ASPEN: EO-1 Mission Activity Planning Made Easy

Rob Sherwood, Anita Govindjee, David Yan,
Gregg Rabideau, Steve Chien, Alex Fukunaga
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California 91109
aspen @aig.jpl.nasa.gov

ABSTRACT

This paper describes the application of an automated planning and scheduling system to the NASA Earth Orbiting 1 (EO-1) mission. The planning system, ASPEN, is used to autonomously schedule the daily activities of the satellite. The satellite and operations constraints are encoded within a software model used by the planner. This paper includes a description of the planning system and the associated modeling language. We then discuss how we encoded the EO-1 spacecraft operations with the modeling language. We conclude with a description of the end-to-end planning system as we envision it for EO-1.

INTRODUCTION

Automated planning/scheduling technologies show great promise in reducing operations costs by increasing autonomy of EO-1 mission operations. The Artificial Intelligence (AI) Group at the Jet Propulsion Laboratory' has been working on a system called ASPEN (A Scheduling and Planning Environment). ASPEN [Fukunaga et al. 1997] is a modular, reconfigurable application framework based on AI techniques [Allen et al. 1990, Zweben & Fox 1994], which is capable of supporting a variety of planning and scheduling applications (similar to [Smith et al. 1996]), The primary application area for ASPEN is the spacecraft operations domain.

EO-1 [Speer et al, 1997] is an Earth imaging satellite to be launched in May 1999. The science payload on EO-1 is an advanced multi-spectral 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. Of particular difficulty is managing the downlinks since 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.

Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals. ASPEN encodes spacecraft operability constraints, flight rules, spacecraft hardware models, science experiment goals, and operations procedures to allow for automated generation of low-level spacecraft sequences. By automating the command sequence generation process and by encapsulating the operations specific knowledge, ASPEN will enable EO-1 spacecraft commanding by a small operations team and thereby reduce costs.

ASPEN

ASPEN is an object-oriented system that provides a reusable set of software components that implements the elements commonly found in complex planning/scheduling systems. These include:

- An expressive constraint modeling language to allow the user to naturally define the application domain;
- A constraint management system for representing and maintaining spacecraft operability and resource constraints, as well as activity requirements;
- A temporal reasoning system for expressing and maintaining temporal constraints; and

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

This paper is submitted to the NASA Workshop on Planning and Scheduling for Space.

' For an overview of Artificial Intelligence work at JPL see [Chien et al. 1997].

- A graphical interface for visualizing plans/schedules (for use in mixed-initiative systems in which the problem solving process is interactive).

The central data structure in ASPEN is an activity. An activity represents an action or step in a plan/schedule. An activity has a start time, an end time, and a duration, Activities can use one or more resources. For more details on ASPEN, see [Fukunaga *et al* 1997].

```
1   Activity ALI_data_take {
2     Fixed fi;
3     Tracking tr;
4     Duration = [1 ,60];
5     Constraint =
6           starts_ after end of SAD_changer with (fi->sad1 ) by [100,300],
7           ends_before start of SAD_changer with (tr->sad1 ) by[16, 16],
8           Contains both of SAD_user with (fi->ap1 ) by [4,4,0,10],
9           Contained_by both of SAD_user with (fi->sad1 ) by [300,300,1,1 6];
10    Subactivities = ALI_user_data, ALI_dark_count;
11    Simple_reservations = processor, array_power = 80;
12  };
13
14  Activity SAD_changer {
15    Sad_mode sadl;
16    Simple_reservations = solar_array change_to sadl,
17          aperture must_be lopenî;
18  };
```

Figure 1 Activity Example

MODELING LANGUAGE

The ASPEN modeling language allows the user to define activities, resources, and states as described above. A domain model is input at start-up time, so modifications can be made to the model without requiring **ASPEN to be recompiled. The modeling language has a simple** syntax, which can easily be used by spacecraft mission operations personnel to create a model. Each spacecraft model is comprised of several plain-text files such as an activities. mdl file, etc. Activities, resources, and states are defined in .mdl files and then instantiated in .ini files.

