Enhanced Satellite Constellation Operations via Distributed Planning and Scheduling

Subrata Das, Curt Wu
Charles River Analytics, Inc.
725 Concord Avenue
Cambridge, MA 02138, USA
+1 617 491 3474
{sdas, cwu}@cra.com

Walt Truszkowski
NASA Goddard Space Flight Center
Code 588
Greenbelt, MD 20771, USA
+1 301 286 8821
Walt.Truszkowski@gsfc.nasa.gov

Keywords distributed, planning, scheduling, autonomy, satellite, constellation, HTN, KQML, Java, Prolog

Abstract

In this paper, we present a system for Distributed Intelligent Planning and Scheduling (DIPS) that helps a spacecraft function as an autonomous agent. A DIPS-based spacecraft receives only high-level goals from ground station operators, and performs its own planning and scheduling onboard, achieving these goals with onboard subsystems and in cooperation with other spacecraft. The task decentralization in DIPS employs a domain distribution algorithm that typically creates a feasible schedule after the first coordination effort, thereby decreasing inter-agent negotiation during the scheduling process. The reasoning performed by DIPS agents to optimize time and resource usage while maintaining flight rules and constraints is based on a constraint propagation paradigm. Priority-based scheduling is implemented, and a hierarchical inter-agent confirmation/authorization system is used for global goal coordination. An enhanced prototype is developed and demonstrated using space-based scenarios involving onboard instruments and a satellite constellation. The vertically layered architecture of the DIPS prototype integrates: 1) Java-based agent inference engine; 2) Prolog platform SICStus for constraint-based reasoning; and 3) KQML for inter-agent communication. We are specifically targeting our effort to enhance the planning and scheduling capability of NASA’s planned nanosatellite constellations.

1 Introduction

Spacecraft autonomy has the potential for effecting significant cost savings in mission operations by reducing the need for dedicated ground staff. In an autonomous operating mode, operators will communicate only high-level goals and deadlines directly to the spacecraft. The spacecraft will then perform its own planning and scheduling by decomposing a goal into a set of sub-goals to be achieved in cooperation with other spacecraft in the environment. In this paper, we present the Distributed Intelligent Planning and Scheduling (DIPS) system that helps a spacecraft function as an autonomous agent by incorporating this distributed approach to onboard planning and scheduling.

The term planning refers to the generation of activities that satisfy a current set of goals. For example, a planning process to satisfy the request for an image generates activities such as rolling the camera to the correct position and activating the camera shutter. The term schedule describes an association of these specific activities with particular start and end times by satisfying temporal constraints (e.g., rolling should be performed before the shutter action). The onboard spacecraft subsystems must execute these time-sensitive activities autonomously to achieve the goals, and if none of the subsystems of the spacecraft is capable of executing an activity then cooperation from another spacecraft in the environment is required.

In our envisioned distributed (or equivalently, multi-agent) environment [1, 2], a set of problem-solving autonomous agents based on DIPS communicate, cooperate, and negotiate to achieve high-level goals through planning and scheduling. Distributed planning and scheduling emphasizes a decentralized organization in which schedules are generated and executed cooperatively and concurrently by agents. This can be contrasted with a centralized planning environment in which goals, rules, and constraints from individual agents are accumulated at a central place, and a centralized planner is used to generate a global schedule. The centralized approach is particularly unsuitable when the problem is inherently distributed such as in a spacecraft environment where each subsystem or spacecraft functions autonomously.

The domain knowledge of tasks and their components in DIPS are manifested through a hierarchical language taking into account spacecraft operational aspects and resource constraints. The task decentralization in DIPS is performed by employing a domain distribution
algorithm that typically allows a feasible schedule after the first coordination effort, thereby decreasing inter-agent negotiation during the scheduling process. The reasoning performed by DIPS agents for scheduling tasks by optimizing time and resources is based on a constraint propagation paradigm. The vertically layered architecture of the DIPS prototype integrates: 1) Java-based agent inference engine; 2) Prolog platform SICStus for constraint-based reasoning; and 3) KQML for inter-agent communication.

