Automated Planning for Interferometer Configuration and Control[1]

Gregg Rabideau, Leonard Reder, Steve Chien, Andrew Booth
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
818-393-5364

constraints between commands. For example, the star tracker must lock the target star before the fringe tracker can lock the interference fringe. Automated planners can plan and schedule these observation activities by constructing lower level command sequences to implement the higher level science goals (e.g., the observations). The planner can also replan in the (likely) event of run-time anomalies – rapidly responding to changing events to regain tracking on a science target or ensure robust operations in the presence of less reliable hardware.

Each command generated by the planner may require additional low-level sequencing. In particular, many closed-loop control tasks must be highly optimized. Event-driven sequencing is more suited for problems with tighter real-time requirements. For problems with fixed state sets and well-defined state transitions, sequential operations can be programmed and executed based on state transition diagrams. For example, a single mirror alignment command may actually start a sequence of commands that repeatedly computes the centroid of a reflected light beam and moves the mirror to change the centroid location. Event-driven sequencers can move through state transitions until the desired state (e.g., “aligned”) is met.

In this paper, we focus on using automated planning and scheduling technology for high-level interferometer configuration and coordination. We use the ASPEN system developed at JPL to demonstrate how planning can be used to perform many operations tasks with many benefits over

compared with the stars they orbit, the center of light

4. THE ASPEN PLANNER

We show that AI planning and scheduling technology can be used to solve the interferometer configuration and control problem. ASPEN (Automated Scheduling and Planning ENvironment) encodes system operability constraints, rules, hardware models, science experiment goals, and operations procedures to allow for automated generation of low-level sequences [4]. By automating the command sequence generation process and by encapsulating the operations specific knowledge, ASPEN enables complex systems to be controlled by a small operations team–thereby reducing costs.

In ASPEN, the main algorithm for automated planning and

since things rarely go exactly as expected, the planner stands ready to continually modify the plan.

6. Plan Optimization

ASPEN also has facilities for representing and reasoning

"`mirror_alignment`" activities for aligning the mirrors when needed. Activities of these types can be scheduled at any time and last for a particular duration.

Finally, we have modeled a set of high-level observation goals for the Keck Interferometer. These goals are eventually detailed into lower level requests for particular resources and states at particular times. At the highest level, we have activities that represent the science experiments (described earlier) for Exozodiacal Dust Measurement, Hot Jupiter Detection, and Astrometric Exoplanet Detection. The Exozodiacal Dust Measurement activity decomposes into interferometry activities that uses the two main telescopes and the nulling combiner. The Hot Jupiter Detection activity also decomposes into interferometry activities using the two main telescopes, but instead has a parameter for indicating which combiner to use. Finally, the Astrometric Exoplanet Detection activity decomposes into astrometry activities that use two of the four outrigger telescopes. Each of the goal activities has parameters for specifying the sky target at which to perform the experiment. Moreover, each goal places requirements on the pointing, mirror alignment, and mirror health of the relevant telescopes for the duration of the experiment.

