Intelligent Rover Execution for Detecting Life in the Atacama Desert

Vijayakumar Baskaran[‡], Nicola Muscettola[†], David Rijsman[§], Chris Plaunt[±], Chuck Fry[‡]

[‡] QSS Group Inc, NASA Ames Research Center, Moffett Field, CA 94035
[†] Lockheed Martin Corporation, Sunnyvale, CA
[§] Mission Critical Technology, NASA Ames Research Center, Moffett Field, CA 94035
[±] NASA Ames Research Center, Moffett Field, CA 94035
{bvk,rijsman,cplaunt,cfry}@email.arc.nasa.gov, Nicola.Muscettola@lmco.com

[1]Abstract

On-board supervisory execution is crucial for the deployment of more capable and autonomous remote explorers. Planetary science is considering robotic explorers operating for long periods of time without ground supervision while interacting with a changing and often hostile environment. Effective and robust operations require on-board supervisory control with a high level of awareness of the principles of functioning of the environment and of the numerous internal subsystems that need to be coordinated. We describe an on-board rover executive that was deployed on a rover as past of the "Limits of Life in the Atacama Desert (LITA)" field campaign sponsored by the NASA ASTEP program. The executive was built using the Intelligent Distributed Execution Architecture (IDEA), an execution framework that uses model-based and plan-based supervisory control as its fundamental computational paradigm. We present the results of the third field experiment conducted in the Atacama desert (Chile) in August – October 2005.

Introduction

At the highest level, the goal of the LITA project [12] was to unambiguously detect life in one of the most arid locations on Earth as a precursor to a futuristic unmanned rover exploration on the Martian surface. For such a mission to succeed, it is imperative to develop a system that is capable of operating for extended periods of time, constantly collecting scientific data, analyzing them and also making real-time decisions with little or no input from human operators. It is also important to build a system that is robust to inevitable fault conditions. Autonomous recovery from faults is an absolute necessity for remote missions. Autonomy also plays a role in resource allocation and task prioritization at many different levels, starting from generating a plan of action to achieve the goals set out for the day to coordinating the activities of several instruments on board that will be vying for resources.

In this paper, we address one of the key components of an autonomous system, namely, the execution system. Many execution systems have been used to control complex systems ranging from robots to spacecraft. Besides differences inherent in the software architecture, execution systems can be categorized by whether they are coupled with a planner and also by the methodology used to represent domain knowledge. Execution systems such as PRS [5, 6] provide a real time response by choosing an appropriate set of plans from a library of procedures during each step of the execution cycle without a planner in the loop. On the other hand, there are also systems where the execution engine works in conjunction with a planner (internal, external or both). Examples of such systems are Apex [4], CRL [2,10], PROPEL [9], SCL [3], Titan [13]. A survey of different planning and scheduling technologies is given in [8]. IDEA (Intelligent Distributed Execution Architecture) provides a framework for an execution system where the planner operates at the core of the execution engine. A detailed description of IDEA can be found in [1].

There are four top-level components in an IDEA-based system: the domain model, plan database, an internal reactive planner and a scheduler. A domain model describes patterns of constraints between time intervals representing concurrent transitions between states of the controlled subsystem and the environment in which they operate. We call such a time interval a *token* and a collection of non-overlapping tokens representing mutually exclusive states is a *timeline*. The plan database connects all tokens and timelines by explicit constraints into a single constraint network. The constraint-based planning technology we use is EUROPA [7]. The reactive planner has the responsibility for generating a locally consistent plan that is executable, by taking into account the current state of the data base and any asynchronous external event that might have taken place.

This paper describes the Rover Executive (RE) and how the nominal executions and recoveries from off-nominal conditions were modeled. A plan is generated on the fly to prescribe valid courses of actions that satisfy the model's constraint patterns. The executive simply sends commands

to subsystems and receives message from sensors and subsystems strictly following the plan. If the plan is incomplete or violates the actual conditions of execution (e.g., sensory messages) then it is extended or repaired. The same occurs if new goals are communicated to the rover.

