and machines, on-board autonomy, human control, and ground automation.

CHASER is comprised of three coaligned instruments that take data in the far and extreme ultraviolet wave-lengths. The first and oldest of these instruments (17 years old) is FARUS, which takes a continuous spectrum from 115 nm to 190 nm with a resolution of .12 nm. LASIT takes images of the full solar disk of the sun in the Lyman-alpha wavelength (121.6 nm) with a Charge Injected Device imager. The final instrument in the scientific package, SXEE, consists of four photometers, each having a different metallic coating so as to enable them to look at different wavelengths between 1 and 40 nm. The objective of these instruments is to measure the full disk solar ultraviolet irradiance and obtain images of the sun in the Lyman-alpha wavelength, providing a correlation between solar activity and radiation flux as well as an association of Lyman-alpha fluxes with individual active regions of the sun.

The flight segment of the DATA-CHASER project consists of a canister that is equipped with a Hitchhiker Motorized Door Assembly (HMDA), which houses the instruments and their support electronics. The second canister contains the flight computer for the payload as well as the 2 GB Digital Audio Tape (DAT) drive that is used to store all data that is collected during the mission. The payload data is also sent to the ground system through both low rate (available 90% of the time, at 1200 bps) and medium rate (available when scheduled, at 200 kbps). The payload is also capable of receiving commands sent from the ground system when uplink is available.

During the mission, the DATA-CHASER payload will be operating in four different modes. Most of the time, when DATA-CHASER is powered, it will be in a passive mode where it is monitoring its state and notifying the ground of any changes. During the time in the mission when the orbiter is scheduled to point the bay at the sun, the DATA-CHASER payload will shift into solar active mode where all instruments take data.

The data is both written to the DAT drive on board and downlinked to the ground system for immediate data analysis. Several times during the mission, DATA-CHASER will take data while not pointing at the sun. This data is used for testing various portions of the DATA experiment with nonsolar pointing data in addition to being used for instrument calibration.

One of the consequences of flying on the shuttle system is that shuttle resources are limited, and their availability is subject to change every 12 hours. These resources include access to uplink and downlink channels, and time that your payload is allowed to operate. In addition to these resources, any given payload may also have environmental constraints as to how much contamination the payload can take. Another example is thermal constraints, such as maximum solar point time.

STS-85, the flight that DATA-CHASER payload is scheduled to fly on, is one of the most complicated flights that the shuttle has flown to date. In addition to the DATA-CHASER payload, there are four other payloads sharing the same HH bridge. In addition to the IEH-2 bridge, there is another HH bridge, a pallet payload, and a Spartan deployable satellite. Needless to say the shuttle pointing requirements are considerably tight.

In addition to modeling what the internal constraints and resources of the payload are, DCAPS must also search the shuttle flight plan for times when we are allowed to operate, downlink our data, uplink new command sets, and when we have to protect the scientific instruments from contamination events.

DATA-CHASER is an interesting scenario for scheduling because of the complex data and power management involved in the science gathering. An automated scheduler must find an optimal “data taking” schedule, while adhering to the resource constraints. In addition, the scientists would like to perform dynamic scheduling during the mission. As an example, the summary data may indicate the presence of a solar flare. If this occurs, scientists have different requirements and goals, such as higher priorities on certain instruments or longer integration times. These new goals may require a different schedule of activities.

3. USER OPERATION

The DATA-CHASER Automated Planner / Scheduler will be part of the DATA-CHASER mission operations software. It will be a ground-based intelligent tool used for generating scheduled commands for uplink to the payload. The user's manual [3] can be found at the Jet Propulsion Laboratory. There are three phases of operating the DCAPS system: a goal satisfaction phase, an interactive repair phase, and an optimization phase.

Goal Satisfaction

The first phase of scheduling in DCAPS involves generating an initial schedule from a set of high-level, user-defined goals. The scientist or engineer simply requests one or more of the predefined goals, and the scheduler will generate the low-level activities that satisfy the goals. For example, the scientist can simply make a request for solar observations during all solar viewing periods. Given this goal, the scheduler will create and position the instrument data-take activities and their supporting activities.