2 Past Work in AI Planning

Two major trends for task representation in the history of AI planning have been observed [3]: goal achievement (GA) and hierarchical task network (HTN). The origin of GA-based planning is in STRIPS [4]. In this model of representation, an initial situation, a set of possible actions, and a goal that is to be achieved are given. Planning consists of finding a sequence of actions that would lead from the initial situation to the final one. Several planners were subsequently built on the GA model including TWEAK [5], and SNLP [6]. In a planner based on the HTN representation, which originated with NOAH [7], planning proceeds by selecting a non-primitive task, decomposing it into subtasks using a library of available decomposition methods, and then detecting and resolving conflicts with other tasks. This process is repeated until no non-primitive tasks remain and all the conflicts have been resolved. Typical examples of HTN planners are FORBIN [8], and NONLIN [9]. There are also planners combining features from these two such as O-Plan [10] and SIPE [11].

Given a representation in either GA or HTN, solving a planning problem can be viewed as a straightforward search problem, but in general the HTN paradigm can lead to more efficient planners because it allows the user to limit the search space by guiding the planner towards acceptable solutions. A typical implementation of the search engine of a planner operates on a temporal database such as the HSTS system [12] and Time Map Manager [13]. The search engine posts constraints to the database. The temporal database then constructs a constraint network and provides a constraint propagation service [14] to verify the global consistency of the posted constraints with the goals, rules, and constraints of the spacecraft. Both the consistency checking and the search for an optimal solution in cooperation with other agents in the environment are computationally intractable, that is, NP-hard. A distributed approach to planning and scheduling allows cooperation among agents in the environment and increases efficiency in the search for an optimal solution by partitioning the whole search space.

In recent years, there has been a growing interest in agent-oriented problem solving [15], which provides the basis of our distributed solution [16] to planning and scheduling. The agent-oriented problem-solving environment increases efficiency and capability [17] by employing a set of agents that communicate and cooperate with each other to achieve their goals. We use the term agent to refer to an entity that operates autonomously or semi-autonomously while communicating with other agents in the environment. Although types of agents range from software agents [18] implementing the behavior of humans, machines or hardware, to mechanical or electronic robots [19] with the capability of perceiving or sensing the environment and executing appropriate actions, our assumption is that every agent will have an interface that understands a common communication language.

3 Satellite Constellation Scenario

We present a scenario that will illustrate our envisioned distributed planning and scheduling by incorporating several key problem areas prevalent in a distributed scheduling environment. The simulation of this environment consists of a constellation of satellites, each with a number of local resources and the knowledge of hierarchical task decompositions. The hypothetical constellation is loosely based on the Three Corner Satellite Constellation, a joint effort of Arizona State University (ASU), University of Colorado at Boulder (CU), and New Mexico State University (NMSU) that is expected to launch in late 2001.

3.1 Scenario Hierarchy Specification

The Satellite Constellation Scenario includes a constellation of spacecraft with the following hierarchy:

![Figure 1: Scenario Hierarchy](image-url)
• ASU1, CU1, NMSU1: These agents represent ground control stations on Earth. Each is the sole parent for a satellite (3CSAT1–3CSAT3), and they all are shared parents for the remaining satellites (3CSAT4–3CSAT6).

• 3CSAT1–3CSAT6: Each of these DIPS-based agents represents a satellite in the constellation. Every satellite has essentially the same collection of instruments onboard with slight variations in capability/efficiency.

• Onboard Resources: Locator (0-1), Transmitter (2), Recorder (1), and Camera System (1).

Each resource onboard a DIPS Satellite Agent has its own local DIPS Resource Agent. The Camera agent is an intermediate DIPS System Agent introduced on each satellite in order to manage its own group of resources.

3.2 Basic Distributed Scheduling

In order to demonstrate basic distributed scheduling capabilities, the scenario begins with a number of high-level goal requests introduced to the constellation by the three ground control agents. Every capable DIPS 3CSAT Agent decomposes the task, and corresponding subtask requests are sent to onboard resource and system agents. The scenario includes multiple compound task requests with overlapping time domains. The multi-parental satellite hierarchy permits some satellites to receive requests from multiple sources.

