
A Typical Spacecraft Autonomy System

Martin Gomez

Johns Hopkins University Applied Physics Laboratory, 11100 Johns Hopkins Rd., Laurel, MD 20723-6099

MARTIN.GOMEZ@JHUAPL.EDU

Abstract

Over the last decade or so, the Johns Hopkins University Applied Physics Laboratory (JHU/APL) has designed, built, launched, and operated several interplanetary and Earth-orbiting spacecraft such as ACE, MSX, NEAR, FUSE, TIMED, and CONTOUR. Several others, such as MESSENGER, STEREO, and New Horizons, are presently under development. The autonomy system in these spacecraft is used primarily to handle anomalies, and secondarily to simplify routine operations. After a brief introduction to spaceflight concepts, this paper describes the autonomy system used by a typical JHU/APL spacecraft. The bulk of the autonomy is provided by a set of autonomy rules. Each rule performs a mathematical operation and a logical comparison, triggering a stored command sequence if the logical operation yields “true.”

1. Introduction

A spacecraft flying an interplanetary mission poses many difficult challenges for its designers. Most of these can be traced back to one source: very long communications distances. Distances of 1 AU (about 150 million km) are routine. Jupiter at 5 AU and Saturn at 9.5 AU from the Sun have each been visited. The New Horizons mission to Pluto and the Kuiper Belt will have to handle distances of over 30 AU. It takes the speed of light about 8 minutes to travel 1 AU, and useful communications requires a round trip. Thus even in the ideal case in which an anomaly occurs on a spacecraft while it's in contact with Mission Operations, at least 16 minutes will elapse (per AU) before the response arrives. Furthermore, contact is intermittent; the spacecraft must survive periods of one week or more with no ground contact, even in the face of failures. This paper describes the autonomy system that

JHU/APL deploys in its spacecraft to help mitigate these limitations.

2. Spacecraft 101

2.1 Communications

A challenging constraint is imposed by the antennas required for communicating at such distances. To receive a useful data rate at interplanetary distances (given the low transmitter power and small antennas that we can install on a spacecraft) requires enormous antennas on the ground. NASA's Deep Space Network uses 37-metre and 74-metre antennas, spaced approximately evenly in longitude around the Earth, to allow for round-the-clock communication with a spacecraft located anywhere on the celestial sphere. Unfortunately, the cost of such installations is such that few other than NASA can afford them. The antennas must be shared among many missions so continuous communications is not possible for any one mission. A fortunate mission makes do with one contact per day, many with less. A spacecraft that declares an emergency can monopolize the antennas for awhile, as can spacecraft traversing a critical phase of the mission, such as launch vehicle separation, a maneuver, an encounter, or a landing. The cost for DSN coverage is several thousand dollars per hour. Typical data rates (space-to-Earth) from 1 AU are on the order of 100 kbits/second. Uplink rates are typically less than 1000 bps.

2.2 The Importance of Attitude Control

Most science spacecraft are three-axis stabilized. In other words, their autopilots can point them anywhere in the sky and hold that orientation to a high tolerance. Simpler spacecraft are spin-stabilized. They rely on their gyroscopic stiffness to keep their spin axis fixed in space. Spin-stabilized spacecraft are able to point the spin axis anywhere on the celestial sphere, albeit not as precisely as their three-axis stabilized counterparts. However, since

science spacecraft often have to point imagers or particle detectors or other directional instruments, most science spacecraft are three-axis stabilized.

It must be emphasized that for both three-axis or spin-stabilized spacecraft, the mission's survival depends on the ability to point the spacecraft. Attitude control is what allows the science payload to be pointed at the target body, the directional antennas to be pointed at Earth, the thrusters to be pointed in the direction of the desired acceleration, and the solar panels to be pointed at the Sun. This latter requirement – pointing towards the Sun – is particularly important given the small batteries carried by most spacecraft. Since batteries are heavy, most spacecraft carry the smallest batteries they can get away with, typically sized to last for an hour or so, with a reduced electrical load. This has the effect of making spacecraft intolerant of protracted loss of attitude control. There are other reasons why loss of attitude control is a serious problem, such as loss of Earth communications, sunlight falling on portions of the spacecraft that can be damaged by it, or the inability to perform an essential maneuver. For these reasons, much of the autonomy and fault protection mechanisms we use are intended to detect and correct a loss of attitude control.

2.3 Spacecraft Commands

Most JHU/APL spacecraft have command vocabularies of a hundred or so different commands. Commands are sent through a reliable, layered protocol, not unlike the ISO/OSI stack familiar to local or wide area network designers. Commands may be executed immediately (also known as “in real time”) or stored on board to be executed at a prescribed time of day. Most of the commands are quite low-level. As with the primitives comprising a typical computer's instruction set, it takes many commands to do anything useful. One command might turn on a single relay, or set the downlink data rate, or tell the spacecraft to point in a specified direction. A very sophisticated ground system is used to send commands. It includes a database that translates command mnemonics into binary opcodes. Ground system designers are by necessity quite clever – they have to make up for the limitations that weight, power, and environmental constraints force upon the spacecraft designers. Data is received from the spacecraft in the form of packets, each containing data from one application on the spacecraft. For instance, a packet might contain science data from an instrument, or a memory dump from one of the spacecraft's computers, or engineering housekeeping data such as temperatures, voltages, and currents. The ground system translates the binary data into useable quantities in engineering units.

2.4 Data Storage

Since the spacecraft collects data even while it's out of contact with Earth, a recorder is carried – usually consisting of a gigabyte or more of solid state memory – to hold the data until it can be downlinked. Many of the commands the spacecraft “knows” are intended to manage the recorder, or to allocate the limited downlink bandwidth among the various competing uses. The bulk of the DSN passes are devoted to emptying the recorders of science data, naturally, but the first few minutes will usually be devoted to downlinking any urgent stored anomaly information.

Most of the time, then, an interplanetary spacecraft is on its own. A low-earth orbit (LEO) spacecraft faces a similar problem: only when passing over a ground station can it communicate. Most LEO spacecraft have a few (say, six) passes per day, for each ground station they can talk to. Each lasts five to twenty minutes, depending on altitude and pass geometry. These passes are not evenly spaced in time, so there can be twelve-hour periods without communication. Naturally, both LEO and interplanetary missions must contend with outages due to failures of ground equipment, failures of onboard communications equipment, or scheduling conflicts for ground-based assets.

It is worth highlighting the different tempo of LEO and interplanetary missions. LEO spacecraft usually gather data at a leisurely rate, over a few orbits, storing it onboard until it can be downlinked. When they pass over a ground station, the data is downlinked at a high rate, since the distances are at most a few thousand kilometers. An interplanetary spacecraft, on the other hand, is usually semi-dormant during its lengthy cruise phase. During its encounter, it briefly gathers data at a furious rate as it swings past the target body. It then spends weeks or months sending gigabits of data back to Earth through the occasional DSN pass.

2.5 Safe Mode

Thus spacecraft design teams take a many-pronged approach to fault protection. At the lowest level, if the bus voltage falls below a certain threshold, the spacecraft is rebooted by a hardware circuit. This causes most non-essential loads to be shut off in hardware, and usually restarts the computer with either a different software load, or in a simpler operating mode. For instance, if a thruster burn was in progress, a reboot might cause the burn to abort, might shut off all of the heaters, valves, and sensors required by the burn, and might cause the spacecraft to enter a “safe mode.” In the lowest safe mode, the spacecraft reverts to what it can sense with its most reliable sensors. The software won't assume it knows

where the Earth is, or where the Sun is, or what time it is, or what it's supposed to do next. It will simply turn its solar panels towards the Sun, lower its data rate to the lowest possible value, and wait for a command from Earth. Some spacecraft incorporate a fan-beam antenna so that by rotating about the Sun line, the beam will occasionally sweep past Earth.

3. The JHU/APL Autonomy Engine

3.1 Autonomy vs. Automation

Returning to the subject of autonomy, spacecraft designers use onboard autonomy as a tool for dealing with intermittent communications. In JHU/APL parlance, autonomy is distinct from automation. To illustrate the difference, consider that most of our modern spacecraft have attitude control systems (ACS) that ensure they point in the desired direction. These systems include devices that measure the rotation rate and orientation of the spacecraft as well as actuators to produce the torque required to keep the spacecraft properly oriented. A computer algorithm calculates the difference between the current attitude and the desired one, and commands the actuators accordingly. That is automation, not autonomy. Autonomy is used as a surrogate for a full-time Mission Operations team, which as described earlier, is not feasible. The autonomy system might tell the attitude control system where to point, or it might be used to detect a failure of the ACS. For example, the autonomy system might ask "Has the Sun been in the spacecraft's forward hemisphere for most of the last minute?" and reset the ACS algorithm if it hasn't.

Similarly, many modern spacecraft have automatic circuits to keep the battery charged, and to keep the solar panels operating at their optimum power point despite changes in sunlight and load. These are also automation. The autonomy system is used to detect low voltage, or to protect the battery against overcharging, or to detect and handle failures in the electrical power system.

3.2 Autonomy Rules

While some autonomy – a small fraction of the functionality -- is hard-coded in the flight software, this paper will concentrate on what we routinely refer to as "the autonomy engine," namely a run-time programmable set of if-then rules used to handle most anomalies. A typical JHU/APL spacecraft built in the last 5 or 10 years will have space for 512 autonomy rules. In a typical autonomy rule load consisting of 200 to 300 rules, roughly half are designed to detect the failure of a primary component and switch to its backup. Each rule consists of one or more arithmetic operations, a logical

comparison, and a command. For example, the following rule guards against having instrument A draw too much power, and switches to its backup, instrument B, if necessary:

```
if (current_drawn_by_instrument A * bus_voltage >
32 watts) then
    shut off instrument A
    turn on instrument B
    wait 10 seconds for instrument B to warm up
    enable rules that checks instrument B's power
    start recording telemetry for instrument B
endif
```

Autonomy rules, in short, are used essentially as a collection of smart circuit breakers, thermostats, and health checkers.

An autonomy rule has several states. As alluded to by the above example, they can be enabled (meaning they are evaluated and executed if their condition is satisfied) or they can be disabled (they are not checked or evaluated.) They can be write-protected or write-enabled, to prevent Mission Ops from inadvertently overwriting a rule. Autonomy rules also have maximum fire counts: after the rule has triggered (i.e.: had its condition satisfied and its command executed) a prescribed number of times, it is disabled. The user must clear the fire count before the rule can fire again. Also, in order to prevent a rule from reacting inappropriately to a transient condition, the user may specify how many times it must be satisfied, i.e. only if the condition is satisfied N times in M attempts should the command be executed. The variables N, M, and the maximum fire count are attributes of the rule which are set by the user when the rule is loaded onto the spacecraft's computer.

3.3 Computed Telemetry and Storage Variables

Note that the math performed by the above rule may be a feature of several similar rules. One can envision, for example, one rule to check if the power is too high and another to check if it's too low, each executing a different set of commands when satisfied. To avoid doing the math several times per second in several different rules, we implement a mechanism known as computed telemetry. The computed telemetry feature allows the user to specify a few mathematical operations and store the results on board, such as:

```
instrument_A_power =
current_drawn_by_instrument A * bus_voltage
```

Any rule wishing to compare instrument A's power to a threshold can just compare the results of this calculation, without having to perform the math itself. While it is a

trivial calculation in the example provided above, it is a useful shortcut for more complex operations.

In order to give the autonomy rule designer some storage to work with, JHU/APL spacecraft usually include a feature known as storage variables. Storage variables are simply run-time programmable regions of memory that autonomy rules can set or test. One typical use is to count the number of times that a certain event has been detected, across multiple rules. Since there is no other connection between individual rules, conditions that are detected by several rules can use a common storage variable to keep track of how many events they have collectively detected. A separate rule can then execute a command when the storage variable indicates that a specified count has been exceeded.

3.4 Macros

The commands executed when a rule is satisfied are worth expanding on. An autonomy rule can actually contain only one command, and it has to be a short one. Autonomy rules are stored in fixed-sized slots, and only a few words (3 to 6 32-bit words depending on the spacecraft) are allocated for the command. The way around this limitation is to have that command invoke a macro. Macros are variable-length lists of stored commands, typically between 10 and 100 commands long. They are logically distinct from rules. A macro may “belong” to a single autonomy rule, or may be invoked by several rules, and may be invoked by another macro. In the above example, if “shut off instrument B” were actually composed of many commands, it may be desirable to have all macros that shut off instrument B call a macro devoted to that purpose. After all, there may be several different reasons for shutting off a device, each detected by a different autonomy rule: it might draw too much power, or not enough, or be too hot, or too cold, or not be responding to requests from the computer, or it may be signaling an internal fault. Each of the rules that detects those conditions can invoke the same macro. Like the rules themselves, macros may be modified in flight.

3.5 Other Uses of Autonomy Rules

Designers don’t use autonomy rules merely to handle anomalies, but also to simplify routine tasks. Since autonomy rules can be disabled, they offer a convenient mechanism for implementing on/off/auto commands, i.e.: commands for which the normal operation is to allow an onboard algorithm to turn a device on or off, but which must provide for user override under some circumstances. A thermostat is a perfect example: one might implement a pair of rules which turn a heater on or off based on a temperature measurement. However, during ground test, or indeed during on-orbit troubleshooting, one might want

to force the heater to turn on, regardless of the temperature. By disabling the autonomy rule that turns it off, the user can send the command that turns the heater on and conduct his testing.

Another common use for autonomy rules is to implement an “autoexec.bat” feature. In this case, the rule is simply designed to fire a specified macro one time, with a condition that’s always true, i.e.:

```
if (true)
    execute macro 3
endif
```

This mechanism is used to initialize the spacecraft to a desired state after power-up. Note that the desired state may evolve as the spacecraft ages or the mission changes. This is a reminder of one of the other reasons for having an autonomy capability: the lack of satellite repair facilities near Jupiter! As radiation, thermal cycling, and other degradation mechanisms take their toll, the Mission Operations team may choose to have the spacecraft initialize to a different state should it reboot. Perhaps they will choose another set of thrusters as the default, or perhaps they will choose to power up a reduced set of components as the solar panels weaken with age. In effect, the autonomy and macro features are used to provide a scripting capability, much as Unix users use the C shell’s language to add features and functionality to their environment.

4. Verifying Autonomy

Dealing with issues such as these, on missions that often last a decade or more, is what makes spacecraft designers so conservative. One of the ways that this conservatism manifests itself is in the amount of testing performed on a typical science spacecraft. Testing is performed at both the subsystem level and at the spacecraft level. The latter usually begins a year or more before launch, as the spacecraft is being integrated. As each subsystem is added, its interfaces to the rest of the spacecraft are tested. Once the entire spacecraft has been integrated, various all-up tests begin, such as thermal vacuum tests, vibration tests, performance tests, and acoustic tests. Among the subsystems that must be exercised are the autonomy rules.

Consider the magnitude of the challenge: there are hundreds of autonomy rules, each looking for a different problem. Each problem must be simulated, and the rule’s response verified. Even testing only on a rule-by-rule basis, this is a large task. In fact, however, more testing is required. The rules are not really independent. They are coupled by the laws of physics and the dynamics of the spacecraft and its subsystems. An anomaly in one subsystem may, in short order, trigger anomalies in others. Testing and code inspections must therefore try

to uncover interaction between rules. A further challenge – which is usually addressed at the design stage, long before testing begins – is the avoidance of “dueling autonomy.” It is imperative that there not be competing autonomy masters on board, each reacting to the same event in conflicting ways. This is usually handled by concentrating the decision-making in one place. While several subsystems may cooperate to detect an anomaly, the resolution of the anomaly must be done by the autonomy engine. For example, the guidance and control (G&C) subsystem is best equipped to detect the failure of a reaction wheel. It should simply inform the autonomy engine that “wheel 3 is bad,” and let the autonomy engine power down wheel 3 and then send a command to the G&C telling it to use the remaining wheels. This allows the autonomy logic to be visible (as opposed to being buried in the G&C code), and to be controllable based on the rest of the spacecraft’s operation (since that rule can be disabled).

5. Future Directions

While the existing level of autonomy has allowed for robust spacecraft design in the face of intermittent communications, it is still a fairly low-level capability. One can envision the same functionality being implemented in an earlier era using analog comparators, relays, thermostats, and circuit breakers. Since there are really two aspects to autonomy rule programming – coding the “if” part and coding the “then” part – these offer two natural areas to improve.

One possible future direction is therefore to determine onboard what conditions the autonomy rules should look for. A typical autonomy rule predicate clause at the moment reads like this – this one intended to guard against an overheating Peak Power Tracker (PPT), a form of voltage regulator:

If

PPT_5 is powered ON AND

the number of peak power tracking converters powered ON is greater than 6

AND (the PPT_5 temperature exceeds the PPT_1 temperature by at least 20 C° AND the PPT_5 temperature exceeds the PPT_2 temperature by at least 20 C° AND the PPT_5 temperature exceeds the PPT_8 temperature by at least 20 C°) AND

(all the temperature measurements are valid) for 40 seconds

At the very least, one would hope that we can someday stop worrying about the validity of the data – perhaps an early use for machine learning algorithms could be to use them to detect stale or vastly incorrect data.

Another improvement might be in choosing the thresholds: what if 20 degrees C is not the correct value? What if it changes over the course of the mission, as radiation effects degrade the electronics and the thermal coatings lose the ability to shed or reflect heat? Since spacecraft are extensively tested before flight, the opportunity exists for an autonomy system to learn the proper behavior of the spacecraft. In the current human-intensive concept of operations, the Mission Operations team that will fly the spacecraft is often tasked with “flying” the spacecraft during testing for that very reason: to let them see how the spacecraft should behave.

A related direction is to increase the level of abstraction used in commanding a spacecraft – the “then” part of an autonomy rule. Much as computer programming evolved from machine language to assembly language to high-level languages, one can envision spacecraft commanding moving along a similar path. While we do currently tend to use macros to create higher-level constructs (i.e.: invoke the “Switch to Instrument B” macro rather than issue a series of individual relay commands) this is akin to the macro language implemented by most assemblers: it’s still up to the user to “compile” his own high-level code.

A perpetual challenge faced by anyone seeking to insert new technology into a spacecraft project is that of ensuring that the failure of the new technology will not imperil the mission. The natural tendency of the spacecraft designers is to fall back on the known, proven methods. One way to overcome this conservatism is to test the new autonomy system on the ground, using archived telemetry from a past mission. This is harder than it sounds, because it requires that the behavior of the subsystems be modeled. For instance, it is not sufficient for the test to check that the new autonomy system correctly detected that Instrument A failed. The test must also show that the response to that failure would be sound, and would not adversely impact the rest of the spacecraft. The simulators created as part of the current development approach model some, but not all, of the subsystem dynamics.

This same approach could then be extended to perform ground-based post-processing of data during a mission, eventually acting in a real-time advisory role, i.e.: “The autonomy system believes that Instrument A has failed, and suggests you issue the following commands.”

A next step might be to employ an “off-line processing” approach on the flight computer, in which the new

autonomy system runs in parallel with the existing one. The new autonomy system, rather than act on the conditions it finds, simply reports what it would have done if it were active, i.e. "The autonomy system believes that Instrument A has failed, and would like to respond as follows." In postprocessing, on the ground, the output of the new autonomy system can be compared with the raw telemetry, and only when the results are found to be valid is the new system brought on line – perhaps in a later phase of the current mission, perhaps in a future mission. It is worth pointing out, however, that space-qualified computers lag far behind their desktop counterparts – a difference of 10x in CPU bandwidth is typical

6. Conclusion

A typical spacecraft's autonomy system has been described, consisting of a series of independent if-then operations. Each such autonomy rule guards the spacecraft against the occurrence of a particular condition, and takes action if it's found. The action taken consists of executing a series of commands previously stored onboard by the Mission Operations team, such as turning off the offending subsystem and switching to its backup. Two possible areas for improvement have been identified, namely onboard autogeneration of autonomy rules (or at least, onboard "tweaking" of human-generated rules) and increasing the level of abstraction used to command the spacecraft when a rule's condition is satisfied. Finally a testing and validation approach was suggested to help establish the robustness and correctness of a new autonomy system.

References

- Baer, G. E., Harvey, R. J., Holdridge, M. E., Huebschman, R. K., and Rodberg, E. H. (1999). Mission Operations, *Johns Hopkins APL Tech. Dig.* 20(4), (pp. 511–521).
- Moore, R.C. (2002). Safing And Fault Protection For The Messenger Mission To Mercury, *21st Digital Avionics Systems Conference, Proceedings: IEEE 02CH37325C* (pp. 9.a.4-1 through 9.a.4-8)
- D. D. Stott, D. A. Artis, B. K. Heggstad, J. E. Kroutil, R. O. Krueger, L. A. Linstrom, J. A. Perschy, P. D. Schwartz, and G. F. Sweitze (1998), The NEAR Command and Data Handling System, *Johns Hopkins APL Tech. Dig. Volume 19, Number 2*
- M. E. Holdridge (2002). NEAR Shoemaker Spacecraft Mission Operations *Johns Hopkins APL Tech. Dig.* Volume 23, Number 1
- Harvey, R. J. (2001). Implementation Of An Autonomous Spacecraft Designed To Reduce Life-Cycle Cost, *Fourth International Symposium on Reducing the Cost*

of Spacecraft Ground Systems and Operations, Johns Hopkins University Applied Physics Laboratory.

Harvey, R.J. The TIMED Onboard Autonomy System (2001). *ESA Workshop on "On-Board Autonomy", October 2001 ESTEC*