Scope of the Rover Executive (RE)

There were several responsibilities levied upon the Rover Executive (RE) in the LITA project. Foremost was to execute a nominal plan consisting of navigational and scientific activities laid out for the day. Goals were specified by scientists working in a remote location (Pittsburgh, PA) that were then up linked to the rover operating in the Atacama desert via a satellite. Examples of high-level goals are: commanding the rover to move to a specific location, approach an area of interest and gather scientific data or deploy a plow to dig a shallow trench to collect information about sub-surface elements. The RE was also responsible for handling faults by coordinating the recovery process, the most important of which was to quickly stop the rover before it puts itself in harm's way.

We implemented the RE using IDEA and deployed it in the Atacama desert as part of the 2005 field campaign between August and October. The core of IDEA provided the execution framework and also the necessary communication services while the domain declarative model captured the rover-specific behavior. Domain models were developed in consultation with system
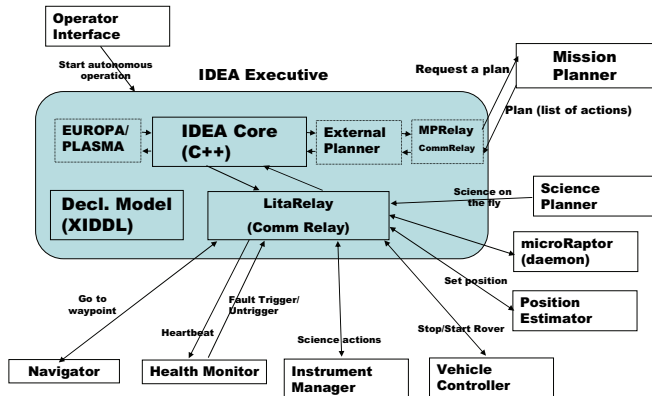

Figure 1 Interaction of the RE with other software modules

engineers who were familiar with the rover hardware. The RE ran on board the rover on a processor dedicated for autonomy. In the following sections we present the RE in greater detail and its interaction with other rover sub-systems. We then provide a description of faults that were handled, details of the declarative model that captures the nominal and fault recovery scenarios and results of the field experiment.

Overall Schematic

Interaction between the RE and the rest of the rover software is illustrated in Figure 1. After all the processes start up and reach a steady state, a human operator is responsible for issuing a command to commence the autonomous activity for the day via the Operator Interface. The mission specification up linked from scientists prior to the operator signaling the start of daily activity is then forwarded without any modification by the RE to the Mission Planner. The RE treats the Mission Planner as an external entity and exchanges no information with it except sending plan requests and subsequently receiving the appropriate response. IDEA's External Planner module requests and then translates the plan received from the Mission Planner into a local representation. The External Planner interacts with the Mission Planner through a communication relay (MPRelay) which publishes the appropriate sequence of messages. The plan returned by the Mission Planner is a series of actions that the rover is expected to execute so as to satisfy the goals laid out by the scientists for the entire day. For example, the Mission Planner fills in waypoints that are approximately 30 meters apart between the points of interest listed in the mission specification. The end times of each of the actions (and therefore the start times of the successive actions) specified in the plan are time intervals thus making the plan a flexible one.

When a complete plan arrives, the RE first loads it into the EUROPA plan database. This is performed by the Goal Loader, which is a part of the External Planner. The actions that make up the plan are translated into tokens that are then inserted into empty slots on a single goal timeline in the same chronological order in which actions appear in the generated plan. The final step of the goal-loading procedure is to activate deliberative planning wherein constraints consistent with compatibilities specified in the domain description model are enforced via propagations in the plan database. Since we already receive a full plan from the Mission Planner, the deliberative planning does not in this case have to fill in any gaps. It merely does a temporal and constraint network consistency check to ensure that the plan generated by the Mission Planner is indeed a valid one.

Upon successful completion of the deliberative planning step, the RE commences the first activity in the plan either as soon as the External Planner finishes its task of loading the plan in the database or at the earliest start time of the first activity, whichever of the two events is later. For every drive action that appears in the plan, the RE sends a waypoint to the Navigator and expects a notification when the rover reaches the intended destination. Similarly for science actions, the RE directs its commands to the Instrument Manager.

