A Study for an Autonomous Agent for Leo Satellite Operations

ABSTRACT

There is a need for application of autonomous agent related technologies in the LEO (Low Earth Orbit) satellite domain. Major contributing factors are limited visibility of the satellite and cost of manpower associated with ground control. This paper describes an approach to apply concepts from the remote agent architecture of Deep Space I to arrive at an autonomy module for LEO satellite payload operations. The system described in this paper demonstrates continuous planning. It has a planner and a high level goal database. It assumes perfect execution of the generated plans, so, no diagnosis & recovery module is involved.

1. Introduction

Leo satellite orbits are very close to the earth and they take very less time (2-6 hrs) to go around the planet. Their limited visibility makes the monitoring process difficult as they are available only for minutes in the span of any ground station antennae and the control commands compete with the telemetry for bandwidth. So, there is a need for reduction in the amount of tele-command data and automating the onboard processes i.e. closed-loop control with little ground intervention. This calls for a program which would monitor all the functions of the satellite and would plan for schedules of operations and would make decisions as the situations crop up i.e. an autonomous agent. The most famous example of autonomous agents is the DS-I (Deep Space I) remote agent architecture [1, 3]. The architecture consists of a mission manager, planner scheduler, mode identification and recovery and an executive. The system proposed in this document demonstrates continuous planning. It has a planner and a high level goal database. It assumes perfect execution of the generated plans, so, no diagnosis & recovery module is involved.

2. DS-I Remote Agent:

Remote Agent (RA), an Artificial Intelligence (AI) system automates some of the tasks normally reserved for human mission operators and performs these tasks autonomously onboard the spacecraft. These tasks include activity generation, sequencing, spacecraft analysis, and failure recovery. The remote agent architecture mainly consists of Mission Manager (MM), Planner/Scheduler (PS), Smart Executive (EXEC), and Mode Identification and Recovery (MIR). This was a part of the successfully demonstrated Deep Space I mission of NASA. The remote agent architecture is shown in Figure 2.1.