3.3 Special Situations

In order to force over-scheduling on some local onboard resources, the scenario includes several conflicting requests. Every agent will prefer to schedule its subtasks on the least expensive resources available—sometimes at the expense of over-scheduling a preferred local resource with tentative subtask requests. Top priority requests are introduced to the constellation as forced requests, which will always succeed and be locked into the necessary local schedules, even if other non-forced tasks have been scheduled on those resources at conflicting times.

As a task nears its execution time, its Time To Live (TTL) decreases. A DIPS-based agent will process a task request with a very short TTL before another task request with a long TTL, even if the latter is of higher priority.

The scenario also includes temporary lapses in communication between specific satellites and other DIPS agents in the community. A satellite will move out of the range of a ground station for a specified time period, during which a message relay system using other agents in the community may still allow messages to reach their destination.

An important real-life issue in a satellite constellation is the management of resources and physical properties. There may also be certain constraints on power consumption that affect resource availability. The scenario includes several task requests that require a DIPS Satellite Agent to reason about its resources and physical properties. Satellites also are required to perform certain mandatory maintenance tasks throughout the scenario.

The scenario allows some idle time for each satellite during which it may explore other feasible solutions to the problem in order to optimize its schedule and accommodate as many task requests as possible. If all requests have been successfully decomposed, a satellite can use its idle time to search for more optimal usage of time and resources.

4 DIPS Agent Architecture

The architecture of a DIPS agent is deliberative; there is an explicit symbolic representation of the model of the dynamic environment, and DIPS agents make decisions via logical reasoning based on pattern matching and symbolic manipulation. Several different deliberative agent architectures have been proposed in the literature, and two of them are most prominent: horizontally layered architecture [20] and vertically layered architecture [21].

The architecture we have adopted is vertically layered and is displayed in Figure 2. The three layers are the world interface layer, the inference layer, and the network management layer. A DIPS agent’s knowledge base is also split into three modules corresponding to the three layers. Note that only the world interface layer has a direct interface to perception and action.

![Figure 2: DIPS Agent Architecture](image-url)
4.1 World Interface Layer

The world interface layer contains a DIPS agent’s facilities for perception, action, and communication, which all require detailed knowledge about the environment. A DIPS agent’s world model contains information about the environment such as the locations and capabilities of other agents. The world interface layer enables a DIPS agent to communicate with other agents in the environment.

4.1.1 Action and Perception

The action and perception facilities can be handled through an advanced real-time scripting language. A DIPS agent’s actions will be performed via calls to scripts that interact with local hardware. One scripting language candidate is the Spacecraft Command Language (SCL) developed by Interface & Control Systems [22]. SCL, with its innovative Real-Time Engine (RTE), is especially well suited for our real-time scheduling application. The RTE supports both time- and event-based script scheduling as well as real-time resource monitoring and exception handling.

Through such an interaction, schedules generated by the inference layer could be actuated on local resources. The world knowledge, or information about the states of these resources, may be gathered through sensor data. The inference layer can access this through the scripting language, which provides the perception component of the world interface.

4.1.2 Communication

All inter-agent communication and knowledge manipulation is done via message passing. A DIPS agent uses messages in Knowledge Query and Manipulation Language (KQML) compliant format to communicate goal requests, goal confirmations/denials, capability insertions, and other standard KQML performatives. Messages are automatically forwarded to appropriate agents to allow two agents not in direct contact to communicate indirectly. The world interface layer sends and receives messages from other agents in the community and passes them to the inference layer to be handled appropriately.

4.2 Inference Layer

Implemented in Java, the inference layer of the DIPS prototype agent applies its domain knowledge to goal-related messages in order to decompose compound task requests or schedule primitive tasks on local resources.