During nominal operation, the RE executes each and every action in the order in which it appears in the plan. In

addition, the RE also makes sure that all actions start and finish within the time bounds stipulated by the Mission Planner. As can be expected, the behavior is different when a fault occurs. Some of the faults are detected by the Health Monitor which, by keeping track of heart beats posted by different modules on a periodic basis, can infer whether a particular module is still operating within normal limits. The RE is inherently capable of detecting faults that are temporal in nature such as actions completing before (or after) a stipulated time interval. Details of fault recovery steps coordinated by the RE are given in subsequent sections. Any fault that has the effect of putting at risk the completion of all the actions in a timely manner results in the RE requesting the Mission Planner to generate a new plan. When such a re-plan request is made, the Mission Planner is informed of the list of unfinished goals, the current time and position of the rover.

The RE also communicates directly with the Vehicle Controller to set or clear the software emergency stop. Setting the software emergency stop prevents the Vehicle Controller from sending control commands to the micro-controller that actuates the wheels of the rover. The RE also has the responsibility for broadcasting the initial position of the RE to the Position Estimator.

The RE supports scenarios involving opportunistic science (science-on-the-fly). A Science Planner runs on board and gets activated whenever the Instrument Manager is asked to perform a planned science activity. The Science Planner sends a message to interrupt the RE to make allowances for any follow-on activity that it thinks might be necessary to explore the terrain for signs of life. The RE reacts by suspending the planned end time of the science action that is currently being executed and waiting for a signal to be issued by the Instrument Manager marking the completion of the follow-on action.

All communication between processes was achieved using the publish-subscribe paradigm implemented in IPC [11].

Fault Handling

Except for faults that are temporal in nature, the RE is informed of the occurrence (triggering) or the clearing (un-triggering) of faults by the Health Monitor. After being informed that a particular fault has occurred, the RE coordinates the necessary steps to recover from it. In this paper, we will not discuss strategies the Health Monitor uses to detect faults, but focus on the recovery actions coordinated by the RE. The RE deployed in the Atacama desert was capable of coordinating the recovery actions for 10 different types of faults. The causes and recovery steps for each of the faults are discussed below.

Early Completion of Actions: This fault is detected by the RE and is triggered whenever actions are completed before their earliest end time stipulated in the plan. An inconsistency is triggered in the temporal network when an action token (DriveAction or ScienceAction) is forced to terminate before the allowed earliest end time by the arrival of a completion status from an external sub-system (Navigator or Instrument Manager resp.). The recovery action is to issue a re-plan command to the Mission Planner.

Timeout (late completion) of Actions: Similar to the early completion fault, this is also detected by the RE when an action has not completed by the latest end time. In other words, the completion status of an action token (DriveAction or ScienceAction) had not arrived from the external sub-system (Navigator or Instrument Manager resp.) before the scheduled latest end time. The recovery action is to first send a command to cancel the action that caused the failure and then issue a re-plan request to the Mission Planner.

Process Crashes: These faults are triggered by the Health Monitor when the Navigator, Position Estimator or the Vehicle Controller crashes (stop sending heart beats). The recovery is performed in multiple steps. In all three cases, a software emergency stop command is issued right away to make sure that the rover does not get into a hazardous situation. In addition, the rover is put into a safeguarded mode. Then we attempt to restart the process that crashed by sending a request to a daemon that runs in the background. This daemon, called *microRaptor*, is capable of starting and stopping processes with the appropriate command line options specified in a configuration file. When a process gets restarted successfully, i.e., when the RE receives a fault un-trigger message from the Health Monitor, the rover is switched to the autonomous mode and the software emergency stop is cleared. In the case of Navigator recovery it may also be necessary for the RE to resend the waypoint which the rover was heading towards prior to the crash. Similarly, in the case of a Position Estimator Crash recovery, it is necessary for the RE to send the position of the rover prior to the crash.