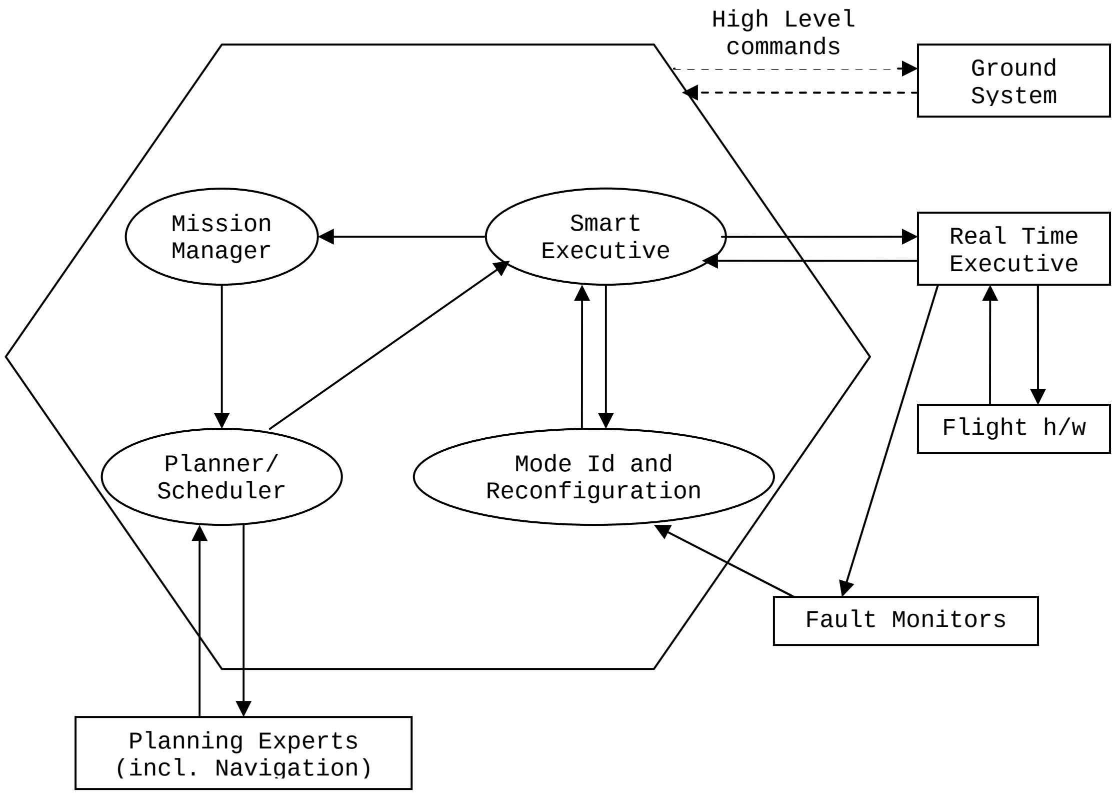

Figure 2.1 DS-I Remote Agent

The MM maintains the mission profile, a partially specified plan containing a database of goals for the entire mission (or a significant portion of it). The PS analyzes the goals, breaks down high-level goals into mid-level activities, orders activities to resolve interactions, and ensures the plan is consistent with resource constraints.

The EXEC invokes planning (PS) when necessary. It also issues a sequence of commands to the real time executive: an abstraction over the spacecraft hardware. The MIR component tracks execution activity. Based on the commands from EXEC and the outputs of the monitors, MIR infers the most likely current state of the hardware and passes its conclusions to EXEC. MIR also assists EXEC in repairing devices by suggesting actions based on the global context. Figure 2.2 describes the flow of data in the remote agent of DS-I.

The prototype demonstrates continuous planning i.e. planning across successive planning horizons. The user enters the goals as and when they crop-up and the system decides when the goals are to be scheduled.

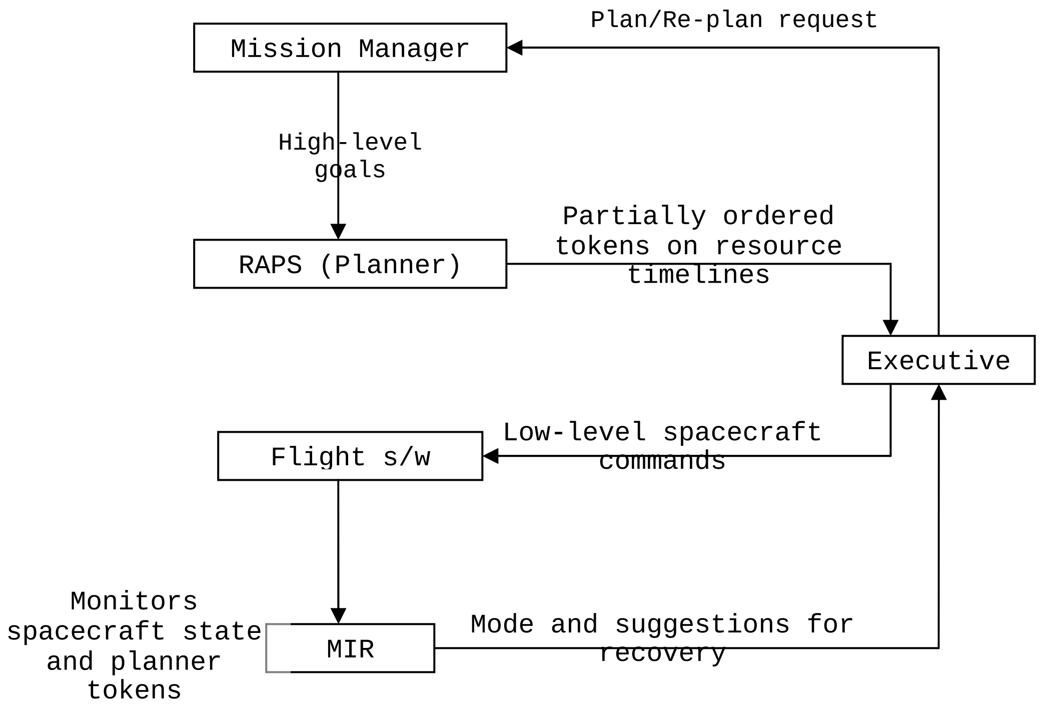
Mission Manager
Plan/Re-plan request
High-level goals
RAPS (Planner)
Partially ordered tokens on resource timelines
Executive
Flight s/w
Low-level spacecraft commands
Monitors spacecraft state and planner tokens
MIR
Mode and suggestions for recovery