Goal satisfaction is a way of generating an initial schedule. Goals are parameterized, and create activities in positions relative to certain schedule events or parameters. In this way, the same goals can be requested for different initial states. This makes them more flexible than alternate ways of creating an initial schedule, such as simply loading activities from a file. For example, the initial state in DATA-CHASER contains shuttle maneuver activities. These activities determine the solar viewing periods of the payload. The solar observation goals are based relative to these solar views, and therefore, are applicable to any maneuver sequence.

Interactive Repair

In the second phase of scheduling, the user has the opportunity for interacting with the schedule at a more detailed level. The scientist or engineer can view the activities at several levels of abstraction. The graphical user interface (GUI) can display activities from the highest level, as a single event, down to the lowest level, showing the detailed steps that make-up the activity.

The user can also modify the schedule by moving, adding, or deleting activities, as well as changing activity parameters. For example, the scientist might want to delete a LASIT data-take and replace it with a FARUS or SXEE data-take. Or, perhaps he/she may simply want to change the target of some data-take, from a solar scan to an earth scan.

Although the user has the capability of making these types of adjustments, he/she does not need to worry about the various interactions, constraints, or resource usage of the activities being modified. This information is monitored by DCAPS, and changes are propagated to the dependent objects. The results of the modification, including any conflicts, are displayed by the GUI. In addition, when the user introduces scheduling conflicts, DCAPS can resolve them automatically.

DCAPS can be called upon at any time to resolve any conflicts residing in the schedule. Conflicts are violations of resource capacities or temporal constraints. In this way, the user does not need to be very informed, careful, or specific about his/her requests. For example, a scientist can move a data-take activity without concern for its power usage. Or, a general request for data-takes can be made, without specifying the exact times for the activities to occur. Although these changes or requests may cause one or more conflicts, DCAPS can resolve these conflicts with one simple command.

Optimization

Finally, the third phase of DCAPS operation is schedule optimization. After resolving all conflicts, the schedule may still contain violations of user preferences. These preference violations can be repaired in a manner similar to repairing regular conflicts. The main difference is that the modeler must explicitly represent the types of violations and general mechanisms for repairing them. As an example, considered an engineer's desire to have all data written to permanent storage at the end of the mission. Having data in the RAM at the end of the schedule is a violation of a preference, rather than a violation of a resource.

Preferences can also be expressed in a schedule evaluation function. In the optimization phase, DCAPS can score valid schedules based on the evaluation function developed by the modeler. One simple evaluation function may give higher scores to schedules with more science observations. Due to the fact that DCAPS utilizes a stochastic scheduling procedure, more optimal schedules can be found by simply running the scheduler many times and retaining the schedule with highest score.

4. Model Representation

In order to use either Plan-IT II standalone or the full DCAPS system, the user must write a software model of the mission activities and spacecraft resources. This process involves defining a set of objects and how they interact. These definitions are then used by the scheduler to create instances of the objects.

Model Objects

The basic objects in the PI2 sequencing tool are activities, resources, slots, and dependencies.

Activities—Activities are used to model the events that happen that affect the DATA-CHASER payload, and the actions that the DATA-CHASER payload can take. All activities have some basic components: a duration, a list of slots, and a list of slot-value assignments. In addition, certain types (described below) have a list of subactivities. For these activities, the user can also define a set of temporal constraints between the subactivities. Next, we describe in more detail the four basic types of activities: events, steps, step-activities, and activities.

Events are used to model activities that do not occur in a fixed relation to other activities (e.g. Tracking and Data Relay Satellite System (TDRSS) contacts) and are not part of an activity hierarchy.

Steps are the "leaf" nodes in the activity hierarchy tree. In other words, they do not contain any subactivities. Steps cannot be instantiated without their parents and are used to model the activities at the lowest level of detail. For instance, we model an activity called CHASER-heating, which consists of two steps, CHASER-heater-on and CHASER-heater-off.

Step-activities are used to model activities at a middle level of abstraction. They can contain steps, but must also have parent activities. In DCAPS, we model an activity SXEE-Data-Take, which models the SXEE instrument opening its aperture and taking a scan. In this case, there is a step-activity called SXEE-Scan-Step, which has sensor read steps and cannot be instantiated by itself.

Activities are used to model activities at the highest level of abstraction. They are the "root" nodes in the hierarchy tree, containing subactivities, but no parent activity. The activity and event objects are what the scheduler can instantiate, and Plan-It II provides methods to access the varying levels of abstraction.

Resources—Resources define the various physical resources and the constraints they impose. Resources come in essentially five varieties: state, concurrency, depletable, nondepletable, and simple.