No Arc Found Failure: This fault is triggered by the Health Monitor whenever the Navigator is not able to find a clear path ahead while the rover is moving towards a waypoint. The responsibility of the RE is to send a command to start the recovery action. In response to this command, the Navigator performs a pre-determined course of action, which is to reverse the rover. If no arcs are visible even after the recovery maneuver, the Health Monitor broadcasts another message to indicate that the fault is still triggered. In response, the RE sends the recovery command again. This cycle is repeated up to three times, after which the control is handed over to the operator. The RE does not actually tell what recovery action to take, but just commands the Navigator to take one.

Pose Uncertainty: This fault is triggered by the Health Monitor when the error in estimation of the current position of the rover exceeds a certain pre-defined threshold. The recovery action is for the RE to just stop the rover until the fault gets cleared. Stopping the rover gives

the position estimator an opportunity to refine and correct its estimates by integrating data gathered from the sun-tracker.

Not On Map Fault: This fault is triggered when the rover veers out of the boundary of an environment map maintained by the Navigator. The recourse from this fault is for the RE to cancel the waypoint that was previously issued and then request the Mission Planner to send a new plan given the current time and position of the rover.

Low Battery Power: This fault is triggered by the Health Monitor when the battery voltage falls below a certain threshold. The recovery action is to stop the rover and let the battery get recharged using solar energy. The fault will get un-triggered when the battery voltage reaches the desired level.

Domain Model

The domain specific information of the RE is captured in the declarative model that gets loaded during the initialization process. It is written in a language we have developed called XIDDL. XIDDL is an XML-based domain description language and has the semantics to define timelines, tokens and constraints (also called compatibilities) that relate tokens within and across timelines. To start the discussion on the model, we'll first look at the nominal mode of operation and then address the more involved case of fault recovery. There are three main timelines that form the backbone of the model. The timeline MissionExecutive_SV, on which the Goal Loader inserts action tokens received from the Mission Planner is a goal timeline. Two executable timelines, RoverMobilty_SV and InstrumentManager_SV get populated during the course of plan execution. Insertion of tokens in the two executable timelines results in commands being broadcast to the Navigator and Instrument Manager modules, respectively.

Every DriveAction token in the MissionExecutive_SV activates a GoToWaypoint token in RoverMobility_SV. Activation of the GoToWaypoint token is achieved by the "starts" compatibility in the model. The exact syntax used in the model to achieve this is;

```
<define_compatibility>
  <master type="single">
    <class>RoverExecutive_Class</class>
    <attr>MissionExecutive_SV</attr>
    <pred>DriveAction</pred>
  </master>
  <subgoals>
    <starts type="single">
      <class>RoverExecutive_Class</class>
      <attr>RoverMobility_SV</attr>
      <pred>GoToWaypoint</pred>
        <constraint name="eq">
          <arg>x</arg>
          <master>x</master>
```

```
      </constraint>
      <constraint name="eq">
        <arg>y</arg>
        <master>y</master>
      </constraint>
      <constraint name="eq">
        <arg>z</arg>
        <master>z</master>
      </constraint>
    </starts>
  </subgoals>
</define_compatibility>
```

