
- Maneuver requests

Once ASPEN is delivered to the EO-1 project, there will only be minor changes made to the model to integrate ASPEN into the existing operations. We plan to automate the loading of the input files such as cloud cover and sun angle predictions into ASPEN, and link the output schedule of ASPEN directly to the existing EO-1 software. In fact, the creation of the input files can be invoked from external calls from the ASPEN GUI. With ASPEN linked directly to its input files through the GUI, the EO-1 planning process will be seamless and efficient.

The output of the ground based validation of the planner will be a text list of time tagged commands that will be translated into binary spacecraft commands by the ground system load generation utility. This utility is already built into the EO-1 ground system.

The on-board planning system will require upload of the ground station view files and maneuver requests. The cloud cover could be obtained by using the ALI science instrument to examine the clouds before a scene. After the image is taken, the cloud data would be analyzed to determine if the scene should be saved and downlinked. Clouded scenes would be erased from the WARP and a new scene would be planned to take its place.

The EO-1 Model in Action

Generating EO-1 mission operations schedules is a fast process. Given a set of EO-1 requests, ASPEN will generate a conflict-free schedule within the order of a few minutes for lengthy schedules, and within seconds for simpler schedules. For example, for 162 EO-1 activities, it takes ASPEN 3.53 seconds (on a SUN Ultra-2) to produce a conflict-free schedule. There are no EO-1 schedules that take more than a minute to schedule, but with other spacecraft models with more activities and lengthier schedules, we have seen a maximum of five minutes to produce a conflict-free schedule.

In addition to having the activity requests specified in advance, the user can make changes to the schedule from the GUI as needed. For example, the user could add an ALI_data_take activity. If this caused conflicts in the schedule, then ASPEN would resolve the conflicts. This whole process takes seconds to execute. For example, with the EO-1 model, if we add three ALI_data_take activities in the GUI (randomly placed), this causes 34 conflicts. Resolving all conflicts, and producing a conflict-free schedule takes 1.54 seconds (on a SUN Ultra-2). This means that it is solving approximately 17 conflicts per second. (Adding just one

Model	Effort	Model			Activities in Plan
		State Vars	Activity Types	Resources	
EO-1	3 weeks	10	38	14	172
UFO-1	1 week	18	79	14	159

Table 5: Modeling Effort

data-take activity causes a large number of conflicts because of the constraints between activities and the states required by different activities).

Currently, activities can be given a particular score, and high-level preferences (such as resource max usage) can be indicated which also determine scores for activities. The generated schedule is then given a score based on the activities' scores. Using this score, the user can then choose one generated schedule over another. We are presently working on an algorithm that will automatically optimize schedules.

Related Work

Plan-It II

The Plan-It II planner (Eggemeyer 1997; Eggemeyer *et al.* 1997) has been used for mission activity planning on a number of JPL missions. It has many of the same concepts as ASPEN, but was not intended for use by non-AI experts. While there have been numerous significant deployments for actual flight projects (e.g., Galileo, Mars Pathfinder, Deep Space 1), these were all developed by Plan-It II experts, not generic mission operations personnel. The syntax is built on top of Lisp, so it is helpful to have at least a basic understanding of Lisp.

Planit-II requires the user to provide domain-specific scheduling algorithms in Lisp. This generally requires some familiarity with Lisp and AI planning techniques. The scheduling methods are expected to use primitives from the Plan-It engine for detecting and resolving conflicts. This requires knowledge of the underlying planning system.

HSTS

The HSTS (Muscettola 1994) planner is scheduled to generate plans for NASA's DS-1 spacecraft for a period of one week in October 1998 (Muscettola *et al.* 1997). The plans will be generated onboard the spacecraft. Like ASPEN, HSTS reasons about metric time and aggregate resources, and its modeling language is clearly sufficient for expressing spacecraft activity planning knowledge.

The HSTS language has a Lisp-like syntax and does not distinguish among activities, states, and resources. HSTS represents all of these elements as a single type called a *token*. This makes for a clean planning semantic, but does not map as well onto the way in which operations personnel think about mission activity planning.

In order to achieve reasonable planning speed for DS1, the plan model required detailed heuristics. Developing and fine-tuning these heuristics required significant effort and an intimate knowledge of the HSTS planner(Smith, Rajan, & Muscettola 1997).

Conclusion

The goal of the ASPEN modeling language is to make it easier for non-AI experts to encode mission activity planning knowledge. Having domain experts build the planing models is important for eliminating the knowledge acquisition bottleneck and for increasing the accuracy and maintainability of the model.

ASPEN is intended to allow operations personnel to express planning knowledge naturally and intuitively, without requiring knowledge of the underlying planning algorithms. ASPEN also facilitates modular design, which allows subsystem experts to create different parts of the model.

ASPEN planning models have been developed by operations personnel for two spacecraft, EO-1 and UFO-1, and are scheduled for use on those missions.

Acknowledgements

This paper describes work performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract from the National Aeronautics and Space Administration.

References

Allen, J.; Hendler, J.; and Tate, A., eds. 1990. *Readings in Planning.* Morgan Kaufmann.

Eggemeyer, C.; Grenander, S.; Peters, S.; and Amador, A. 1997. Long term evolution of a planning and scheduling capability for real planetary applications. In *Proceedings of the 1997 International Workshop on Planning and Scheduling for Space Exploration and Science.* Oxnard, CA: NASA/JPL.

Eggemeyer, C. 1997. *Plan-It II Bible.* Jet Propulsion Laboratory, Pasadena, CA.

Fox, M., and Zweben, M., eds. 1994. *Intelligent Scheduling.* Morgan–Kaufman Publishers.

Fukunaga, A.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Towards an application framework for automated planning and scheduling. In *Proceedings of the IEEE Aerospace Conference.*

Muscettola, N.; Smith, B.; Chien, C.; Fry, C.; Rajan, K.; Mohan, S.; Rabideau, G.; and Yan, D. 1997. On-board planning for the new millennium deep space one spacecraft. In *Proceedings of the 1997 IEEE Aerospace Conference*, volume 1, 303–318.

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Fox and Zweben (1994).

Smith, S.; Lassila, O.; and Becker, M. 1996. Configurable mixed-initiative systems for planning and scheduling. In Tate, A., ed., *Advanced Planning Technology.* AAAI Press.

Smith, B.; Rajan, K.; and Muscettola, N. 1997. Knowledge acquisition for the onboard planner of an autonomous spacecraft. In Benjamin, R., ed., *Proceedings of the 1997 European Knowledge Acquisition Workshop*, Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag.

Smith, S. 1994. *OPIS: A Methodology and Architecture for Reactive Scheduling,* In Fox and Zweben (1994). 29–65.

Zweben, M.; Daun, B.; Davis, E.; and Deale, M. 1994. *Scheduling and Rescheduling with Iterative Repair,* In Fox and Zweben (1994). 241–255.