State resources are used to model the systems in the DATA-CHASER payload that have states associated with them. For each state resource, the modeler must specify the possible values that the state can be. Most of the systems have at least one state variable, which is whether or not they are activated. The orientation of the payload is also modeled as a state variable.

Concurrency resource constraints are used to model rules that stipulate that an activity either must occur with another activity or cannot occur with another activity. One relationship that is modeled with a concurrency resource is the requirement that a downlink or uplink can only occur during contact with a TDRSS satellite. This is modeled as a resource that is present when there is TDRSS contact activity, and required when there is a downlink or uplink activity.

Depletable resources are used to model resources with a fixed quantity, such a fuel or RAM. Activities can use some finite amount of a depletable resource, which may or may not be restorable. The amount used by the activity is persistent to the end of the schedule. In addition, the modeler must specify a maximum capacity for each depletable resource. In DCAPS, RAM is modeled as a depletable resource. Science observations produce data and use some amount of the depletable resource. Other activities, such as a transfer to permanent storage, may restore this resource.

Non-depletable resources are used to model resources which have a limit to the usage at any one time, but are reset at the end of the activity that consumes the resource. Similar to depletable resources, nondepletables are assigned a maximum capacity. Resources like power are modeled with nondepletable resources.

Simple resources are used to model devices that can only be used by one activity at a time. For instance, each of the instruments on board DATA-CHASER, FARUS, SXEE, and LASIT, are capable of taking only one image at a time and are modeled with simple resources. Simple resources are essentially nondepletable resources with an capacity of one.

Slots—Slots are parameters of activities that allow them to affect resources. They are defined separately but referenced inside activity definitions along with a value assignment for each slot. In the slot definition, the modeler must specify which resource it affects. The main types of slots are: info slots, simple slots, availability slots, choice slots, amount slots, and state slots.

Info slots are for embedding text information in activities. They are merely placeholders and do not have any effect on scheduling.

Simple slots are included in activity type definitions in order to model usage of a simple resource. For instance, there is a slot called FARUS, which is included in activity definitions of activities which use the FARUS instrument. This info slot models the usage of the FARUS instrument.

Availability slots are the slots that allow activities to provide or require the presence of a concurrency resource. There is a slot in DCAPS called TDRSS-coverage and a slot called TDRSS-coverage-needed. Both affect the TDRSS-coverage resource. TDRSS activities have the TDRSS-coverage slot, and downlink activities have the TDRSS-coverage-needed slot. TDRSS activities can be placed anywhere and provide the presence of the resource. Downlinks can only be placed in intervals where TDRSS activities are present, since this activity possesses the slot that requires the TDRSS resource to be present.

Amount slots come in two varieties: amount and reset-amount. Amount slots reduce a depletable or nondepletable resource, and reset-amount slots increase a depletable or nondepletable resource. Amount slots do not have to be associated with a resource, however. In DCAPS, we have an amount slot called Rate, which is how we model the different bit transfer rates in activities that move data, such as a downlinks or DAT reads and writes. To find the amount of data an activity transfers, we multiply the rate by the duration of the activity.

There are also two types of state slots: state-users and state-changers. State-user slots require the presence of a certain state in a state resource, and state-changer slots change the state of a state resource. The modeler must define the set of possible states. In DCAPS, there is a state resource that models the shuttle orientation, which can be solar, earth, lunar, or deep-space. Solar science activities require the shuttle orientation state to be solar, while shuttle maneuver activities change the orientation state.

Dependencies—Plan-It II provides the ability to set up links that allow one object to affect another object. These links are called dependencies. There are several types of dependencies based on the types of objects it relates: slot-to-resource, slot-to-slot, slot-to-activity start or duration, activity start or duration-to-slot, and resource-to-resource dependencies.

Slot-to-resource dependencies are the default dependencies in the Plan-It II system. They allow a slot to affect a resource, and are created automatically when a slot is defined with the same name as a resource.

Slot-to-slot dependencies allow the value of one slot to affect the value of another slot. For instance, in the DAT-transfer activity, there are two slots, one that models the removal of data from the RAM, and one that models the addition of data to the DAT (digital tape). In DCAPS, a dependency has been defined that sets the value of one of the slots equal to the value of the other slot (so that the amount subtracted from RAM is never different than the data added to the DAT).