where, [*x*, *y*, *z*] is the waypoint in ECEF (Earth Centered Earth Fixed) coordinates towards which the rover is required to move. A similar constraint is defined between a ScienceAction token in MissionExecutive_SV and InstrumentManager_SV. Start of the GoToWaypoint token results in an IPC command consisting of the [*x*, *y*, *z*] coordinates being published to the Navigator. After the rover reaches the waypoint, the Navigator publishes a completion message that is handled by the RE. Confirmation that the rover has reached a waypoint (i.e., the return status variable being set to "OK") results in the termination of the GoToWaypoint token that is currently executing. GoToWaypoint then terminates the goal token DriveAction via the "ends" compatibility defined in the model as shown below.

```
<define_compatibility>
  <master type="single">
    <class>RoverExecutive_Class</class>
    <attr>RoverMobility_SV</attr>
    <pred>GoToWaypoint</pred>
    <guard name="eq">
      <arg>status</arg>
      <value>OK</value>
    </guard>
    <guard name="eq">
      <arg>statusFlag</arg>
      <value>True</value>
    </guard>
  </master>
  <subgoals>
    <ends type="single">
      <class>RoverExecutive_Class</class>
      <attr>MissionExecutive_SV</attr>
      <pred>DriveAction</pred>
        <constraint name="eq">
          <arg>rStatus</arg>
          <value>OK</value>
        </constraint>
        <constraint name="eq">
          <arg>statusFlag</arg>
          <value>True</value>
        </constraint>
    </ends>
  </subgoals>
</define_compatbility>
```
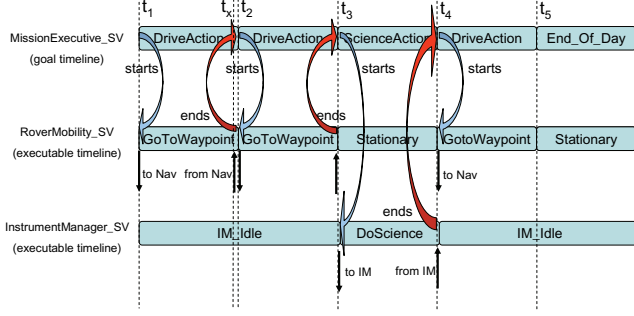
Figure 2 shows the interaction between the three timelines that participate in the nominal scenario described above.

Figure 2 Nominal scenario

A simple plan consisting of three DriveActions and a ScienceAction is inserted on the MissionExecutive_SV timeline in the RE's database. There is also a dummy marker token called the End_Of_Day which signifies the end of the plan. The first DriveAction is dispatched at time t_1. This results in the activation of the GoToWaypoint token on the RoverMobility_SV timeline, which publishes a command to the Navigator with the ECEF location of the waypoint. When the RE receives a notification from the Navigator that the rover has indeed reached the destination at time t_x, the executable token GoToWaypoint gets terminated which then ends the DriveAction goal token. The next goal action is then dispatched at time t_2. The time difference between t_2 and t_x will depend on the latency[2] of the RE. Upon completion of the second drive action, a ScienceAction goal token is dispatched at time t_3. This results in the insertion of a DoScience token on the InstrumentManager_SV timeline in a manner very similar to the interaction between the DriveAction and GoToWaypoint tokens. The execution process is flagged as complete when the End_Of_Day token gets dispatched at time t_5.

The model gets more complicated when fault recoveries are involved. For each of the ten faults handled by the RE, we have defined a triplet of timelines to intercept, register faults and coordinate the recovery steps. An executable timeline is used to intercept the message published by the Health Monitor and post the appropriate information in the database. Another timeline is used for registering the state of a particular fault at any given time, i.e, it tells us whether a fault is triggered or un-triggered. The third timeline is an executable one which consists of tokens that command the recovery sequence.

Recovery from the Navigator process crash during a rover traversal is illustrated in Figure 3. Let us consider the scenario where the rover is heading towards a waypoint and midway through the traversal, the Navigator crashes and ceases to send heart beats to the Health Monitor. The absence of heart beats will prompt the Health Monitor to trigger a Navigator fault message which the RE, say, receives at time t_1. For the sake of simplicity we'll ignore the processing delay between the time the notification arrives and the time when the RE commences the fault recovery procedure. A message to trigger the fault is posted to the database via a status variable of an executable token called NavCrashFault in FaultNotification_SV. In the domain model, we have specified compatibilities to be enforced when the status variable of NavCrashFault is bound to a particular state, in this case a fault being triggered. As can be seen in Figure 3, at time t_1, the FaultMode_SV transitions from a NoFault state to a Fault state. In addition, FaultRecovery_SV shows the activation of a token SStopSet which results in a software emergency stop message being published to the Vehicle Controller.
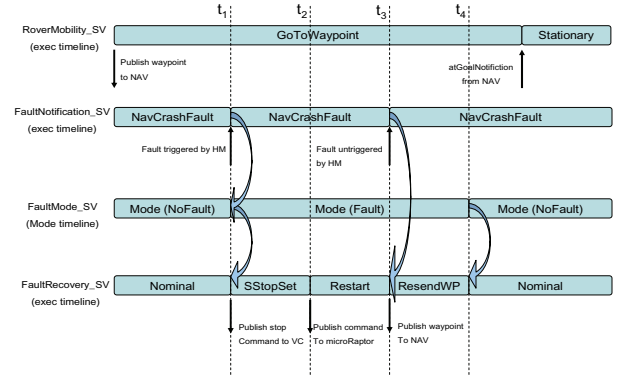

