ITERATIVE REPAIR PLANNING FOR SPACECRAFT OPERATIONS USING THE ASPEN SYSTEM

G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660, Pasadena, CA 91109-8099
phone: +1 818 393-5364, fax: +1 818 393-5244, email: {firstname.lastname}@jpl.nasa.gov

ABSTRACT

This paper describes the Automated Scheduling and Planning Environment (ASPEN). ASPEN encodes complex spacecraft knowledge of operability constraints, flight rules, spacecraft hardware, science experiments and operations procedures to allow for automated generation of low level spacecraft sequences. Using a technique called *iterative repair*, ASPEN classifies constraint violations (i.e., *conflicts*) and attempts to repair each by performing a planning or scheduling operation. It must reason about which conflict to resolve first and what repair method to try for the given conflict. ASPEN is currently being utilized in the development of automated planner/scheduler systems for several spacecraft, including the UFO-1 naval communications satellite and the Citizen Explorer (CX1) satellite, as well as for planetary rover operations and antenna ground systems automation. This paper focuses on the algorithm and search strategies employed by ASPEN to resolve spacecraft operations constraints, as well as the data structures for representing these constraints.

1. INTRODUCTION

Planning and scheduling technology offers considerable promise in automating spacecraft operations. Planning and scheduling spacecraft operations involves generating a sequence of low-level spacecraft commands from a set of high-level science and engineering goals. We discuss ASPEN and its use of an *iterative repair* algorithm for planning and scheduling as well as for replanning and rescheduling.

ASPEN is a reconfigurable planning and scheduling software framework [Fukunaga, et al., 1997]. Spacecraft knowledge is encoded in ASPEN under seven core classes: activities, parameters, parameter dependencies, temporal constraints, reservations, resources and state variables. An activity is an occurrence over a time interval that in some way affects the spacecraft. It can represent anything from a high-level goal or request to a low-level event or command. Activities are the central structures in ASPEN, and also the most complicated. A more detailed definition is given in a later section. Together, these constructs can be used to define spacecraft components, procedures, rules and constraints in order to allow manual or automatic generation of valid sequences of activities, also called *plans* or *schedules*.

Once the types of activities are defined, specific instances can be created from the types. Multiple activity instances created from the same type might have different parameter values, including the start time. Many camera imaging activities, for example, can be created from the same type but with different image targets and at different start times. The sequence of activity instances is what defines the plan or schedule.

The job of a planner/scheduler, whether manual or automated, is to accept high-level goals and generate a set of low-level activities that satisfy the goals and do not violate any of the spacecraft flight rules or constraints. ASPEN provides a Graphical User Interface (GUI) for manual generation and/or manipulation of activity sequences. However, the automated planner/scheduler will be the focus of the remainder of this paper.

In ASPEN, the main algorithm for automated planning and scheduling is based on a technique called *iterative repair* [Zweben, et al., 1994]. In iterative repair, the conflicts in the plan are detected and addressed one at a time until no conflicts exist, or a user-defined time limit has been exceeded. A conflict is a violation of a reservation, parameter dependency or temporal constraint. Conflicts can be repaired by means of several predefined methods. The repair methods are: moving an activity, adding a new instance of an activity, deleting an activity, detailing an activity, abstracting an activity, making a reservation of an activity, canceling a reservation, connecting a temporal constraint, disconnecting a constraint, and changing a parameter value. The repair algorithm may use any of these methods in an attempt to resolve a conflict. How the algorithm works is largely dependent on the type of conflict being resolved.

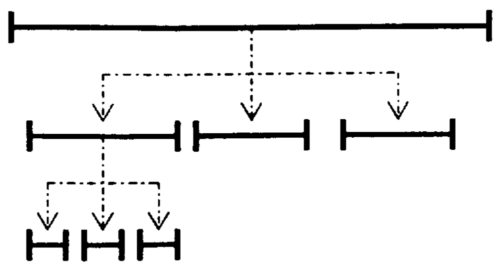

Figure 3: An activity hierarchy.