The domain knowledge consists of the knowledge of the application such as definitions of different task abstractions and the effects of a task when it is carried out. Although most of the domain knowledge is static, it can still be modified at runtime. When a new resource comes online, for example, any new relevant scripts or task definitions can be added to a DIPS agent’s domain knowledge through capability insertions. The content of the domain knowledge and the functionality of a DIPS agent’s inference layer depend on the type of agent. The DIPS system currently recognizes two distinct subclasses within the DIPS agent architecture: System Agents and Resource Agents.

4.2.1 DIPS System Agent

A DIPS System Agent represents a system of primitive resources, so the capabilities of a DIPS System Agent (e.g., a Satellite) consist only of decomposable compound tasks. A DIPS System Agent does not maintain a local schedule for it has no local resources; rather it coordinates a group of subagents, some of which may also be System Agents. When a DIPS System Agent receives a high-level goal request, it uses a predefined decomposition from its domain knowledge to create a plan for that task, and then uses its constraint management layer to create a feasible schedule for that plan. Each subtask in the plan is assigned a certain time interval based on the decomposition constraints, and subtask requests are sent out to all subagents.

4.2.2 DIPS Resource Agent

A DIPS Resource Agent represents a physical onboard resource. Its capabilities are all primitive tasks that can be performed via calls to predefined low-level scripts. When a DIPS Resource Agent receives a primitive task request, it posts new temporal constraints describing the task to the constraint database. The inference layer uses the constraint management layer to verify the feasibility of its new augmented local schedule and to generate an instance if feasible.

4.3 Constraint Management Layer

The constraint management layer of a DIPS agent is implemented using a version of the Prolog language called SICStus Prolog. SICStus Prolog has capabilities for Constraint Logic Programming in Finite Domains (CLPFD) that allow several important developments. Most notably, CLPFD allows arithmetic constraints on variables to be introduced into a program, and it can perform arithmetic on these variables even when they are uninstantiated [23].

A database of constraints is maintained by the inference layer of a DIPS agent, and the SICStus Prolog emulator is used as a back-end schedule solver. When a DIPS Resource Agent receives a task request, it posts any new constraints regarding that task to the constraint database and then queries the Prolog constraint management layer for a feasible instance of its local schedule. When the new constraints are propagated through the constraint network, the schedule will reflect these changes.
A DIPS System Agent uses its constraint management layer for a different purpose. Because a DIPS System Agent has no local resources and therefore no local schedule to maintain, it uses the SICStus emulator to generate regional schedules for task decompositions.

5 Hierarchical Modeling

A planning process based on a HTN representation first constructs a plan containing abstract high-level activities and then refines these components in increasing detail. This process of refinement continues until these high-level activities correspond to the physical actions in the real world. The advantage of this approach is that the feasibility of a plan can be studied incrementally. If a DIPS agent is implementing the above refinement process, then domain knowledge of the tasks and their components need to be codified in some language. We provide here some examples of HTN representations (similar to Das et al. [24]) used in DIPS.

5.1 Compound Goals

A compound task specification used by DIPS has two components: 1) a set of subtasks that compose a possible plan for achieving the goal; 2) a set of temporal constraints including constraints on the ordering of the subtasks. Each of these subtasks may be compound in turn; a DIPS agent can only reason about the immediate sublevel in the HTN. The minimum duration of each subtask is specified as an integer by the “span” constraints, while the rest of the constraints may be any of those recognized by the DIPS constraint management layer. This decomposition plan, a list of subgoals and temporal constraints, is sent through a query to the constraint management layer, which returns a feasible instance of the schedule. The agent can then distribute the subtasks to the appropriate subagents with the allocated portions of the original request domain.

5.2 Primitive Tasks

Each primitive task in DIPS corresponds to a call made to a scripting language such as SCL. These scripts themselves may contain several actions, but due to the fixed nature of a primitive task, these are considered immutable subatomic actions.

6 Agent Communication

Coherence, cooperation and conflict resolution can be improved by carefully designing the amount and type of communication among agents in the form of messages [25]. The information communicated should be relevant, timely, and complete [26]. Inter-agent communication in DIPS uses a KQML-compliant format to enhance robustness and modularity.