Figure 3 Navigator process crash recovery

After a pre-specified duration of one latency, at time t_2, a command to restart the Navigator is issued by the RE to the microRaptor daemon as a result of the Restart token being activated on FaultRecovery_SV. After the Navigator comes up and starts sending stable heartbeats to the Health Monitor a message is broadcast to signal un-triggering of the fault at time t_3. The fault un-trigger message is also posted to the database via the status variable of the NavCrashFault token, as was done when the fault was initially triggered. The RE reacts to this by resending the waypoint coordinates towards which the rover was headed when the Navigator crashed. One latency after the RE resends the waypoint, the FaultMode_SV is reset to a NoFault state.

The example described above illustrates a scenario in which the recovery process completes successfully and, more importantly, before the latest end time of the drive action during which the fault occurred. Since there is still a possibility for the rover to reach its waypoint before the latest end time specified in the plan, it is not necessary to

interrupt the original plan that is being executed. However, if the drive action was scheduled to end at a time before the recovery process completed, it would have been necessary to discard the remainder of the original plan and request the Mission Planner for a new plan immediately after the recovery action completes.

Results

The RE was deployed on board the rover during the 2005 field campaign in the Atacama desert. Data from all the logs collected in the field were analyzed from the point of view of stability and robustness of the RE and also from the perspective of the state of integration with the other processes running on-board. As can be expected, an undertaking of this magnitude, where there are 15-20 different software processes competing for resources and working in coordination with each other, is bound to stretch the limits of software integration. In addition, there were several hardware components to be controlled and operated in one of the harshest of environments. All processes contributing to rover autonomy, such as the Rover Executive, Mission Planner, Science Planner and Science Observer ran on a dedicated Pentium 4, 2.2GHz processor. A second processor was dedicated to the rest of the rover software, such as the Navigator, Position Estimator, Vehicle Controller, Health Monitor and various Instrument Controllers. As a measure of robustness, we collected information on the duration of operation of the RE. Obviously, an executive that operates uninterrupted for a long period of time bodes well for any autonomous system in general. Looking at the duration of operation of any process also sheds some light on the software integration process as a whole. We ran into several software startup and integration glitches that had to be resolved during field operations. This resulted in processes being started and killed manually in quick succession. We have also collected information on all the instances where the RE ceased to operate due to a fault of its own or forced to operate under conditions it wasn't equipped to handle. Reasons for RE malfunction have been summarized in the form of a histogram later in this section. We have also gathered data on the faults that were triggered during the entire course of the field operation.

Figure 4 shows a histogram of the duration of operation of the Rover Executive. The start and stop times of every single run of the RE were extracted from the log to calculate the duration of operation per run. This analysis sheds some light on the stability of the RE as well as software integration. Over the course of about six weeks of the field campaign, the RE was started a total of 322 times in conjunction with the full suite of processes onboard the rover. There were 145 instances where the RE run time was shorter than five minutes. Except for a total of 27 instances, the run was manually terminated by the operator to track down non-RE related problems. Of the 27 instances where the RE contributed to an aborted start, 9

instances can be narrowed down to problems inherent in the RE implementation. The remainder of the RE premature terminations were caused by corrupted input data, spurious messages being sent from other modules and similar protocol violations.