Figure 2.2 Autonomy: Flow of Data

3. Prototype Agent

The prototype consists of a mission manager, planner, an executive and a state database as shown in Figure 3.1. The working of the components is explained below.

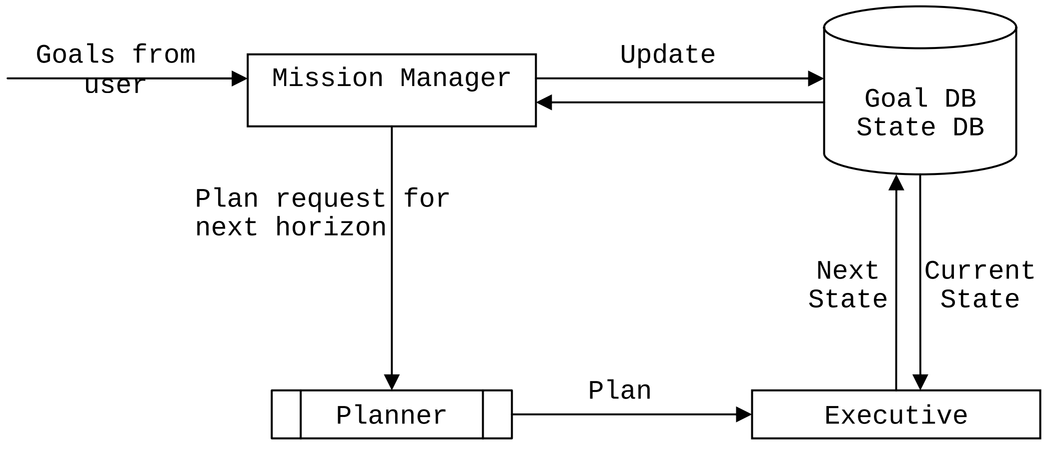

Figure 3.1 Flow Diagram of the Prototype

3.1 Mission Manager (MM):

Mission manager gets the input from the user. The user enters the goals as and when they crop up. The MM maintains these goals in the goal database, formulates the plan request for the next planning horizon and passes it onto the planner.

The plan request consists of the initial state (the state of the system at the start of planning) and the final state (the goal state to be achieved). It also updates the goal status if the plan is feasible. If it is not feasible, it decides on which goals to be neglected based on the priority and updates the goal status accordingly. For this purpose it maintains a priority queue containing the goals sorted in their order of their priority. Figure 3.2 gives a more detailed description of the prototype.