Activities

As mentioned above, activities are the central data structure of ASPEN. An activity is a data structure that performs a specific function. Activities are generally defined in the activities.mdl file. The example in Figure 1 includes an instrument data take activity and a solar array drive state change activity. These examples will be used to explain the components of an activity.

An activity is defined in line 1 and 14 of Figure 1. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. These are the only required components of an activity definition. Once the activity is defined, it can be instantiated in the initial state file. Generally, this instantiation will consist of just the activity name followed by the instantiated name and a pair of braces. Many of the components below that are specified as ranges can be fixed to specific values in the activity instantiation.

Parameters are generally used to pass values to subactivities (child activities) or reservations. Lines 2 and 3 contain parameters defined in the parameters.mdl file. In this case, they are constants that represent state names, Parameters can also be passed into activities from higher level activities (parent activities). Line 15 contains an example of a parameter that is passed into an activity. The parameter sad_mode is an enumerated type variable that contains the list of states for a solar array drive. Any one of the states can be passed into the SAD_changer activity when called from a parent activity.

The duration of an activity is given as a range [x, y], a list (a, b, c, d] or a constant. Line 4 defines the duration as a range of 1-60 seconds. The time scale of the spacecraft mission planning can also be specified. All ranges within ASPEN can be specified from zero to *infinity*. If a range is given for the duration, ASPEN will have more flexibility in considering different schedules. This can result in better optimized schedules,

Constraints are temporal constraints a child activity must satisfy with respect to the parent activity in which they are defined. When a constraint is defined, an instantiation of a child activity is created. There are six types of constraints: starts_before, starts_after, ends_ before, ends_ after, contains both of, and contained by. The first four constraint types include a time range and a temporal relationship to the start of or end of the activity in question. For example, on line 6 in Figure 1, the ALI_data_take activity must start after the end of the SAD_changer activity by 100-300 seconds. This constraint tells the scheduler that the SAD_changer activity must be completed at least 100 seconds before the ALI_data_take activity starts. Using the same method, the start or end time of any child activity can be" specified relative to the start or end time of the parent activity. If the time duration is specified as [0,0], the start or end times will coincide exactly.

The [contains both of] constraint is used for child activities that fall within the parent activity. This constraint definition combines a starts_before start of and an ends_ after end of pair of constraints. For example, line 8 defines a constraint for child activity SAD_user that is contained within parent activity ALI_data_take. The first two and last two numbers in the constraint represent ranges of time which separate the start times between the two activities and the end times between the two activities. SAD_user must start exactly four seconds (4,4) after the start of ALI_data_take but the end time can coincide with the end time of ALI_data_take or up to 10 seconds (O, 10) earlier. This relationship is graphically represented in Figure 2.

The [contained_by both of] constraint is used for child activities that are the same size or larger than the parent activity. For example, line 9 defines a constraint for activity SAD_user that starts exactly 300 seconds before the start of and ends 1-16 seconds after the end of activity ALI_data_take.

Subactivities are child activities that can be scheduled any time within the parent activity subject to resource constraints within the subactivity. Subactivities are similar to the constraint-defined activities above without the exact temporal relationship between the parent and child activities. For example, line 10 defines subactivities ALI_user_data and ALI_dark_count. These activities must fall within the temporal range of the parent activity ALI_data_take.

Reservations are used to reserve a portion of a resource or state for the duration of the activity. There are two types of resource reservations: atomic and non-atomic. Line 11 of Figure 1 contains examples of an atomic reservation (processor) and a non-atomic reservation (array_power). The processor reservation reserves the processor for the duration of the activity. No other activities can use the processor during this time. The array_power reservation uses 80 units of array_power for the duration of the activity. If the array_power were a depletable resource, the 80 units would be reserved from the start of the activity until the end of the planning horizon.