At the other end of the spectrum we had two instances where the rover executive performed uninterrupted for over four hours.
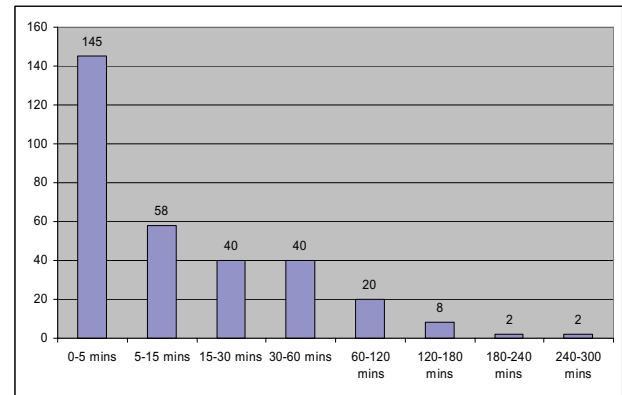

Figure 4 Duration of operation

Figure 5 is a histogram of the types of faults that were triggered during the entire course of the field operation. In each of the pairs of graphs, the bar on the left indicates the number of times a particular fault was triggered and the bar on the right indicates the number of times the rover executive successfully took the necessary step(s) to rectify the fault. The most commonly triggered fault was the NoArcFound fault, which the RE successfully handled 380 out of a total of 396 times (96% rate of success). On seven of the remaining 16 occasions, the NoArcFound fault was triggered three times consecutively and the RE went into a standby state as per the design and waited for the operator to intervene and clear the fault. This typically happened while negotiating a very rugged terrain, where backing up the rover was not sufficient to clear the fault.

The second most frequently triggered fault was the early completion of science action. The Mission Planner was pessimistic about the completion time of science actions, thereby allocating more time for such activities than what was actually required. The RE successfully coordinated recoveries for early completion of science action with a success rate of 100%.

The success rate of handling NotOnMap fault was not very high (only 58%). On three of the unsuccessful recovery cases, the RE received a NotOnMap during the initialization process at startup. This was the result of an inconsistency between the state of the RE and the Navigator. The problem was rectified during the course of

the field operation by requiring the RE to clear all waypoints cached in the Navigator before requesting the Mission Planner for a new plan.
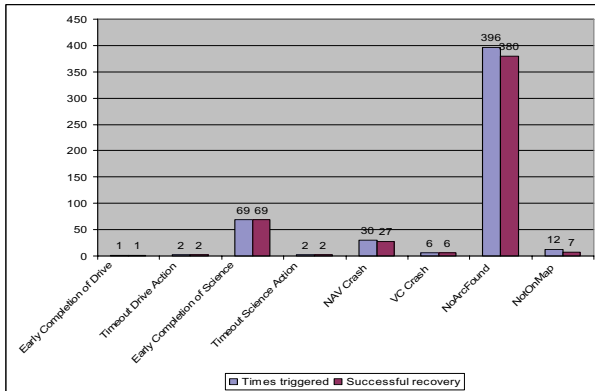

Figure 5 Faults triggered

There were 55 occasions out of 322 instances where the RE ceased to operate and Figure 6 identifies the range of causes. There were 14 instances where the RE terminated with a "controller_timeout" error. This typically happened when the RE was not able to complete the reactive planning cycle within one latency (3 seconds) due to CPU starvation. When the RE succumbed to this situation, it would automatically be restarted by the microRaptor daemon. Therefore, a premature termination of the RE did not always mean the end of an autonomous run because mechanisms were put in place to restart the RE autonomously and submit the remaining set of goals to the Mission Planner without any human intervention.

A plan parsing bug in the Goal Loader module resulted in the RE crashing on eight occasions. One particular plan returned from the Mission Planner consisting of the character "#" in the comment field exposed a bug in the Goal Loader. Although there were eight instances of such a crash, they all took place in succession when the operator kept loading the same plan repeatedly before realizing the cause.

On seven occasions conflicting messages were published by the Navigator and the Health Monitor. Within a matter of milliseconds, the Health Monitor triggered a NoArcFound fault while the Navigator published a message that it had reached the goal. Since this scenario was not taken into account in the domain model, the RE was unable to resolve the conflict. The problem was eventually resolved by improving the interaction between the Navigator and the Health Monitor.