A DIPS KQML message is considered valid if it has at least the following fields: sender, receiver, id, and path. Every KQML message also has a performative that describes the type of communication. Most messages also have a content field containing an expression describing the purpose of the message.

Agents from the DIPS system currently use only a subset of the standard KQML performatives. The most commonly used are the following: insert, evaluate, confirm, authorize, and sorry.

7 Decentralization and Coordination

The key concept of the DIPS system is to incrementally partition the scheduling problem into smaller independent subproblems of increasing granularity, which can then be solved in parallel. While this may sacrifice global completeness and optimality in the search for a feasible plan, the distributed scheduling approach greatly reduces the complexity of a large-scale multiple-resource scheduling problem.

7.1 Least Commitment Scheduling

The important development made by the use of the constraint management layer is that relational constraints on time and resource consumption can be specified in the scheduling process. This allows the full and precise description of plans and task decompositions without any specification of actual start times. Empowering a DIPS agent with this mentality of least commitment allows reactive planning and dynamic rescheduling; the Prolog scheduler will allow local changes to be made while adhering to the original global constraints.

As proposed by Allen [27], we recognize seven basic temporal relations between two actions. These relations for two actions, A and B with intervals [S_A,E_A] and [S_B,E_B] respectively, are shown in Table 1. Any other relation can be expressed as the inverse of one of these.

| 1. A before B | E_A<S_B |
| 2. A during B | S_B<S_A, E_A<E_B |
| 3. A overlaps B | S_A<S_B, S_B<E_A<E_B |
| 4. A equals B | S_A=S_B, E_A=E_B |
| 5. A meets B | E_A=S_B |
| 6. A starts B | S_A=S_B, E_A<E_B |
| 7. A finishes B | S_B<S_A, E_A=E_B |

Table 1: Recognized Temporal Constraints

The first three constraints, before, during, and overlaps, are the three basic types of relations. Constraining two actions with these relations allows some flexibility in the instantiation of real start and end times. The last four can be considered special cases of the basic types, which all involve reasoning about
specific start or end times for the actions and do not allow any flexibility for one event once the other has been bound to a real interval.

7.2 Domain Distribution Algorithm

In contrast to previous distributed planning and scheduling systems such as DAS [28], DIPS employs a domain distribution algorithm that typically creates a feasible schedule after the first coordination effort. As an extension to the least commitment approach, each node in the DIPS HTN leaves as much flexibility in the next sublevel as possible when coordinating the domain distribution. The basic underlying algorithm: when choosing a feasible schedule of subtasks for a given compound goal, allocate to each subtask as generous a time interval as possible without allowing global constraints to be violated when the subtasks are instantiated on the next level in the HTN.

We can demonstrate the DIPS generous approach to domain distribution with the following example: Two tasks, A and B, are to be scheduled during the domain [100,200]. The only constraints on the tasks are their durations A spans 10, B spans 30, and A before B. The subtask domains are proportionally expanded as much as possible in order to provide maximum flexibility at lower levels in the HTN.

The schedule will allow flexibility in the instantiation of both subtasks when they are distributed to the subagents. This example follows the principle of least commitment by relaxing the constraints on the tasks as much as possible while still fulfilling the constraints.

Applying a generous domain distribution algorithm is straightforward in the case of the before constraint. In order to distribute portions of a scheduling problem that includes any of the other temporal relations, a system for managing global constraints must be developed.

The DIPS algorithm solves this problem by treating these constraints as special cases. If one subtask is related to another by one of the inflexible constraints, (meets, starts, finishes, equals), the domain size is left equal to the duration of the subtask. If there exist any constraints relating this subtask to another with during or overlaps, several new variables are introduced in order to reason about the constrained intervals, and a limited domain expansion is allowed.

Using the previous example with task durations A=10 and B=30 but with a different temporal constraint: A overlaps B, a domain expansion would then produce the following schedule for the subtasks:

8 Implementation

A scenario is simulated with a Java application that starts each DIPS agent as an independent thread of execution. Each agent has an initialization file of capability insertions in the form of KQML messages. The primary interface has three views: the Architecture View, the Communication View, and the Physical View.