An *activity* has a set of parameters, parameter dependencies, temporal constraints, reservations and decompositions. All activities have at least three parameters: a start time, an end time and a duration. There is also at least one parameter dependency, relating these three parameters. In addition, all activities have at least one temporal constraint that prevents the activity from occurring outside of the planning horizon. Any additional components are optional.

3. CONFLICTS

A complete plan may not always be consistent with the constraints in the model. A conflict is a violation of one of the model constraints. There are nine basic types of conflicts in ASPEN:

- Abstract activity conflicts
- Parameter dependency conflicts
- Unassigned temporal constraint conflicts
- Violated temporal constraint conflicts
- Unassigned reservation conflicts
- Depletable resource conflicts
- Non-depletable resource conflicts
- State usage conflicts
- State transition conflicts.

Each conflict provides information about what objects are involved and how to repair the conflict.

An *abstract activity conflict* is simply an activity that has not yet been decomposed into its sub-activities. All activities must be expanded to their most detailed level. If an activity has more than one decomposition, the planning algorithm must decide which decomposition to use when detailing the activity. Detailing an activity involves creating instances of the activities specified in the decomposition. In addition, all temporal constraints and parameter dependencies must be connected among the new sub-activities and the parent activity.

A *parameter dependency conflict* is a violation of a functional relationship between two parameters. In other words, the value of a parameter is not equal to the result of a function that constrains that parameter value. For example, a parameter p may be required to be the square of another parameter q. If q is assigned to 5 and p is assigned any value other than 25, this will be a parameter dependency conflict. This conflict can be resolved by assigning a different value to either p or q.

An *unassigned temporal constraint conflict* occurs when a temporal constraint exists for an activity, but an activity instance has not been selected to satisfy the constraint (see Figure 4). A temporal constraint is defined in one activity type A and specifies the requirement for another activity B within some temporal relationship. When an instance of A is created, the temporal constraint is created and is not initially assigned an instance of B. The conflict computes all activity instances that can repair this conflict (basically, all instances of type B).

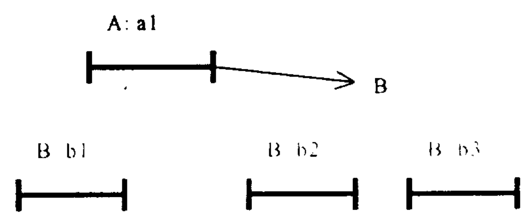

Figure 4: An unassigned temporal constraint conflict requiring an activity of type B. Any of b1, b2 or b3 can be use, or a new instance of type B can be added.

A *violated temporal constraint conflict* occurs when a temporal constraint has been assigned, but the relationship (specified in the model) does not hold for the two participating activities (see Figure 5). For example, consider an activity instance A that must end before the start of activity instance B by at least 10 seconds but at most 1 minute. If A ends at time t, then there is a conflict if B does not start between time t+10 and t+60. The conflict keeps track of the contributing activities, which in this example includes activities A and B. In addition, the conflict computes the start time intervals for moving an activity that would repair the conflict. Continuing with

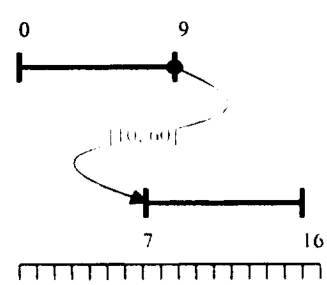

Figure 5: A violated temporal constraint conflict.

conflicts, the contributor is only the activity that changes the state (i.e., makes the illegal transition). Again, the changer must be moved to a time later than the state in conflict or earlier than the previous state. As with resource conflicts, new activities can be created to repair state variable conflicts. For a state usage conflict, we can add activities that can change to the desired state. These activities must be added at a time before the conflicting user, but after the conflicting changer. For state transition conflicts, we can add activities that can change to a state that makes a legal transition. These activities must be added between the two conflicting changers.