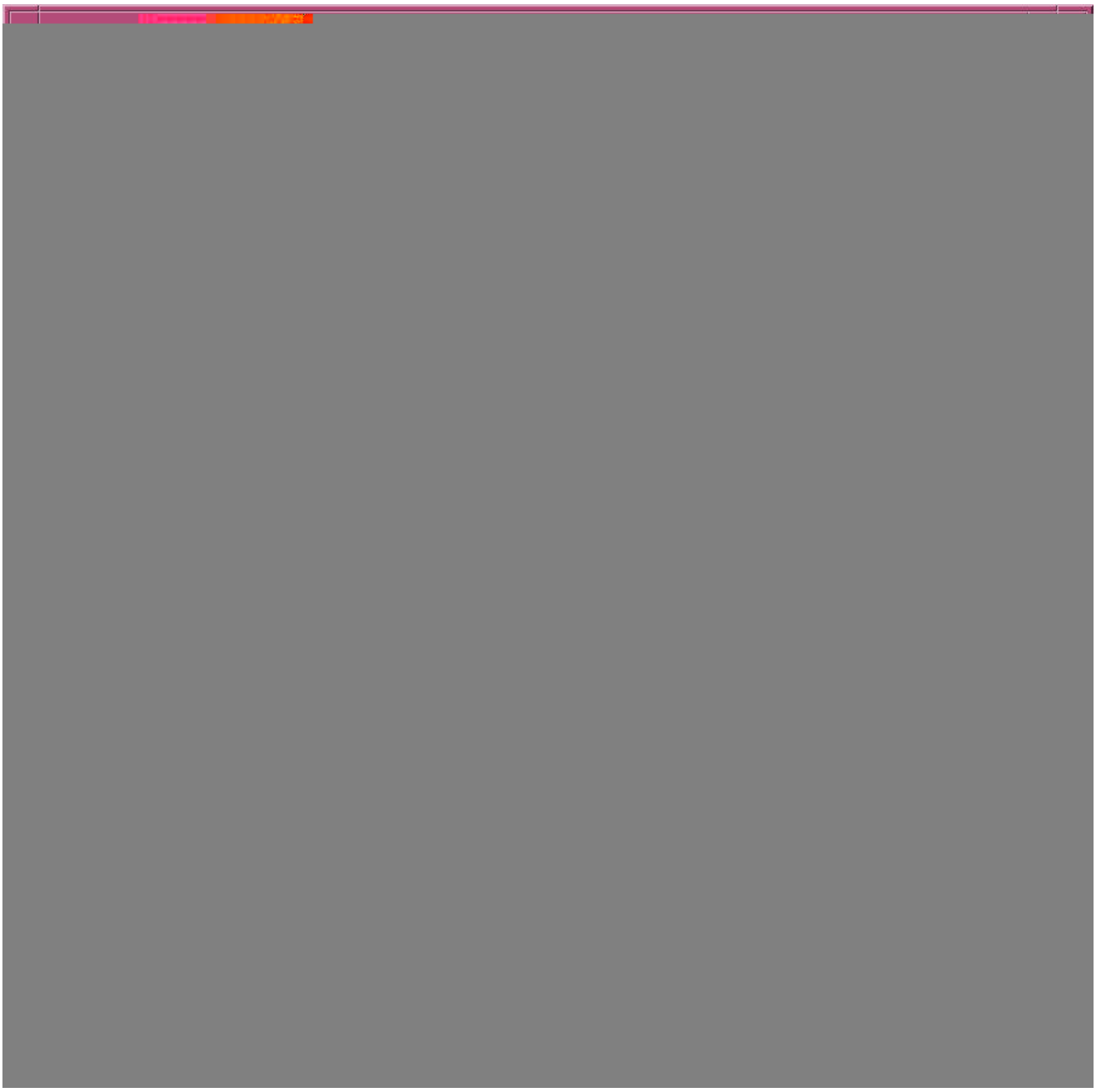

Figure 4 – ASPEN GUI for a hypothetical Keck 3-night observation schedule.

Once all of the model components are specified, we are now ready to generate observation plans for the Keck Interferometer (see Figure 4). Whether generated manually or automatically, all model constraints will be monitored and conflicts will be flagged. For example, we have automatically generated a three-night plan from an initial request of nine science experiments (three per night). After making the goal requests, conflicts exist from the various goal requirements for pointing, etc. Using the iterative repair algorithm, conflicts were resolved until none remained. For example, adding target search activities satisfied the pointing requirement but violated the mirror alignment requirement. The repair algorithm would then add mirror alignment activities to re-align the mirrors after the target search but before the experiment. The algorithm continued this way, finishing with a valid plan of 220 activities.

KI Preference Model

While the repaired plan satisfied all of the hard constraints, it would not be considered a high quality plan. In order to define plan quality, and subsequently optimize plans, we need to state a set of preferences for KI operations. First, the most common preference is one that prefers more science experiments. While the original request had nine experiments, it is possible that more could fit in the three-night plan. The user can make optional requests for additional science experiments. The corresponding preference will evaluate to a higher score when the plan contains more of the optional experiments. Other preferences are for smaller start times for activities. These make sure experiments are completed as soon as possible, within the requested start time window. Still more preferences are for fewer state changes on the pointing and alignment state variables. These keep the telescope and mirror movements at a minimum.