Activity start time or duration-to-slot dependencies and slot-to-activity start time or duration dependencies facilitate the modeling of convenient relationships among Plan-It II objects. For instance, the DAT-transfer has a slot called Rate, which is the rate at which data can be moved from the RAM to the DAT. We have a dependency that sets the amount of data that is removed from the RAM equal to the rate multiplied by the duration. An example of a dependency that goes from slot to duration is a dependency which links the selected target for a science image to the length of time it takes for the instrument to scan. The duration of a FARUS scan varies depending on its use of the shuttle orientation state (solar, earth, or lunar).

Resource-to-resource dependencies allow one resource to affect another resource directly. This is very convenient for modeling power usage, since power consumption can be tied to activities or states. For instance, power consumption by the heater can be linked to an activity (e.g., the activation of the heater), or to a state of the heater (e.g., when the state of the heater is "on," more power is used).

Hierarchy

The modeler can create an activity hierarchy when defining the activities. All this means is that activities can have subactivities which can also have subactivities, and so on. Only the activity at the top of the hierarchy can be instantiated in the schedule. When an activity is created, all of its subactivities are created automatically. Therefore, the scheduler must schedule the entire hierarchy if it wants to schedule one of the components.

In modeling the DATA-CHASER shuttle payload, decisions had to be made about where to put activities in the activity hierarchy. We decided to model those activities that could be scheduled arbitrarily (and had no subactivities) as events not in a hierarchy. Some activities that were modeled as events were TDRSS contacts, shuttle venting, and very simple activities that could occur independently, like relay activations and HMDA operations (opening and closing).

If one event always occurred in some fixed temporal relationship to another, then we modeled it as an activity in a hierarchy. For instance, a SXEE data-take consists of a number of calibrations, a door opening activity, several scans, a door closing activity, then a data transfer to the RAM buffer. We modeled all of these activities as steps in an activity called SXEE-Data-Take.

Common Strategies

There were a number of strategies that we employed in the modeling process that made modeling the DATA-CHASER payload simpler.

One strategy that we employed was to reduce the number of states that state variables could have through discretization. For instance, spacecraft orientation is best modeled with a real valued 3 dimensional vector. But for modeling purposes, we reduced the number of possible orientations to a discrete set of four: solar, lunar, earth, and deep space.

Another strategy that we employed in modeling DATA-CHASER was to separate one component into several. For instance, there was really only one memory buffer that was used for storing several types of data, but we modeled it as though it were three buffers: one for science data, one for engineering data, and one for storing data to be downlinked. We also did this with power. There are really only two power sources, DATA power and CHASER power, but we modeled them as though there were different power resources for each of the science instruments and several of the other subsystems. This allowed us to track power usage more conveniently.

5. Automated PLANNER/SCHEDULER

The DATA-CHASER Automated Planner / Scheduler produces a complete, valid schedule of payload operation commands from a model, initial state, and set of high-level goals. In addition, it can input intermediate, invalid schedules (resulting from user changes) and produce a similar, but valid schedule. Finally, the scheduler can take several valid schedules, score them, and select the most optimal schedule.

The planner/scheduler consists of two main parts, the Plan-IT II (PI2) sequencing tool [4] and the schedule reasoner (see Figure 2). PI2 was written by William C. Eggemeyer and originally designed as an "expert assistant sequencing tool." PI2 includes a GUI that allows for easy manipulation of the schedule. In addition, it serves as an activity/resource database that supplies valuable information to the schedule reasoner. PI2 supports complex monitoring and reasoning about activities and

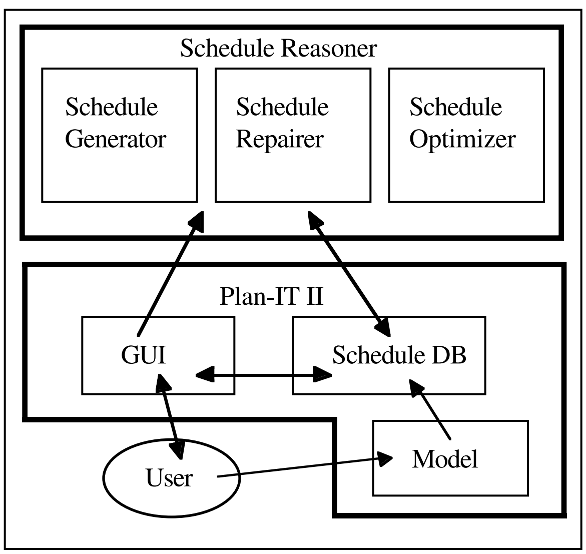

Figure 2: DCAPS architecture

the various constraints between them. The schedule reasoner uses Artificial Intelligence (AI) techniques to automatically generate new schedules, repair existing faulty schedules, and optimize valid schedules. PI2 provides information about resource availability and conflicts; the scheduler must decide which activities to use to resolve the conflicts and where to place the activities temporally. The two components work together to provide easy and fast sequencing of mission activities.

