Programmable Deep Space Autonomy: The First 25 Years

G. Mark Brown
Avionics Systems Engineering Section
Jet Propulsion Laboratory
April 9, 1997

Acknowledgements: George Chen, Chris Jones, Pete Kobele, Wayne Kohl, Rob Manning, John Slonski, Bob Rasmussen
* 

Outline

- Autonomy Drivers for Deep Space Missions
- Autonomy Objectives for Deep Space Missions
- Fundamental Autonomy for Deep Space Missions
- The Evolution of Computing Resources
- The Evolution of Autonomous Tasks
- Past and Present Shortcomings
- Reflections on the Future
* Autonomy Drivers for Deep Space Missions

◆ The operations team is not in constant contact with the spacecraft
  ◆ The Deep Space Network (DSN) is a shared resource
  ◆ DSN tracking is expensive
  ◆ Spacecraft may be “out of view” for extended periods

◆ Even if constant contact was possible, the round-trip light time makes many types of ground-in-the-loop activities inefficient, others risky or impossible
Autonomy Objectives for Deep Space Missions

- A fundamental objective of any project is to maximize its Return On Investment (ROI).

- Derived autonomy objectives for deep space missions that contribute to increased "return" are:
  - Survive failures
  - Complete time-critical activities following failures or other anomalies
  - Protect valuable data until it can be returned to Earth
  - Conserve life-limiting spacecraft resources

- Derived autonomy objectives for deep space missions that contribute to decreased "investment" are:
  - Decrease the use of DSN resources
  - Decrease the workload of the mission operations team
* Fundamental Autonomy for Deep Space Missions

- Time-driven execution of stored sequences
  - The ground uplinks large sets of time-tagged commands for later execution
  - Some commands activate spacecraft “macros”
  - The ground can “tweak” command parameters prior to execution
  - Real-time commands can be executed concurrently

- State estimation and feedback control
  - Often limited to those states that cannot be controlled accurately enough by the ground

- Health monitoring and redundancy management
  - Often limited to the protection of “critical” functions and resources
* Fundamental Autonomy for Deep Space Missions

- Autonomous fault responses suspend stored sequences
  - Avoids any possible contention between the two command sources
  - Mitigates any possible errors in the sequence design
- Autonomous fault responses are compatible with a small number of restartable time-critical sequences
  - Time-critical sequences must use goal-oriented commands that can be issued repeatedly if necessary
  - If a time-critical sequence is terminated, it waits for autonomous responses to complete, and then starts again from the last mark point
  - Autonomous activity is logged in onboard memory, and then preserved for future ground inspection
- The ground has enable/disable control over autonomous functions, and can also change the threshold and persistence criteria that are used by autonomous functions
*The Evolution of Computing Resources*

<table>
<thead>
<tr>
<th>Spacecraft</th>
<th>Viking Orbiter</th>
<th>Voyager</th>
<th>Galileo</th>
<th>Mars Observer</th>
<th>Cassini</th>
<th>Mars Pathfinder</th>
</tr>
</thead>
</table>

| Programmable Memory (Mbytes) | 0.02 | 0.07 | 0.5 | 0.5 | 5 | 128* |
| Processing Speed (MIPS)      | (0.003) | 0.003 | 0.3 | 0.5 | 1 | 10 |
| Bulk Data Storage (Mbytes)   | 64 | 64 | 128 | 260 | 250 | 128* |
| Maximum Uplink Rate (bps)    | 4 | 16 | 32 | 500 | 500 | 500 |
| Maximum Downlink Rate (Kbps) | 16 | 115 | 134 | 85 | 249 | 249 |

* = shared memory
( ) = guesstimate

Over the past 25 years, the growth in onboard computing resources has greatly outpaced the growth in uplink and downlink bandwidth.
# The Evolution of Autonomous Tasks