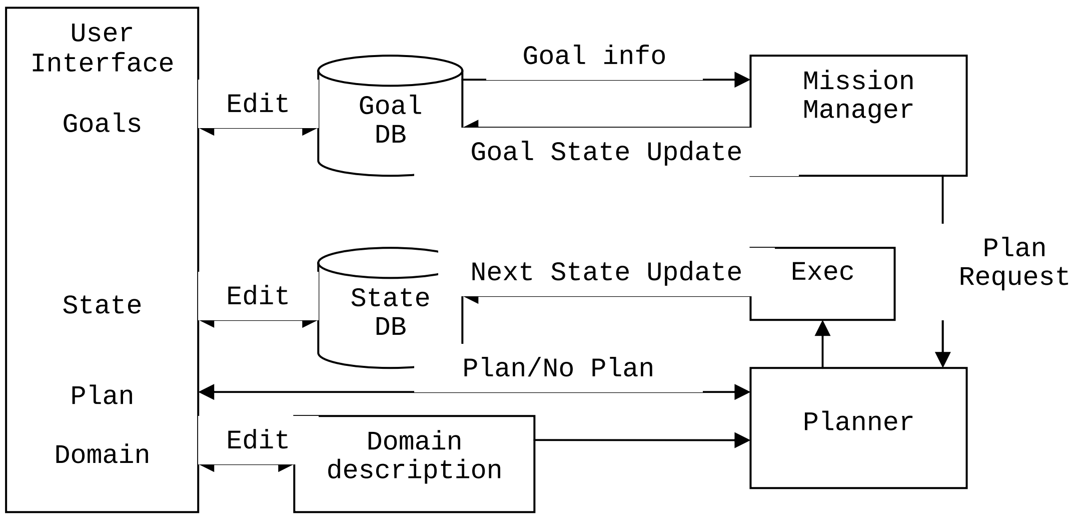

Figure 3.2 Prototype: Block Diagram

3.2 Planner:

The planner used in the prototype is the Sapa planner [2] developed by the Arizona State University. The working of Sapa planner has been briefly described in Section 4.3. The planner uses a domain description file which defines the different actions that are possible in the domain. An action is described in terms of preconditions and effects. The preconditions are the conditions which are to be satisfied before the action is applied. Once the action is applied, the predicates, which represent the effects of that action, become true. In case of any delete effects, the predicates, which go contrary to them, are deleted. The planner generates a plan if possible and gives the plan to the executive. If the plan cannot be generated it reports back to the MM.

3.3 Executive:

The executive is assumed to be perfect i.e., once the plan is given to the executive, the plan will be executed perfectly without any problem. So, there is no need for a Mode Identification and Recovery module in this system. It takes the feasible plan from the planner and calculates the next state. It then updates the state database with the next state information.

3.4 State Database:

The state database maintains the current state information of the system. It stores the set of predicates that are true in the current state and the information on all the resources. The information on resources are the availability of the resource (enabled or disabled), the type of the resource (shareable or exclusive) and the current value of the resource.

3.5 Goal Database

The goal database is used to store the goals that are to be achieved and other details about them. It stores the start time of each goal and the deadline before which the goal is to be achieved. The status of each goal is also stored in this database. The status can be accomplished, not accomplished or deleted (as it is not possible to achieve it before deadline).

3.6 User Interface

The User Interface component provides user the facilities to

1. Update the domain
2. Update the Current State (for testing purposes)
3. Update the goal set and
4. Verify the plan

The following section provides a brief description of the technologies demonstrated in the prototype.

4. AI planning:

Given an existing state of a system and a desired goal state the task of planning is the selection and sequencing of the set of actions, from the available set of actions, which when executed will move the system to the desired state. Depending upon the context, the goal state could be the positioning or orientation of the system; it could be an internal state from the health maintenance point of view; or it could be directly related to a task that is being undertaken.

4.1 Classical AI Planning:

Classical planning is based on the following assumptions

- There is complete knowledge of the initial state.
- The actions are deterministic, their effects are completely specified, and they do not modify the set of objects in the world, thus they preserve completeness of knowledge. (Closed World)
- The goal is a property of an individual state
- All actions take an equal amount of time i.e. unit time

4.2 Planning: issues concerning this domain

- Handling durative actions i.e. imaging, downloading
- Handling "real" start and end times
- Planning within "real" time constraints
- Handling constraints between activities and devices
  - e.g. attitude correction vs. image acquisition
- Handling commands coming in continuously and randomly
  - Plan for as much as you can.
  - Some of the goals/actions are periodic in nature
  - Issues of priority and interrupts
  - Concurrent planning and execution

4.3 Planner:

Sapa is a forward state space metric temporal planner [2], which handles durative actions (both dynamic and static durations). Classical planners assume the actions to be instantaneous which is not true in real world domains such as satellite domain. It also handles goals with deadlines and temporal constraints. Some of the issues involved in metric temporal planner are discussed below. Metric temporal planner has to handle additional constraints (other than logical constraints) involving time and functions which represent resource values. It tends to have a larger search space when compared to classical planners.