The Architecture View shows the overall layout of the agents in the scenario. In Figure 3 we see that the ASU1 agent has been selected, and its four children are displayed. The tabs below an agent represent child agents that can be expanded if selected.

The Communication View shows the communication status between a given agent and its related agents (i.e., parents, siblings, and children). The View is in 3-D space and can be rotated or scaled to best display the information. In Figure 4 we see that the selected agent, ASU1, has four children that it can potentially communicate with, but currently communication is only enabled with 3CSAT1.

The Physical View shows the view of the satellite constellation and the Earth in real spatial coordinates. Similar to the Communication View, the Physical View can be rotated or scaled as desired. Figure 5 displays the satellites in their physical location relative to the Earth. Note the visualization of the message being transmitted from satellite 6 to 4 via satellite 5.
In the DIPS architecture, each agent maintains a message log, a set of capabilities, and a schedule if appropriate. This information is displayed in a window that can be opened for each active agent.

Figure 6 shows the schedule for a TransmitterSlow agent. The time scale is adjustable. There are currently two transmit tasks that have been scheduled: one tentative and one final. Note that there can be any number of tentative tasks scheduled at a given time, but only one final task can be scheduled for any instance in time. The schedule is automatically updated when appropriate.

Figure 7 below shows the message log for a 3CSat agent. Currently the table is sorted by ID, but the user has the flexibility to sort by any column in either direction and can arrange columns in any order. There is a mix of insert, evaluate, and confirm messages. Note that the confirm messages have a cost associated with them. As with the Agent Schedule, the Agent Message Log is automatically updated when new messages are received by the agent.

Figure 7: Agent Message Log

In addition to testing conflict resolution and other scheduling capabilities in the scenario, we submit a set of similar task requests within a certain time interval in order to examine the performance of the DIPS-based community under heavy request and scheduling loads. An example scenario consisting of 4 System agents and 14 Resource agents (a total of 18 HTN nodes) can distribute and schedule 35 high-level goal requests corresponding to 280 primitive tasks in under 5 minutes, even when running on a single CPU.

9 Conclusion and Future Work

In this paper, we have demonstrated how our distributed approach to planning and scheduling helps to achieve high-level goals and thereby enhances spacecraft autonomy. A hierarchical syntax has been adopted for representing domain knowledge of task decompositions and employed to solve the task decentralization problem. A constraint propagation paradigm has been employed for the required planning and scheduling tasks performed by an autonomous agent, and an innovative system for the decentralization of a global scheduling problem has been developed and employed with promising results. Priority-based scheduling has been implemented, and a hierarchical inter-agent confirmation/authorization system was used for global goal coordination.

The relative speed with which new requests can be processed indicates two areas of success by the DIPS system: 1) efficient multiple-resource scheduling in a distributed environment by reduction of the complexity of the global scheduling problem; and 2) implementation of a generous domain distribution algorithm that minimizes inter-agent negotiation. The global scheduling problem in our multi-resource scheduling example would be computationally difficult for a single scheduling agent to solve as a whole; yet after decentralization through the DIPS HTN, the total search space was reduced exponentially, and a solution
was produced with relative efficiency. A solution is found with much greater efficiency while still satisfying global constraints.

Due to the emphasis on autonomy, the efforts of the DIPS system currently focus on individual schedule creation and maintenance rather than global schedule optimization. Further efforts include the use of CPU idle time for replanning and schedule optimization based on cost utilities. Parent-child negotiation will be enhanced to solve difficult scheduling problems. There are also ongoing efforts to enhance the GUI.

10 Acknowledgements

The authors would like to thank Paul Gonsalves and Dan Grecu of Charles River Analytics for their contributions to the planning and scheduling algorithm, Dan Knights and Igor Levit of Charles River Analytics for assisting in the implementation, and Allan Posner and David Woodard of ICS for their contributions to the scenario. This work was performed under contract NAS5-99168 with NASA Goddard Space Flight Center.

11 References