State reservations either change the state of a state variable or reserve a state for the duration of an activity. Line 16 of Figure 1 changes the state of

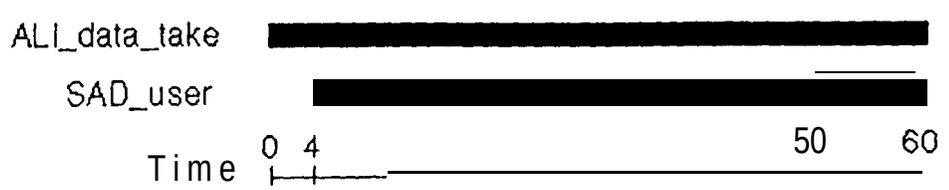

Figure 2 Constraint Relationship: contains both of

the SAD state variable to the value of parameter sad 1. Line 17 of Figure I reserves the "open" state of the aperture state variable for the duration of the activity. If the aperture state variable was in a state other than "open" prior to this activity, the state is changed to "open" by the reservation.

Resources

Resources are items that can be scheduled. There are four types of resources: atomic, concurrency, depletable, and non-depletable. Atomic resources are physical devices that can only be used (reserved) by one activity at a time, Examples of atomic resources include: science instrument, star tracker, reaction wheel, or CPU. Concurrency resources are similar to atomic except they must be made available to the activity before they are reserved. An example would be a telecommunications downlink pass. The telecommunications station would have to be made available before the spacecraft could initiate a downlink. Non-depletable resources are resources that can used by more than one activity at a time and do not need to be replenished. Each activity can use a different quantity of the resource. Examples include solar array power and 1773 bus. Depletable resources are similar to non-depletable except that their capacity is diminished after use. In some cases their capacity can be replenished (battery energy, memory capacity) and in other cases it cannot (fuel). A summary of the four types of resources is presented in Table 1.

Resources are contained in the resources.mdl file. The four types of resources are defined in lines 1, 6, 12, and 18 of Figure 3. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. The type is one of: atomic, concurrency, depletable and non-depletable. The name and type are the only required components of a resource definition. Once the resource is defined, it can be instantiated in the initial state file. Generally, this instantiation will consist of just the resource name followed by the instantiated name and a pair of braces. Note: concurrency resources are not yet implemented.

The capacity of a resource can be specified as a constant, list or range. A range would be used if several similar resources with specific capacities were defined when the resources were instantiated.

Resource Type	Properties
Atomic	Always available when not in use, only 1 user at a time Ex: science instrument, star tracker, reaction wheel, cpu
Concurrency	Only available when made available, only 1 user at a time Ex: telecommunications downlink pass
Non-depletable	Always available when not in use, many users can use different quantities Ex: solar array power and 1773 bus
Depletable	Capacity is diminished after use, may or may not be replenished by another activity Ex: battery energy, memory capacity, fuel

Table 1 Resource Types

```
1   Resource ALI {
2     Type = atomic;
3     Capacity = 1;
4   };
5
6   Resource Solar_array (
7     Type = non_depletable;
8     Capacity = 600; // watts
9     Min_capacity = O;
10  };
11
12  Resource warp_storage {
13    Type = depletable;
14    Capacity = 40000; // Mbits
15    Min_capacity = O;
16  );
17
18  Resource Propellant (
19    Type = depletable;
20    Capacity = 15; // 15 kg
21    Min_capacity = O;
22  );
```