<table>
<thead>
<tr>
<th>Spacecraft</th>
<th>Viking Orbiter</th>
<th>Voyager</th>
<th>Galileo</th>
<th>Mars Observer</th>
<th>Cassini</th>
<th>Mars Pathfinder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointing Control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Delta-V Control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Redundancy Management</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reset Recovery</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Stored Sequence Execution</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Critical Sequence Retry</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Inertial Vector Propagation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Attitude Constraint Enforcement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Guide Star Selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Temperature Control (Thermostat)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Command Macro Execution</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bit Error Detection/Correction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Event-Based Audit Trail Generation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Event-Based Telemetry Generation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Target Tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Feature Recognition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orbit Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Margin Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Science Planning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Science Data Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Science Telemetry Selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evolutionary Example: Redundancy Management

- Command Loss Monitor and Response
  - Objective is to prevent permanent loss of command reception capability
  - Voyager
    - Sets a timer to a user-controlled value each time a command is received
    - Timer expiration causes initial reconfiguration of equipment in the command path
    - Continued absence of commands causes additional periodic reconfiguration of equipment in the command path, eventually asserting all possible combinations
    - Reception of a valid command freezes the configuration and resets the timer
  - Galileo
    - Same as Voyager
  - Cassini
    - Same as Voyager, but continued absence of commands eventually causes an entire CDS string swap
  - The command loss timer must be carefully maintained by the operations team
    - "No-op" commands may need to be sent to keep the timer from expiring
    - The timer value may need to be decreased prior to planned time-critical uplinks
Evolutionary Example: Constraint Enforcement

- Sun-Relative Pointing Constraints for Science Instruments
  - Objective is to protect thermally sensitive elements from irreversible damage
  - Voyager
    - Does not estimate the Sun’s position, does not recognize any pointing constraints
    - “Dumb safing” of scan platform during autonomous fault responses
    - Boresights are protected by pointing them at a body-fixed calibration target
    - No recognition or enforcement of radiator constraints
  - Galileo
    - Estimates the Sun’s position, and recognizes some boresight constraints
    - “Smart safing” of scan platform if a pointing command violates constraints
    - No recognition or enforcement of radiator constraints
  - Cassini
    - No scan platform; instruments are rigidly mounted to the spacecraft
    - Estimates the Sun’s position, and recognizes both boresight and radiator constraints
    - Restricts attitude to constraint boundary if a pointing command violates constraints
    - Can recognize and enforce “timed” constraints
Evolutionary Example: Time-Critical Activity

- Orbit Insertion Burn
  - Objective is to achieve the desired orbit at all costs
  - Viking
    - Single engine, one burn start attempt
    - Delta-V direction and magnitude were fixed
    - Time-based command sequence
    - Two CDS strings issued commands in parallel, without any shared data
  - Galileo
    - Same as Viking, but the two CDS strings shared selected data
  - Cassini
    - Redundant engines, several burn start attempts if necessary
    - Autonomous pyro firings to unisolate the backup engine if necessary
    - Delta-V direction and magnitude are functions of time
    - Time-based command sequence
    - Backup CDS shadows the prime CDS, but does not issue any commands
    - Prime and backup CDS share selected data
Past and Present Shortcomings

- Pre-launch design and testing
  - Autonomy cost-benefit trades are difficult to make
  - Failure modes analyses are costly, incomplete, and often too late to capture in the autonomy design
  - Design decisions and philosophy are not easily documented
  - Design is difficult to validate
  - False alarms from health monitors impede testing

- Operations
  - False alarms are more prevalent than real detected failures, often causing undesired termination of the stored sequence
  - Recovery from stored sequence termination is time- and labor-intensive
  - Unforeseen scenarios almost always cause undesired behavior
Reflections on the Future

- Past deep space missions have flown only as much autonomy as they required. Many types of future missions (e.g. in situ experiments, coordinated formations) will require additional autonomy.
- Past deep space missions have flown only as much autonomy as their computing resources could accommodate. This constraint is rapidly disappearing.
- Past deep space missions have viewed autonomy as a necessary expense. Future missions may actually be able to save money via autonomy.
- Past deep space missions have employed “procedural” autonomy. These systems are not well-suited to handle circumstances that their designers did not foresee. Future missions will present even more unforeseen circumstances, requiring a departure from “procedural” autonomy.
- Deep space missions are already inheriting many of their autonomy requirements and approaches from their predecessors. The challenge for future missions is to increase the portability of autonomy implementations.