4. ITERATIVE REPAIR SEARCH

ASPEN organizes its search around several types of constraints that must hold over valid plans. ASPEN then has organized around each constraint type, a classification of the ways in which the constraint may be violated. These violations are called conflicts. Organized around each conflict type, there is a set of repair methods. The search space consists of all possible repair methods applied to all possible conflicts in all possible orders. We describe one tractable approach to searching this space.

The iterative repair algorithm searches the space of possible schedules in ASPEN by making decisions at certain choice points, and modifying the schedule based on these decisions. The choice points are:

- Selecting a conflict
- Selecting a repair method
- Selecting an activity for the chosen repair method
- Selecting a start time for the chosen activity
- Selecting a duration for the chosen activity
- Selecting timelines for reservations
- Selecting a decomposition for detailing
- Selecting parameters to change
- Selecting values for parameters

Given a schedule with a set of conflicts of all types, the first step in the iterative repair algorithm is to select one of the conflicts to be attacked. Next, a method is selected for repairing the conflict. All possible repair methods are:

- Moving an existing activity to a new location
- Creating a new activity and insert at a location
- Deleting an existing activity
- Connecting a temporal constraint between two activities
- Disconnecting a temporal constraint between two activities
- Detailing an activity
- Abstracting an activity
- Making reservations of an activity
- Canceling reservations of an activity
- Grounding a parameter in an activity
- Applying a dependency function between two parameters

As described in the previous section, the type of conflict will determine the set of possible repair methods for any given conflict. If it was decided to try to move or delete an activity, the algorithm must decide which activity to move or delete. The type of conflict and the location of the conflict will determine the set of possible activities that, if moved or deleted, may resolve the conflict. In addition, a new start time and duration must be assigned to the activity. If it was decided to try to add a new activity, the activity type must be chosen from the list of possible types determined by the conflict. For abstract activity conflicts, the repair algorithm will most likely choose to detail the activity. If it has multiple decompositions, one of them must be chosen. Deciding to abstract an activity requires choosing which activity to abstract. When making a reservation in an attempt to resolve a conflict, a resource or state variable must be chosen for the set of possible resources or state variables. Also, if the reservation has an unspecified value, one must be chosen for it. Canceling reservations only requires choosing which reservation to cancel. If the repair algorithm has decided to connect a temporal constraint, the specific activity for the constraint must be selected. When disconnecting, only the constraint to be disconnected must be chosen. Finally, changing a parameter value requires choosing a new value for the parameter. After all decisions are made and the repair method is performed, the effects are propagated and the new conflicts are computed. This process repeats until no conflicts exist or a time limit has been exceeded.

5. SEARCH HEURISTICS

All throughout the iterative repair algorithm, many decisions must be made. In other words, there are many ways in which a conflict may be resolved. Some ways ultimately work better than others do. For example, deleting an activity may resolve a resource conflict caused by that activity. However, that activity may have been required by other activities. Or, if the activity was a high-level goal, the user might prefer to have as many goals satisfied as possible. Another typical example involves choosing a location to move an activity. Many locations may resolve the conflict being addressed, but many locations may also create addition conflicts. In order to guide the search toward more fruitful decisions, the user can define a set of search heuristics.

In ASPEN, a heuristic is a function that orders and prunes a list of choices for a particular decision in the search. Heuristics can be defined at each of the choice points in the algorithm. For example, one heuristic might sort the

Smith, B., Sherwood, R., Govindjee, A., Yan, D., Rabideau, G., Chien, S., Fukunaga, A., "Representing Spacecraft Mission Planning Knowledge in ASPEN," *Artificial Intelligence Planning Systems Workshop on Knowledge Acquisition*, Pittsburgh, PA, 1998.