The search algorithm used in Sapa is the forward state space search. A state queue is maintained which contains all the states visited so far, sorted according to the heuristic function which measures the difficulty of reaching the goal state from the current state. It initially contains the initial state alone. It chooses the first state from the queue and applies all applicable actions to it. The resultant states are all stored in the queue. This is repeated again and again until the goal state (solution found) is reached or the state queue becomes empty (no solution).

Each state is tuple $S = (P, M, \Pi, Q, t)$ where $P=(p_i, t_i)$ is a set of predicates which were last achieved at $t_i<t$. M represents the set of values for all continuous functions, which may change over the course of planning. Functions are used to represent the value of metric resources and other continuous values such as amount of free memory space available (satellite domain). Π is a set of persistent conditions, which are to be maintained true for a particular period of time. Q is an event queue containing a set of updates along with the time at which they are scheduled to occur. t is the time at which this state is achieved.

Sapa takes the domain file and problem file in PDDL 2.1 (Planning Domain Definition Language) [4] format. The actions in planning problems with resource and temporal constraints are not instantaneous but have durations. Each action has a start time, duration and end time. The duration of an action can be statically defined or dynamically decided at the time of execution. An action can have preconditions that can be instantaneously true or required to be true for a particular duration which is less than or equal to the duration of the action. The effects of an action will be instantaneous either at the starting time or ending time of action. An example domain description file is shown in Figure 4.1. Figure 4.2 shows an example problem file and the plan given for it by the planner.

```
(define (domain satellite)
    (:requirements :strips :equality :typing :fluents :durative-actions)
    (:types satellite direction instrument mode)
    (:predicates
          (on_board ?i - instrument ?s - satellite)
          (supports ?i - instrument ?m - mode)
          (pointing ?s - satellite ?d - direction)
          (power_avail ?s - satellite)
          (power_on ?i - instrument)
          (calibrated ?i - instrument)
          (have_image ?d - direction ?m - mode)
          (calibration_target ?i - instrument ?d - direction)
          (sun_avail ?i - instrument)
    )
    (:functions (slew_time ?a ?b - direction)
           (calibration_time ?a - instrument ?d - direction)
           (data_capacity ?s - satellite)
           (data ?d - direction ?m - mode)
           (charge ?i - instrument)
           (data-stored)
    )
    (:durative-action turn_to
        :parameters (?s - satellite ?d_new - direction ?d_prev – direction)
        :duration (= ?duration (slew_time ?d_prev ?d_new))
        :condition (and (at start (pointing ?s ?d_prev))
                       (over all (not (= ?d_new ?d_prev)))
                   )
        :effect (and  (at end (pointing ?s ?d_new))
                      (at start (not (pointing ?s ?d_prev))
               )
     )
  )
```