On 11 occasions the RE was handed an empty plan from the Mission Planner with only the End_Of_Day action which is a dummy marker that denotes the last action in a plan. The RE did not parse that special case properly but

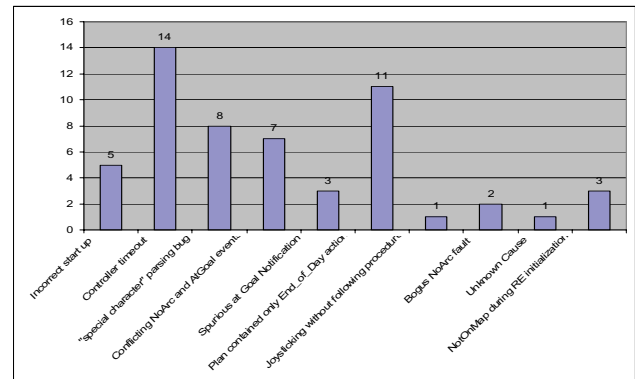that problem was resolved during the course of the field operations.

Figure 6 Causes for RE crashes

Conclusions

The 2005 LITA field experiment successfully demonstrated the usefulness of a model-based execution system like IDEA. The Rover Executive played its part in achieving the overall goals laid out for the project. Its presence in the loop aided during both the nominal and off-nominal situations, thereby reducing the burden on the mission operators.

There were, however, several issues that were brought to the forefront. The RE was found to be sluggish and wasn't quite realtime, considering it operated at a latency of three seconds. Several improvements are being actively explored to streamline the IDEA framework. Current simulated studies have shown that we are able operate the RE at a latency of one second.

References

1. Ashwanden, P,. Baskaran, V et al. *Unified Planning and Execution for Distributed Autonomous System Control*. AAAI Fall Symposium on Spacecraft Autonomy. Washington D.C. 2006.
2. Bresina, J.L. and Washington, R. *Robustness via Run-time Adaptation of Contingent Plans*. Proceedings of the AAAI Spring Symposium: Robust Autonomy, Stanford University, CA, 2001.

3. Chien, S,. Sherwood, R., Tran, D et al. *Using Autonomy Flight Software to Improve Science Return on Earth Observing One*. Journal of Aerospace Computing, Information, and Communication . April 2005 .

4. Freed, M. *Managing Multiple Tasks in Complex Dynamic Environments*. Proceedings of the National Conference on Artificial Intellignece. Madison WI, 1998

5. Georgeff, M and Lansky, A. *Procedural Knowledge*. Proceedings of the IEEE Special Issue on Knowledge Representation, Volume 74, pages 1383-1398, 1986.

6. Ingrand, F., Chatila., R, Alami, R and Robert, F. PRS: *A High Level Supervision and Control Language for Autonomous Mobile Robots*. IEEE ICRA, Minneapolis, MN. 1996.

7. Jonsson, A and Frank, J. *A Framework for Dynamic Constraint Reasoning Using Procedural Constraints*. 5th Internaltional Conference on Principles and Practices of Constraint Programming, 1999.

8. Kortenkamp, D, *A Day in an Astronaut's Life: Reflections on Advanced Planning and Scheduling Technology*, in IEEE Intelligent Systems Magazine, April/March, 2003

9. Levinson, R. *Unified Planning and Execution for Autonomous Software Repair*. Workshop on Plan Execution: A Reality Check. ICAPS 2005.

10. Pederson, L., Smith, D., et.al. *Mission Planning and Target tracking for Autonomous Instrument Placement*. IEEE Aerospace Conference, 2005.

11. Simmons, R and James, d. *Inter-Process Communication* v3.4, Carnegie Mellon University, 2001.

12. Wettergreen, D., Cabrol, N., Baskaran, V et al. *Second Experiments In the Robotic Investigation of Life in the Atacama Desert*. Proceedings of iSAIRAS Conference. Germany, 2005.

13. Williams, B.C., Ingham, et. al. *Model-Based Programming for Fault Aware Systems*. AI Magazine, vol 24., no.4, 61-75. 2004.