Figure 3 Resource Examples

An atomic resource has a unit capacity and does not have to be explicitly set such as on line 3 of Figure 3. Depletable and non-depletable resources definitions can contain a minimum capacity such as in lines 9, 15, and 21 of Figure 3.

```
1    State_variable ALI_sv (
2            states = ( “data”, “standby”, “idle”, "off" );
3            transitions = ( “standby”-> >"data", “data”-> "standby", “idle’’-> ’’standby",
                                  "standb y"->"idle","off "->"idle", " idlei->ioff");
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 4 State Variable Examples

States

A device, subsystem, or system may be represented by a state variable that gives information about its current operation. The state variable contains the current state, which is defined as an enumerated type. Some examples of possible states are: on, off, open, closed, record, playback, standby or idle. States can be reserved or changed by activities. A state variable must equal some state at every time. At the beginning of a planning horizon, this state is just the default state. Figure 4 contains two examples of state variable definitions, State variables are defined in the state-variables. mdl file.

A state variable is defined in lines 1 and 7 of Figure 4. The definition includes the name followed by a pair of braces and a semi-colon similar to the C language syntax. Lines 2 and 8 contain a list of the states the state variable can contain. The default state must be defined and must be one of the states in the list. Once the state variable is defined, it can be instantiated in the initial state file.

The allowable state transitions between states can be indicated using the transitions keyword with a forward (->) arrow, a hi-directional arrow (<->), or with the “all” keyword (e.g., all<->all).

Parameters

The ASPEN modeling language includes parameters, which are variables or constants. Parameters can consist of integers, strings, floats, or lists. Parameters can be defined as enumerated types for a list of states in a state variable. Ranges of values can also be used. Some examples are:

- parameter string ALI_mode { domain = (“data”, “idle”, “standby”, “off ‘););
- parameter int warprange (domain = [1,40960];);

In the first example, the ALI_mode parameter can take on any of the four values in the list. In the second example, the warprange parameter can be any integer in the indicated range from 1. to 40960.

EO-1 MODEL

EO- 1 is an Earth imaging satellite that is part of the New Millennium Program of technology validation missions. The NASA Goddard Spare 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 main activity in EO- 1 operations is the Advanced Land Imager (AU) 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 in 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 limited amount of downlink time available, only four data takes per day can be

taken. 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 (he 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 activity decompositions. The reason we chose to model these activities as constraints is to ensure they would move with the parent activity if the parent activity were moved. The data take activity breaks down into 14 separate activities as listed in Figure 5.

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 Figure 6.

ALI **Scene Collection**	
ALI_data_take	aperture_changer
ALI_user_data	engdata_user
ALI_user_standby	engdata_changer
ALI_changer	ACS_user
SAD_user	ACS_changer
SAD_changer	cloud_cover_changer
aperture_user	sun angle changer

Figure 5 EO-1 Science Activities

EO- I communication activities are modeled as follows:

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

A keyword "command" 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 Figure 7.

ALI Calibrations
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

Figur 6 EO-1 Calibration Activities

Resources	
Non-Depletable	**Depletable**
ALI	Battery
S_band_Receiver	Warp_storage
Transponders	Propellant
solar_array	
ACDSE	
Warp	
Processor	
Bus_1773	
Cat_bed_heater	
WFF	
DSN	

Figure 7 EO-1 Resources

The EO-1 ASPEN model has ten different state variables which are listed in Figure 8. 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.

State Variables	
Variable	***Possible States***
ALI_sv	data, standby, idle, off
SAD_sv	off, tracking, fixed
aperture_sv	open, closed
aperture_test_sv	small, reed, 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, reed, med_high, high, none
Sun_Angle_sv	low, reed, high, none
WFF_inview_sv	in, out

Figure 8 EO-1 State Variables

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 would 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.

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

The on-board planning system would require uploads 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.

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.

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.

CONCLUSIONS

Modeling EO-1 mission operations in ASPEN is easy and compact. The entire EO-1 model consisting of the activities, parameters, reservations, resources, and state variables as described above, is represented in approximately 700 lines (in plain text files) which also includes comments and headers. The simplicity of the modeling language will allow the operations personnel to easily change the model when needed. The changes will not require a recompile of the code.

We have successfully modeled EO-1 mission operation activities with ASPEN. We have created a model which encapsulates information about: data takes, calibration activities, maneuvers, uplinks, downlinks, validation activities, cloud cover and sun angle states, and initialization activities of power, propellant, and data storage. Using this model with ASPEN will enable EO-1 to function with a very small operations team.

REFERENCES

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

S. Chien, D. Decoste, R. Doyle, and P. Stolorz 1997. Making an Impact: Artificial Intelligence at the Jet Propulsion Laboratory. *AI Magazine,* 18(1):103-122.

A. Fukunaga, G. Rabideau, S. Chien, and D. Yan 1997. ASPEN: A Framework for Automated Planning and Scheduling of Spacecraft Control and Operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space (i-SAIRAS),* Tokyo, Japan.

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

D. Speer, P. Hestness, M. Perry, and B. Stabnow, "The New Millennium Program EO-1 Mission and Spacecraft Design Concept," In Proceedings of the IEEE Aerospace Conference, v. 4, pp. 207-227, Snowmass, CO, 1997.

M. Zweben and M. Fox 1994. *Intelligent Scheduling.* Morgan Kaufmann.