Schedule Data-Base

In the DCAPS system, PI2 is used primarily as a "schedule database" and resource constraint checker. It was originally developed as a graphical sequencing tool. Activities and resources are displayed on a graphical output. An activity represents some mission event that occurs over a period of time and uses some of the mission resources. A resources represents some limited available material whose usage is modeled as discrete blocks over time.

For each type of activity and resource, PI2 displays a timeline, which represents the behavior of that activity/resource type over a period of time. When activities are created, they are placed at a specified time on the timeline. Resources used by that activity are updated to reflect the additional usage. In addition to schedule visualization, PI2 provides an easy-to-use input interface for modifying the schedule. Moving activities is as simple as a click-and-drag with a mouse.

PI2 helps ease the burden on sequencers by continually monitoring all activities in the sequence. As activities are added or moved, the change in resource usage is automatically updated, and the new resource profiles are displayed. With this information available, the user can immediately see the effects of a schedule change on the mission resources. For each resource, PI2 also monitors any conflicts that are occurring on the resource.

Conflicts are time intervals where the limitations of the resource have been exceeded. These conflict intervals are highlighted in red to flag their existence for easy identification. Finally, PI2 monitors any dependencies that have been defined between activities and resources. The values of specific parameters of activities and resources may be functionally dependent on values of other parameters. PI2 automatically keeps these parameter values consistent.

PI2 also helps out by serving as an activity and resource database, producing/accepting information to/from a sequencer. The functional interface to PI2 has been extended to better assist an automated sequencer. A basic set of "fetch" functions have been developed to quickly retrieve information about conflicts and the resources and activities involved in the conflict. For example, an interface function has been written to fetch the legal times where an activity can occur in the schedule. Here, "legal times" refers to positions where no conflicts are caused by any of the resources used by the given activity.

In addition to fetching information about the current state of the schedule, the user will need to be able to change the current state in attempt to fix or optimize the schedule. Some basic primitive functions are provided by PI2 to allow an external system to add and move activities, change their duration, etc. These primitives make up the set of actions that a scheduler can take when trying to resolve conflicts.

Schedule Reasoner

The second major component of DCAPS is the automated schedule reasoner. This is the next step in automating and simplifying the spacecraft command sequencing process. There are three parts to the schedule reasoner: a schedule generator, a schedule repairer, and a schedule optimizer. First, the schedule generator will transform a set of user-defined, high-level goals into a valid sequence of low-level commands. Second, the schedule repairer will automatically restore the consistency of the sequence after arbitrary user interaction by rescheduling using repair actions. The scheduler repairer iteratively attempts to resolve each conflict, which involves making choices on what to repair and how to repair it. Finally, the schedule optimizer can optimize a valid schedule to increase the scientific return.

Schedule Generator—The first step in sequencing spacecraft commands is to come up with an initial schedule of events for each phase of the mission. This process has been partially automated in DCAPS with the schedule generator. Expressing schedules and partial schedules to be generated is done through user defined goals. There are two ways in which user goals are handled in DCAPS. First, initial science and engineering goals are handled with parameterized scheduling functions. Each function implements a goal. For example, there is a "Place-Power" function that schedules power switching activities in appropriate places based on some engineering parameters. Parameters may include such things as a minimum time between switching, or a power on during a particular state of a different resource.

Second, science goals can also be expressed through data-take requests, which do not have to be a part of the initial schedule generation. For example, a scientist can request ten additional scans from a particular instrument to occur any time during some phase of the mission. This type of general request does not include specific locations or necessary supporting activities. The scheduler will simply place them at random positions and allow any conflicts to be resolved by the automated repairer.