Now that we have the preference model defined, we can use the iterative optimization algorithm to improve the plan. Each preference is evaluated to a score between 0 and 1, with 1 being the highest score. These individual scores are then weighed and averaged into an overall score for the plan. The iterative optimization algorithm selects and improves one of the preferences until all have a score of one or a time limit has been exceeded. For example, it will delete unnecessary alignment activities to improve the score for the preference for fewer alignment state changes. It will also move activities to earlier start times (while avoiding conflicts) to improve other preferences.

KI Continuous Planning Interface

Finally, we have developed the required interface between the CASPER continuous planner and a KI simulator. Most of the code for the interface and sim were automatically generated from the ASPEN model. The generated sim is a discrete event simulator that simulates the execution of activities and the evolution of resources and state variables. The generated sim executes activities as expected by the planner. However, the user can augment the sim code to simulate variations at execution time. The generated interface links CASPER to the simulator. It is code that must match a specific template required by CASPER. In particular, it must have a function "`commit(Activity)`" that, given a CASPER activity, translates the activity into a simulator command. Because these commands are time tagged, the simulator will queue the commands and pop them off the queue at the appropriate time. In addition, the state determination code translates the execution status into plan updates when the simulated state deviates from the planned state at any time. Whenever the plan updates result in conflicts, the repair algorithm is automatically invoked to fix them. Figure 5 shows the overall architecture.

To understand this process, we will step through an example of a plan being executed by the simulator. First, we start ASPEN and generate a feasible plan. Then, we start CASPER, indicating the hostname and port number where the ASPEN socket server is running. This will allow CASPER to communicate with ASPEN. Next, we start the simulator and indicate the host and port where CASPER is running. This allows CASPER to communicate with the sim. As the simulator runs, CASPER monitors the current time reported by the sim. When the current time approaches the start time of a planned activity (within a predefined delta), CASPER reads the activity from ASPEN and calls the "`commit`" interface function. For example, one of the first planned activities for Keck is a "`search_for_target`" activity. One minute before the activity's start time, CASPER calls "`commit`" on this activity to translate it into a "`search_for_target`" simulator command with the target parameter value (e.g., "`t1`"). The simulator pushes this command in its queue. One minute later, the simulator pops it off the queue and begins simulating the execution of this command. For this command, the simulator simply waits

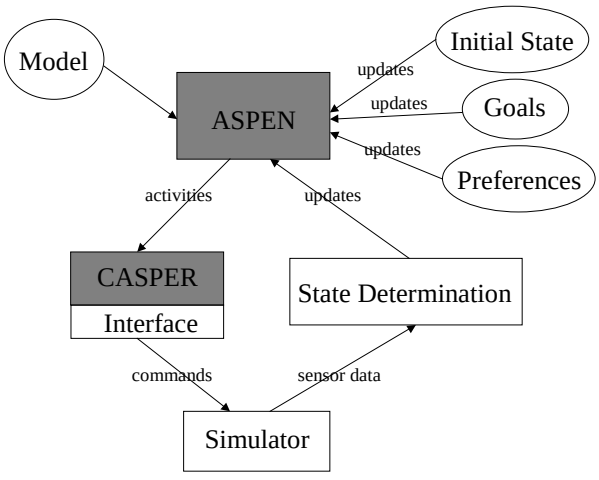

Figure 5 – ASPEN/CASPER architecture. Ovals represent inputs required for each application. For running CASPER, a baseline for the unshaded modules (rectangles) can be generated from the model.

[8] M. Zweben, B. Daun, E. Davis, and M. Deale, “Scheduling and Rescheduling with Iterative Repair,” in Intelligent Scheduling, Morgan Kaufman, San Francisco, 1994.

[9] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee, "Iterative Repair Planning for Spacecraft Operations in the ASPEN System," International Symposium on Artificial Intelligence Robotics and Automation in Space, Noordwijk, The Netherlands, June 1999.

[10] A. Jonsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith, "Planning in Interplanetary Space: Theory and Practice," Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, April 2000.

[11] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve