S7-82
47257
p. 11

Integrated Planning and Scheduling for Earth Science Data Processing

Mark Boddy Jim White Robert Goldman
Honeywell Technology Center (HTC)
MN65-2200, 3660 Technology Drive
Minneapolis, MN 55418
{612-951- 7403 | 7355 | 7336} Fax 612-951-7438
{boddy | jwhite | goldman}@src.honeywell.com

Nick Short, Jr.
ISTB, Code 935
NASA Goddard Space Flight Center
Greenbelt, MD 20771
{short@dunloggin.gsfc.nasa.gov}

Abstract

Several current NASA programs such as the EOSDIS Core System (ECS) have data processing and data management requirements that call for an integrated planning and scheduling capability. As we have shown in previous work, the scale and complexity of data ingest and product generation for ECS will overwhelm the capabilities of manual planning and scheduling procedures. Meeting this challenge requires the innovative application of advanced technology. Some of our work on developing this technology was described in a paper presented at the 1994 Goddard AI Conference, in which we talked about advanced planning and scheduling capabilities for product generation. We are now moving to deploy some of the technology we have developed for operational use.

We have implemented a constraint-based task and resource scheduler for the GSFC Version 0 (V0) Distributed Active Archive Center (DAAC) requirements. This scheduler, developed by Honeywell Technology Center in cooperation with the Information Science and Technology Branch and with the V0 DAAC, makes efficient use of limited resources, prevents backlog of data, and provides information about resource bottlenecks and performance characteristics. It handles resource contention, prevents deadlocks, and makes decisions based on a set of defined policies. The scheduler efficiently supports schedule updates, insertions, and retrieval of task information. It has a graphical interface that is updated dynamically as new tasks arrive or existing tasks are completed. The kernel scheduling engine, called Kronos, has been successfully applied to several other domains such as space shuttle mission scheduling, demand flow manufacturing, and avionics communications scheduling. Kronos has been successfully applied to scheduling problems involving 20,000 tasks and 140,000 constraints, with interactive response times for schedule modification on the order of a few seconds on a SPARC10.

In this paper, we describe the experience of applying advanced scheduling technology operationally, in terms of what was accomplished, lessons learned, and what remains to be done in order to achieve similar successes in ECS and other programs. We discuss the importance and benefits of advanced scheduling tools, and our progress toward realizing them, through examples and illustrations based on ECS requirements. The first part of the paper focuses on the Data Archive and Distribution (DADS) V0 Scheduler described above. We then discuss system integration issues ranging from communication with the scheduler to the monitoring of system events and re-scheduling in response to them. The challenge of adapting the scheduler to domain-specific features and scheduling policies is also considered. Extrapolation to the ECS domain raises issues of integrating scheduling with a product-generation planner (such as PlaSTiC), and implementing conditional planning in an operational system. We conclude by briefly noting ongoing technology development and deployment projects being undertaken by HTC and the ISTB.

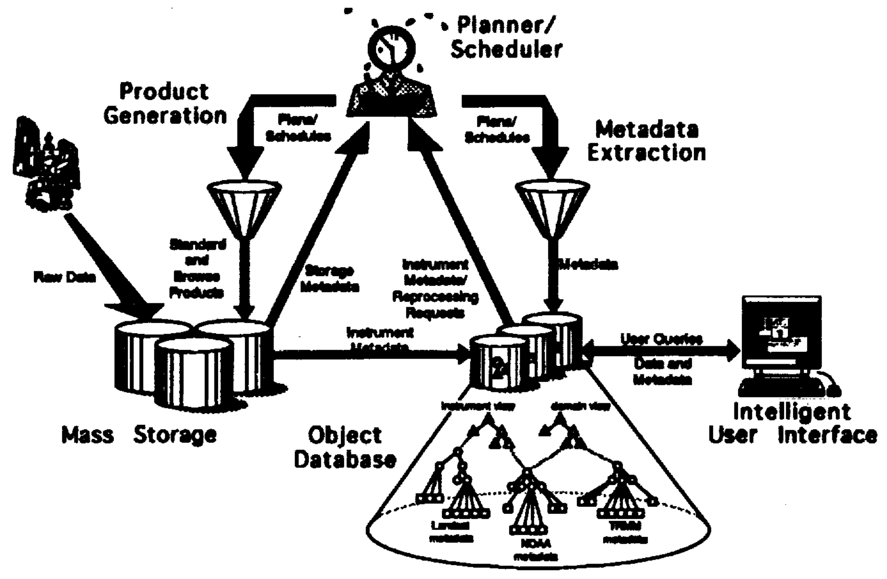


Figure 1: The IIFS

1 Introduction

In both joint and separate work at NASA's Goddard Space Flight Center and the Honeywell Technology Center, we have been working on automating the acquisition, initial processing, indexing, archiving, analysis, and retrieval of satellite earth science data, with particular attention to the processing taking place at the DAACs.

After describing our motivation in section 2 and related work in section 2.1 and section 3 we focus on the DADS V0 Scheduler. In section 4 we present general scheduling requirements initially derived from the DADS application, but extended to encompass similar NASA instrument processing such as the Clouds and Earth's Radiant Energy System (CERES) and the Moderate Resolution Imaging Spectroradiometer (MODIS). Then, we describe the implementation and operation of the prototype DADS V0 Scheduler with particular attention to lessons learned that have enhanced its generality and reusability for other applications. We conclude with a short summary of our conclusions and plans for future work.

2 Motivation

Management of complex systems requires skill in a variety of disciplines. Two critical management disciplines involve deciding what activities to perform, which we call *planning*, and deciding when those activities should be performed, which we call *scheduling*. In large systems such as ECS these functions must be automated, since the sheer volume of data will overwhelm human managers.

It is not sufficient to simply plan and schedule the required activities. These decisions inherently model the target system. Even when this model is made highly detailed, it can never capture all of the details and possible future behaviors of an actual system. Scheduled activities will require more or less time than scheduled. Requests will arrive unexpectedly. Resources will be unavailable or will fail during use. Efficient operation and resource utilization requires that execution must be monitored and future activities rescheduled in response to real world events.

Hence, the overall advantages for using scheduling include:

- automation of routine operations,
- timely delivery of data products,
- efficient use of computational resources.

Satisfaction of these requirements will lead to reduction in staff, use of cheaper hardware, and user satisfaction. These principles are being applied to both the Intelligent Information Fusion System (IIFS) and the DADS in the next sections.

2.1 Intelligent Information Fusion

Since 1989, the IIFS is an prototype system for testing advanced technologies for processing, archiving, and retrieval of remote sensing imagery. The IIFS is currently being applied to the next generation direct-readout domain, whereby data are received from the

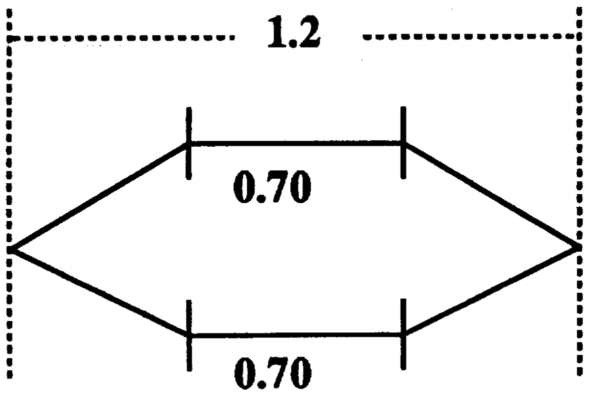


Figure 2: A Simple Problem with Duration and Partial Orders

direct broadcast from orbiting platforms for the region encompassing the acquisition and loss of the spacecraft's signal. These inexpensive ground systems are used for weather forecasting in remote areas of the world, collection of *in situ* data, and calibration/validation of sensor data to name a few.

The planning/scheduling portion of the IIFS is used to manage the production pipeline. Essentially, the planner/scheduler sacrifies accuracy for time in generating the data product. That is, if enough time and resources exist, then the planner/scheduler generates the normal data product, called a standard product in the EOS nomenclature. If not, then the planner substitutes computationally cheaper algorithms until resource constraints can be met. The result is called a browse product and is used soley by the scientists during the data selection phase. Should the scientist decide that greater accuracy is required, then the plan can be regenerated under less computationally constrained circumstances. This may, for example, involve issuing a request to EOSDIS' DADS if the direct-readout center is incapable of handling the request.

During the past few years, several planner/schedulers have been tested in this domain. The next section briefly discusses only one of these planners, called PlaSTiC.

3 PlaSTiC

PlaSTiC (Planning and Scheduling Tool implemented using Constraints) is an automated planning tool designed to automatically generate of complex plans. PlaSTiC was developed as a prototype for the generation image analysis plans (browse products and scientific data) in the EOSDIS domain.

A typical plan might detail the processing steps to be taken to clean up, register, classify, and extract features from a given image. Plan steps will be executed in a resource-limited environment, competing for such resources as processing time, disk space, and the use of archive servers to retrieve data from long-term mass storage. Choices of these algorithms depends on the type of satellite, region of the country, computation characteristics (e.g., deadline, resource requirement, etc.).

PlaSTiC is an integration of hierarchical planning and constraint-based scheduling. TMM provides the basis for temporal reasoning and constraints. The planning component is based on an implementation of NONLIN developed at the University of Maryland. PlaSTiC extends NONLIN-style planning to include reasoning about durations and deadlines.

The schemas used by PlaSTiC, which are based on NONLIN's Task Formalism (TF), have been extended to record information about the estimated and worst-case duration of a given task, and about the task's resource usage. This information is used during plan construction, for example in the rejection of an otherwise promising expansion for a given sub-task because it requires more time than is available. It is also used in the construction of detailed schedules for image processing tasks.

The fact that actions take time was abstracted out in the earliest domain models. Planners using these models will be of limited use in domains where synchronisation with other events or processes is important. This may include such domains as manufacturing planning and scheduling, spacecraft operations, robot planning in any but the most simplified domains, and scheduling distributed problem-solving or other processing. It certainly includes analysis and retrieval planning within ECS.

Several planners include representations for metric time and action durations. This kind of reasoning tends to be computationally expensive. Forbin, Deviser, and Sipe all suffer from performance problems limiting the size of the problems to which they can be applied. Oplan-2 appears to be able to handle somewhat larger problems than the other planners mentioned here.

Implementing an efficient temporal reasoning system is not the sole hurdle, however. Adding duration to nonlinear plans increases the difficulty of determining whether or not the current partial plan can be refined into a plan that will have the desired effects. In fact, it becomes difficult to determine simply whether the actions described in the current partial plan can even be executed.

Consider the simple plan fragment in Figure 2. There are two unordered tasks, each annotated with an estimated duration. If actions can only be taken in sequence, the two tasks depicted must eventually be ordered. When the planner tries to order them, it will discover that neither ordering will work, because there simply isn't room for them to be performed in sequence. In general, determining whether there is an ordering for a set of actions constrained in this way is a hard problem.

To date, two methods have been used to address this problem. The first is *simulation*: the planner maintains a partial order, and after every modification expends some effort exploring the corresponding set of total orders to ensure that there is some feasible total order [Miller, 1985, Muscettola, 1990]. As generally employed, this is a heuristic method: the planner gives up before exploring the complete set of consistent total orders. Another approach, described in [Williamson and Hanks, 1988], involves organizing a partially-ordered plan into a tree of abstract operator types, known as *Hierarchical Interval Constraints* (HIC). Each HIC type has a function defined for calculating bounds on its duration. For the example in Figure 2, the two activities would be contained in an HIC whose duration was calculated by summing the duration of the included operators. The problem with this approach is the required tree structure. If actions must be ordered for reasons that are not locally determinable (e.g. because of resource conflicts, not because they are sequential steps in some task reduction), this representation will break down. It may be possible to augment Williamson and Hanks' representation to cope with a limited number of special structures representing such nonlocal information.

In PlaSTiC, we have started with the assumptions that resource conflicts are significant, that activity durations are nontrivial, and that deadlines will be a factor. For these reasons, the temporal reasoning underlying PlaSTiC is implemented in a full-fledged scheduling engine, so that resource conflicts can be noted and resolved *as part of the planning process*. Similarly, deadline checks are performed automatically as task reduction and order proceeds, triggering backtracking as necessary. The task hierarchy employed by PlaSTiC maintains at all levels a set of duration estimates, so that deadline and resource conflicts may be noticed before a task is expanded all the way to primitive actions. This approach is consistent with the simulation-based technique described above, but so far we have had considerable success in simply resolving possible problems (e.g., potential resource conflicts) as they arise.

The scheduling component of PlaSTiC is built on the Kronos scheduling engine. The DADS V0 Scheduler described below employs this same technology, but with significant extensions to address domain specific scheduling and system integration issues.

4 DADS V0 Scheduler

Unlike direct-readout centers which will dynamically create data flow sequences, the DADS of EOSDIS maintains a database of fixed data flow diagrams. These are retrieved upon request from a database to accomplish various DADS functions. Hence, the DADS required only scheduling and dispatch technology for nominal operations.

In particular, the DADS V0 Scheduler is responsible for scheduling actions and resources to ingest data from a network to buffer disks, transfer buffered data to a mass storage archive, and to retrieve archived data upon request. The scheduler was developed concurrently with the design and implementation of the GSFC V0 DADS. Consequently it was essential that the architecture and interfaces be able to tolerate changes as the system design evolved. The baseline architectural environment of the scheduler is depicted in Figure 3. This environment continues to evolve, but its conceptual and functional characteristics remain stable, so many system changes can be accommodated in the Application Program Interface (API).

The DADS Manager submits scheduling requests, handles errors, and retrieves schedule information. The Task Dispatcher periodically queries the scheduler for a list of upcoming scheduled activities to be executed. The execution monitor notifies the scheduler of events that affect the schedule.

4.1 Approach

The scheduling tool described in this paper was designed to meet the scheduling and resource allocation needs of the GSFC V0 DAAC while simultaneously using the IIFS as a testbed.

Constraint envelope scheduling technology offers an attractive, proven method of meeting the scheduling needs of data archiving and distribution. This technology, embodied in Honeywell's enhanced implementation of the Time Map Manager (TMM), supports the concept of a Temporal Constraint Graph (TCG) which can be used to represent multiple projections of future system behavior, thereby providing rapid rescheduling with minimal disruption in the presence of schedule uncertainty.

The DADS V0 Scheduler is an application of the Kronos scheduling engine, built on top of TMM. Kronos has been successfully applied to domains such as

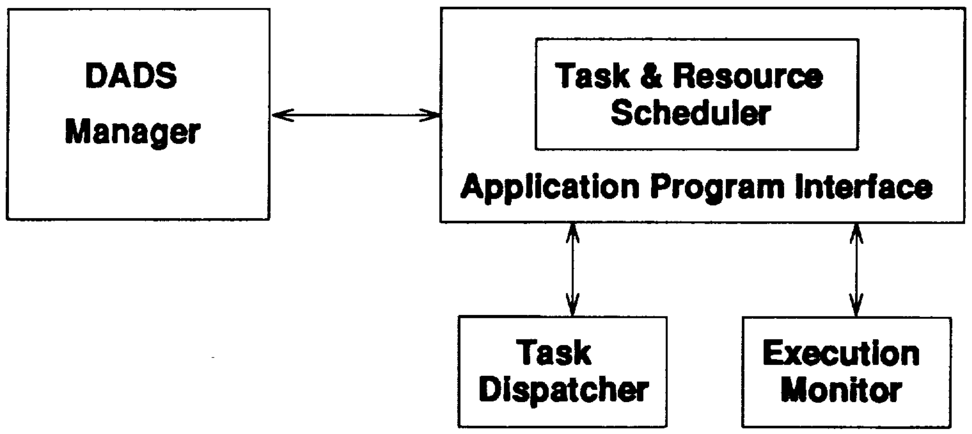


Figure 3: The DADS VO Scheduler's Architectural Environment

space shuttle mission scheduling, demand flow manufacturing, and avionics communications scheduling. It has handled scheduling problems involving 20,000 tasks and 140,000 constraints, with interactive response times for schedule modification on the order of a few seconds on a SPARC10.

4.2 Scheduler Requirements

Detailed scheduler requirements were initially established for the DADS application, then extended and adapted to encompass the scheduling needs of other NASA programs. The following paragraphs summarize requirements at a high level. They confirm the need to be appropriate to the application domain, to be compatible with the target system, and to provide responsive performance reliably.

Domain Appropriate - Commercial scheduling tools sacrifice domain relevance to extend their range of applicability, and hence their marketability. They often lack the capacity to efficiently handle the precise scheduling needs of large, complex applications. In order to select or define a scheduling tool that is domain appropriate, application driven requirements must be established. Whenever possible, these requirements should be based on multiple examples of domain operations and scheduling functions using realistic data sets. They must include quantitative demonstration that capacity and performance goals can be met simultaneously.

Since the GSFC V0 DADS is being developed concurrently with the prototype scheduler, we were careful to maintain a high degree of generality in the scheduler implementation. By first building a core scheduling capability derived from our Kronos scheduling engine, and then extending that capability through specialization, we were able to meet the specific needs of DADS while providing a scheduling tool that can easily be applied to similar problem domains.

Stated as a system requirement, the scheduling core domain model must be compatible with objects and functions required by the target application. Further, its customization capabilities must support accurate modelling of every schedule relevant aspect of the domain. Care should be taken to ensure that this model reflects the intended scheduling policies and procedures of the application, and not the characteristics of analytical models used to project system performance.

Details of the scheduling core domain model are described in section 4.4.1. For the prototype scheduler, subclasses were created to capture application specific attributes and relationships. These attributes may be used to carry system data through the schedule or to support performance monitoring and analysis.

In one instance this derivation was particularly enlightening. The Kronos scheduling engine associated resource utilization with the duration of the activities to which a resource was assigned. If a common resource was to required by multiple disjoint activities, it was expected that a an encompassing parent activity would specify the requirement and would be assigned the shared resource. In the GSFC V0 DADS, there is no encompassing parent activity. Resource utilization can be initiated by one activity (e.g., through transfer of network data to a space on buffer disk) and must persist indefinitely into the future (e.g., until a future activity transfers it to the archive).

By creating persistent requirement and persistent resource profile classes as subclasses of the requirement class and resource profile class, respectively, we were able to provide the necessary scheduler functionality with a minimum of disruption. Persistent requirements have the option of specifying that they begin, use, or ending with their associated activity. This allows the resource allocation to be open ended if desired.

To be effective, any tool must be functionally complete. That is, it must be able to solve the problems to which it is applied. A scheduler must enforce structural constraints (i.e., predecessor-successor and parent-child relationships), temporal constraints (e.g., earliest start or deadline), and resource availability constraints while carrying out the desired scheduling and resource allocation policies in an automated fashion. In the prototype scheduler, policies are currently encoded as functions and a domain specific algorithm (as described in section 4.4.3. We plan to eventually excise policy details from the scheduler by defining syntax for policy specification. This specification will then be input to the scheduler and used to control scheduling and resource allocation decisions.

Compatible - The scheduling tool described here is designed be integrated as a functional component into the target application system. It cannot dictate requirements to that system, rather, it must adapt to the physical and logical demands of the encompassing system. The scheduler must execute on available hardware running the specified operating system. It must be able to communicate with asynchronous functional modules of application system via standard interprocess communication system facilities.

The scheduler must also be linguistically compatible with the surrounding system. It must be able to interpret and respond appropriately to requests for service and information. The prototype scheduler meets this requirement in several ways. The scheduler includes an API customized to the syntactic and semantic needs of the DADS modules with which it interacts. An underlying set of basic API functions facilitates this customization.

The scheduler supports the notion of activity state. The exact states and legal state transitions are defined for the application. In DADS, activities can be scheduled, committed, dispatched, executing, complete, or failed. Additional states and even additional state dimensions can be added as the need arises.

Responsive - Performance is often a critical requirement, but it is frequently overlooked in scheduling. It is assumed that scheduling will be performed once in an initial scheduling effort and that the resulting schedule will satisfactorily describe the actual execution of activities. This view is seldom correct.

We have segregated the total problem into two phases, planning (what to do) and scheduling (when to do it). By making this distinction, we have not only, made each aspect more manageable, but we can tailor the functionality an performance of each component's implementation to the needs of the application. Planning typically occurs before scheduling, though replanning may become necessary. In the GSFC V0 DADS application, there is a small set of functions to be performed (e.g., ingestion, distribution). These can be pre-planned in advance and described to the scheduler as tasks (with subtasks).

The scheduler must, on demand and in near real time, fit each new instance of a task into the current schedule in accordance with task priorities and deadlines while ensuring that necessary resources will be available. As actual events occur in the execution of the scheduler, it must rapidly reschedule to reflect the impact of the event. It must provide data to support graphic presentation of the current schedule, and even allow operator manipulation of tasks.

Reliable - The fault tolerance approach employed by the target application must be supported by the scheduler. In the GFSC V0 DADS this translates to requirements for redundant archiving of schedule information and rapid recovery of the schedule after a failure. The prototype scheduler does not fully include these features at present. However, basic mechanisms needed for reload are present in the script processor described in section 4.3. Also, previous schedulers based on the Kronos engine have included schedule storage and reload capabilities.

4.3 Prototype Environment

The DADS V0 Scheduler is being developed concurrently with the GSFC V0 DADS. Consequently it was necessary to provide a stand-alone environment in which to test and demonstrate scheduler functionality. The operation of components external to the scheduler was simulated via a script processor as shown in Figure 4. The script processor is controlled from a demonstration Graphical User Interface (GUI) that displays schedule activities and resource utilization profiles. Snapshots of the demonstration GUI screen may be seen in Figures 7 and 8. The GUI supports selection and execution of an event script which the script processor translates into API commands that it sends to the scheduler.

A typical script initializes the scheduler by describing the resources available for scheduling, commands the creation of activities to be scheduled, and simulates execution events such as completion of execution. The script also notifies the GUI as objects to be displayed are created.

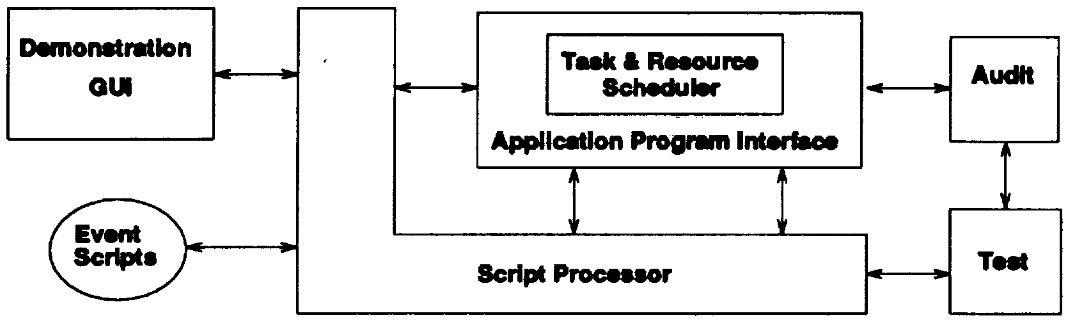


Figure 4: The Prototype System Architecture

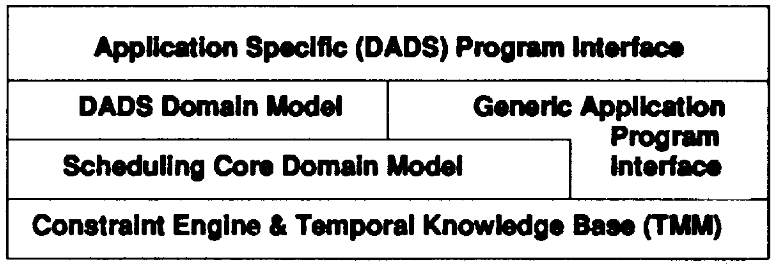

Application Specific (DADS) Program Interface
DADS Domain Model
Generic Application Program Interface
Scheduling Core Domain Model
Constraint Engine & Temporal Knowledge Base (TMM)

Figure 5: The Architecture of the Scheduler

Graphical presentation of scheduler operation is visually convincing, but it is inconvenient for testing and benchmarking purposes. Recently, auditing and test functions were added to facilitate execution and validation of complex event scripts. The test function automates the execution of scripts and the invocation of the audit function, which checks the schedule for consistency and correctness.

4.4 Architecture of the Scheduler

The internal architecture of the scheduler is depicted in Figure 5. The base layer supplies basic temporal reasoning capability. This includes objects such as uncertain time-points and constraints, and functions for updating and querying the temporal knowledge base.

The Scheduling Core Domain Model supplies the basic objects and functions needed for scheduling and resource management. Combined with the Generic API, these layers form a core scheduling capability that can be applied to various scheduling domains. In the DADS V0 Scheduler implementation, the base domain model was extended through specialization and extension to provide appropriate domain-specific capabilities, shown in the figure as the DADS Domain Model and the DADS API.

4.4.1 Domain Model

Key object classes of the scheduling core domain model include resources, requirements, activities and hierarchical activities. These are shown in Figure 6 along with related objects classes of the DADS scheduling domain model.

An activity represents an action to to be scheduled. Each activity has an associated main-token which defines its end points in time and its possible duration range. An activity may be linked to multiple resource requirements. These abstractly define attributes that must be satisfied by the resources allocated to the activity. A subclass of the activity allows hierarchical activity structures to be defined. These were used in the DADS scheduler to implement tasks with component subtasks.

As an example, in the DADS application, a data ingestion task will have several subtasks. The data buffering subtask requires access to the FDDI network and a specific amount of space on one of the data ingestion magnetic disks. A subsequent archiving subtask requires access to the data on buffer disk and space on the UNITREE archive magnetic disk.

The core resource classes allow resources to be conceptually organized into pools using a hierarchical name structure (which permits wildcards) and using a list of resource attributes. Each resource has an associated availability that defines the maximum quantity of that

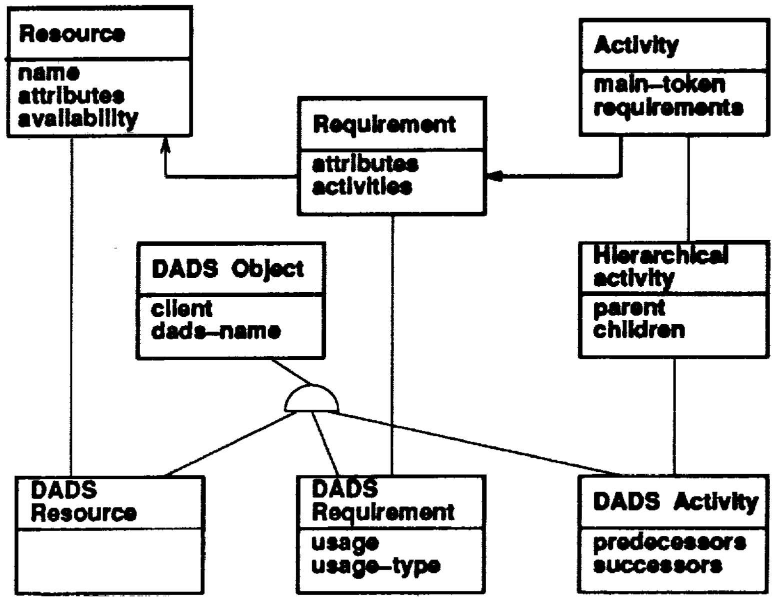


Figure 6: Key DADS Scheduling Object Classes

resource and its temporal range.

Specializations of the core object classes extend the hierarchy to include characteristics of the target domain. In the DADS scheduler these specializations share a common parent class, the DADS object, which defines attributes every DADS activity, resource requirement, or resource must have. Only the client and dads-name attributes are shown in the figure.

4.4.2 Application Program Interface (API)

The Application Program Interface was specified formally by documenting data content (i.e. fields and forms) of the primary information components (i.e. tasks, subtasks, resources, etc.) exchanged between the scheduler and DADS subsystems. For each command, the documentation details the participants in the exchange utilizing the command, the conditions under which the command occurs, the intent (semantics) of the command, and the scheduler's response to the command under both normal and error conditions.

The following command categories describe the functions of the scheduler visible via the API. The categories have been intentionally kept rather abstract and high level here. Not all command categories have been fully implemented in the prototype scheduler.

Definition/Instantiation - Inform the scheduler of the existence of scheduling entities such as activities (i.e. tasks and subtasks), resources, and abstract resource utilization requirements. These commands do not cause scheduling to occur.

Modification - Change the specifics of information known to the scheduler. This category encompasses only changes to the scheduling problem (e.g. relaxation of a deadline). It does not include notification of real-world execution events.

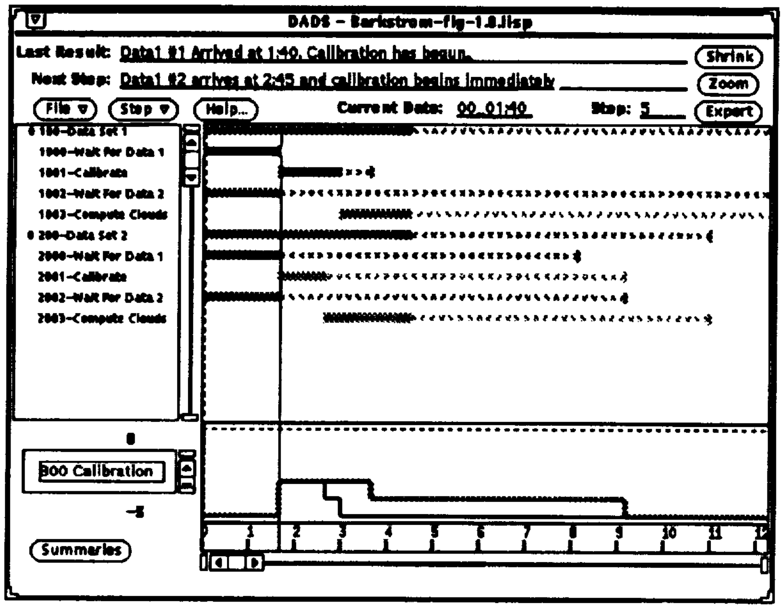
Interrogation/Retrieval - Retrieve schedule and resource allocation information from the scheduler. This information is based on the scheduler's model of the problem space, its record of past events, and its projection of future events including resource utilization.

Scheduling/Rescheduling - Compute a new schedule with resource allocations. Commands in this category may be invoked indirectly by commands in the Update/Synchronization category.

Update/Synchronization - Inform the scheduler of the occurrence of real-world events (e.g. activity execution completion) which may affect the schedule. This category also includes commands for the transfer of responsibility for an activity from the scheduler to another subsystem (e.g., an execution monitor or dispatcher).

Notification - Inform another subsystem that a problem (or potential problem) has been detected by the scheduler.

Communication Handshaking - Provide positive acknowledgement of information transfer.


DADS – Barkstrom-fig-1.8.lisp

Last Result: Data1 #1 Arrived at 1:40. Calibration has begun.

Next Step: Data1 #2 arrives at 2:45 and calibration begins immediately

Current Date: 00 01:40 Step: 5

100–Data Set 1
1000–Wait For Data 1
1001–Calibrate
1002–Wait For Data 2
1003–Compute Clouds
200–Data Set 2
2000–Wait For Data 1
2001–Calibrate
2002–Wait For Data 2
2003–Compute Clouds

800 Calibration

Figure 7: Schedule after Data 1 #1 Arrives

Fault-Tolerance/Recovery - Support for information backup and recovery from failures.

4.4.3 Scheduling Policy

The operation of the scheduler is controlled by scheduling policies. These are currently captured in domain specific algorithms for resource assignment and activity scheduling.

The baseline resource assignment and scheduling algorithm is:

For each activity to be scheduled:

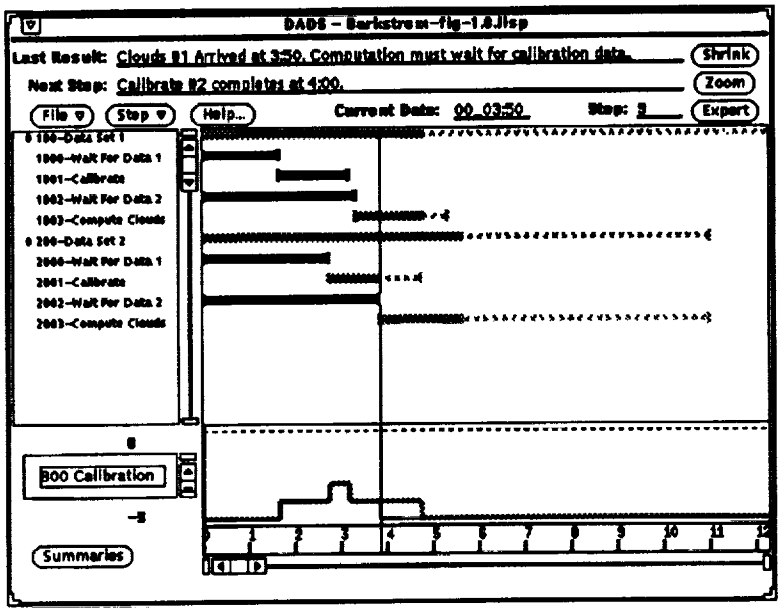
- **If the activity has component activities, Schedule** each of its component activities (i.e., apply this algorithm recursively).
- **If the activity is scheduleable, For each resource requirement** of this activity:
  - **If a satisfactory resource is available** for use without causing it to be oversubscribed, **assign that resource** to meet the requirement. Availability implies that the resource is part of the resource pool specified in the resource requirement and has the attributes specified in the resource requirement.
  - **If no satisfactory resource is available, apply the following stratagems** in sequential order, using the possible resources until one of them successfully eliminates the oversubscription:
    - **Constrain the order of activities** involved in the oversubscription:
      - Individually *before* the activity, or
      - Individually *after* the activity, or
      - Collectively *before* the activity, or
      - Collectively *after* the activity.
    - **Relax the deadline** of activities involved in the oversubscription **and constrain the order** of activities (as above)
    - **Constrain the order of parent activities** of the activities involved in the oversubscription (as above)
    - **Report failure** [and Exit]
- **If the activity is still scheduleable and all component activities** of this activity **have been scheduled,** Mark the activity scheduled.

Then update:

- The schedule's temporal knowledge base,
- The time bounds of all changed resource utilization profiles.

4.5 Scheduling Example

The operation of the prototype scheduler is revealed in Figures 7 and 8. In this simple example, taken from the Clouds and the Earth's Radiant Energy System (CERES) domain, two instances of a single task type



Last Result: Clouds #1 Arrived at 3:50. Computation must wait for calibration data.
Next Step: Calibrate #2 completes at 4:00.
Current Date: 00 03:50 Step: 9

100–Data Set 1
1000–Wait For Data 1
1001–Calibrate
1002–Wait For Data 2
1003–Compute Clouds
200–Data Set 2
2000–Wait For Data 1
2001–Calibrate
2002–Wait For Data 2
2003–Compute Clouds

Figure 8: Schedule after Data 2 #2 Completes Execution

have been scheduled. Each task consists of four related subtasks with interdependencies. The first subtask is to wait until a particular radiation budget data set arrives. The second subtask is to calibrate and Earth-locate that data set. A calibration resource is required by this subtask. The third subtask is to wait for a corresponding cloud identification data set. The final subtask is to compute cloud data by combining the calibrated radiation budget data with the cloud data to produce a combined data product. The Calibrate subtask cannot occur until its Data 1 is available. The Compute Clouds subtask cannot occur until the Calibrate subtask is complete and its Data 2 is available. For illustrative purposes, the second task has been given a deadline of 11:00 while the first task has no deadline.

Figure 7 shows the situation after the first dataset arrives. The earliest scheduled time for each activity is shown to the right of its name as a solid horizontal bar. Dashed lines indicate the the range of possible occurrences of the activity. The current time is represented as a vertical line.

Subtask 1001 has now started because subtask 1000 has finished. Subtask 1003 cannot start until subtask 1001 completes. Subtask 2001 could start immediately, but since its predecessor subtask, 2000, is still executing, it will slip as time passes. Because of a similar predecessor dependency on subtask 2001, subtask 2003 will also slip. The scheduler automatically reschedules the earliest start and earliest end times of these activities as time passes.

The resource utilization profile of one of the resources used by the example activities is shown at the bottom of Figure 7. The profile indicates both the scheduled (black) and potential (gray) utilization of the resource. The API of the DADS V0 Scheduler provides query commands for determining the relationships between resource utilization and scheduled activities, but in this example careful examination of the shape of the profile reveal that increments of the Calibration tool resource have been allocated to satisfy the requirements of subtasks 1001 and 2001.

At a later time, after more of the subtasks have completed execution, the situation is noticeably different. This is shown in Figure 8. Subtask 1003 did not start immediately after Subtask 1001 (Calibrate) because of its additional dependency on the completion of subtask 1002 (Data 2). Notice that although task 100 has no deadline, a maximum end time for subtask 1003 has been scheduled because that subtask has an associated maximum duration.

The resource utilization profile for the Calibration tool resource has changed significantly from that projected in Figure 7. This is because the start of subtask 2001 could not be predicted reliably because of its dependency on the completion of subtask 2000. The execution of subtask 2001, and the utilization of the Calibration resource was rescheduled until its Data 1 arrived.

Even this simple example shows that accurate scheduling and optimization of resource usage requires a scheduling tool that can rapidly reschedule future activities in response to real-world events.

5 Summary, Conclusions and Future Work

In this paper, we have presented results of the application of Honeywell's scheduling technology to an application of data archiving and distribution. We have described our progress to date and some insights regarding further application of this technology to other domains. Moving to broader operational use will require further refinement and development.

We plan to continue development and refinement of the planning and scheduling capabilities described in this paper. Our efforts will be focused on increasing their applicability and achieving the goal of realization of the Intelligent Information Fusion System. In the near term we will be provide documentation, training, and support materials in order to obtain design feedback through use of these tools. We will simultaneously continue to extend their functionality in support of additional application domains.

References

[Miller, 1985] David P. Miller. Planning by search through simulations. Technical Report 423, Yale University Computer Science Department, 1985.

[Muscettola, 1990] Nicola Muscettola. Integrating planning and scheduling to solve space mission scheduling problems. In *Proceedings DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 220–230. Morgan Kaufmann, November 1990.

[Williamson and Hanks, 1988] Mike Williamson and Steve Hanks. Efficient temporal reasoning for plan projection. In James Hendler, editor, *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 313–314, 1988.