Figure 4.1: An example domain description file for satellite domain in PDDL 2.1 format

```
Problem:
 1:(define (problem strips-sat-x-1)
 2:    (:domain satellite)
 3:    (:objects
 4:          insat - satellite
 5:          battery - instrument
 6:          camera - instrument
 7:          image - mode
 8:          thruster - instrument
 9:          Chennai - direction
10:          Bangalore - direction
11:          Hassan - direction
12:    )
13:    (:init
14:          (supports camera image)
15:          (= (charge battery) 100)
16:          (calibration_target camera Hassan)
17:          (= (calibration_time Camera Hassan) 5.9)
18:          (on_board camera insat)
19:          (sun_avail battery)
20:          (on_board battery insat)
21:          (on_board thruster insat)
22:          (power_on thruster)
23:          (power_avail insat)
24:          (pointing insat Chennai)
25:          (= (data_capacity insat) 1000)
26:          (= (data Chennai image) 500)
27:          (= (data Bangalore image) 320)
28:          (= (data Hassan image) 180)
29:          (= (slew_time Chennai Bangalore) 18.17)
30:          (= (slew_time Chennai Hassan) 38.17)
31:          (= (slew_time Bangalore Chennai) 18.17)
32:          (= (slew_time Bangalore Hassan) 28.04)
33:          (= (slew_time Hassan Bangalore) 28.04)
34:          (= (slew_time Hassan Chennai) 38.17)
34:          (= (data-stored) 0)
35:    )
36:    (:goal (and
37:                (have_image Bangalore image)-74
38:                (have_image Hassan image)-152
39:                (have_image Chennai image)-122
40:          )
41:    )
42:    (:metric minimize (total-time))
43:)

Solution:
       0.01: (switch_on camera insat) [2.0]
       0.01: (turn_to insat hassan chennai) [38.17]
       38.189995: (calibrate insat camera hassan) [5.9]
       38.199993: (turn_to insat bangalore hassan) [28.04]
       66.25: (take_image insat bangalore camera battery image) [7.0]
       73.26: (turn_to insat chennai bangalore) [18.17]
       73.26: (charge battery) [10.0]
       91.44: (take_image insat chennai camera battery image) [7.0]
       98.450005: (turn_to insat hassan chennai) [38.17]
       136.62999: (take_image insat hassan camera battery image) [7.0]
```

Figure 4.2: An example problem file and its solution in PDDL 2.1 format in the satellite domain

The example in Figure 4.1 defines the satellite domain. The first line defines the name of the domain. The next line gives the requirements used in this domain. “durative-actions” says that durative actions have been used in this definition. Line number 3 defines the different types of objects that are used in the domain. A set of predicates in the satellite domain had been listed in line numbers 4 to 14. The predicate in line number 7 defines the direction in which the satellite is pointing to. ?s and ?d represent the variables representing the types satellite and direction respectively. Functions are used to represent the value of metric resources and the value of other varying parameters of the system. The function in line 20 gives the amount of data stored in the satellite. One action turn_to has been shown in the example. This action is used to represent the act of turning the satellite from one direction to another. The parameters for this action are satellite and the two directions (previous and new). The direction of the action is obtained from slew_time function. The preconditions listed under the conditions say that the satellite doesn’t point to the previous direction at the start of the action and previous direction is not same as the new direction throughout the action. The effect of this action is that satellite ceases to point to the previous direction at the start of the action and points to the new direction at the end of the action. The model contains several other actions such as switch_on, take_image, calibrate and charge which can be similarly defined.

The first line in Figure 4.2 defines the name of the problem. The next line states the domain in which this problem is defined. Lines 3 to 12 list a set of objects used in this problem along with their types. The set of predicates, which are true initially, and the values of different functions are listed in lines 13 to 35. Line 24 says that initially the satellite is pointing to Chennai. The list of predicates, which are to be made true in order to achieve the goals, are listed in lines 36 to 41 along with their deadlines. Line 42 defines the metric which is to be minimized, time or cost. The solution gives the set of actions along with their start time and duration, given at the beginning and at the end of the each line respectively.

5. Conclusion

There is a definite need for automation of the LEO satellite operations. This paper investigates an approach towards application of autonomous agent technologies on the LEO satellite payload operations based on the DS-I remote agent architecture. It describes a system which demonstrates continuous planning.

6. References

[1] B. Pell, S. Sawyer, D. Bernard, N. Muscettola, and B. Smith "Mission Operations with an Autonomous Agent" in *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

[2] Minh B. Do, and Subbarao Kambhampati “Sapa: A Scalable Multi-objective Heuristic Metric Temporal Planner” in *Journal of Artificial Intelligence Research* (to appear).

[3] N.Muscettola, P.Nayak, B.Pell, and B.C.Williams “Remote Agent: to go boldly where no AI system has gone before” *Artificial Intelligence*, vol. 103, no. 1-2, pp. 5-48, 1998.

[4] Fox, M. and Long, D. “PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains” Technical Report, 2001.