Schedule Repairer—The generated initial schedule may still violate some of the spacecraft constraints. Also, the scientists and engineers might feel that their goals were not completely satisfied, and may need to interact with and modify the generated schedule. By modifying the schedule, new conflicts may be introduced. Therefore, we need some way of automatically resolving any existing conflicts in the schedule, while disrupting the current state of the schedule as little as possible. Having the process automated allows the user to be less careful, and therefore spend less time on the details of sequencing the activities. When general requests or changes have been made, all conflicts can be resolved by executing one simple command to invoke the schedule repairer.

Before describing the schedule repairer, we must present a few definitions. A "hard conflict," or just "conflict," is a violation of one of the resource constraints. A conflict occurs over a certain time period and is caused by activities called "culprits." For example, if the power capacity is exceeded from time t_1 to time t_2, then a conflict exists from time t_1 to time t_2, and the culprits are any activities that use power during this time (see Figure 3). A "soft conflict" is a violation of one of the user's high level goals. "Hard conflicts" are violations of

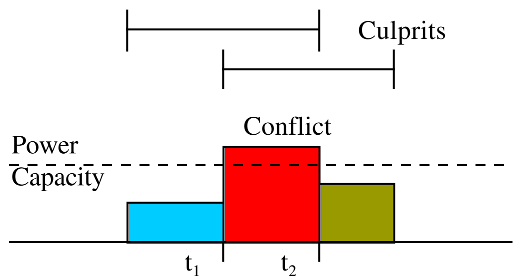

Figure 3: Conflicts

legal constraints, while "soft conflicts" are violations of user preferences. "Choice points" are places in the scheduling algorithm when a decision must be made. For example, when there are many conflicts to resolve, the scheduler must decide which conflict to resolve first. A "hard choice," or just "choice," is a decision made solely on the basis of possible hard conflicts. It may be decided, for example, not to place an activity at a certain time because new conflicts will be added as a result of that placement. A "soft choice" is a decision made on the basis of user preferences or heuristics with the hopes of generating a more optimal schedule. An example of a user preference is a priority scheme on certain activities. One heuristic may be to move lowest-priority culprits to the nearest legal position.

There are three possible actions to take in attempt to resolve a conflict: move, add, or delete an activity. The "move" action involves moving one of the culprits of the conflict to a positions that will either resolve the conflict or at least ensure that the moved activity is no longer a culprit. Some conflicts can be resolved by adding a new activity. These activities usually provide some resource that was previously not available. Finally, a conflict can also be resolved by simply deleting the culprits. This is obviously not a preferred method and is only used as a last resort.

The resolution of a conflict greatly depends on the type of resource that is in violation. There are five different types of conflicts corresponding to the five types of resources. A state conflict occurs when an activity requires the resource to be in a state which it is not. The culprits in this type of conflict are all of the activities that require the incorrect state and the activity that changed the resource to the incorrect state. Several possibilities for resolving a state conflict include moving the culprits to another interval where the required state is present or adding an activity that will change the state of the resource to the required state.

A concurrency conflict is when an activity requires the presence of the resource during a time for which it is absent. The culprits in this type of conflict are all of the activities that require the presence of the resource. To resolve a concurrency conflict, the scheduler can move the culprits to an interval where the resource is present or add an activity that provides the presence of the resource.

A depletable conflict means that the activities of the schedule have used too much of the resource. In this type of conflict, the culprit is the activity that caused the resource to overflow during the time that it first overflows. Some depletable resources have "resetter" activities and this sort of conflict can be resolved by adding an activity that "resets" the available resource. For example, a downlink activity will free up space in the downlink buffer. A nondepletable conflict is when activities overuse a resource during a particular time interval. The culprits in this type of conflict are all of the activities that use the resource during the conflict interval. This sort of conflict can be resolved by moving or deleting culprits. There are no activities in the DATA-CHASER model that can add to a nondepletable resource.

Simple conflicts occur when two or more activities use the same resource at the same time. This type of conflict can only be resolved by moving culprits.

For any type of initial schedule, the schedule repairer must find the correct activities to move, add, or delete and position them temporally in such a way that no conflicts remain. The scheduler makes decisions randomly except at certain choice points where heuristics are used. The scheduler relies on some interface functions to PI2 that describe the conflicts in the current schedule, describe the activities that could resolve a conflict, and manipulate the schedule. We first describe the random scheduler, followed by the heuristic enhancements that facilitate scheduling within the DATA-CHASER domain. The ultimate task