Mars Science Laboratory CHIMRA/IC/DRT Flight Software for Sample Acquisition and Processing

Won S. Kim, Chris Leger, Joseph Carsten, Daniel Helmick, Stephen Kuhn, Richard Redick, Diana Trujillo
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA, USA
Won.S.Kim@jpl.nasa.gov

Abstract - *The design methodologies of using sequence diagrams, multi-process functional flow diagrams, and hierarchical state machines were successfully applied in designing three MSL (Mars Science Laboratory) flight software modules responsible for handling actuator motions of the CHIMRA (Collection and Handling for In situ Martian Rock Analysis), IC (Inlet Covers), and DRT (Dust Removal Tool) mechanisms. The methodologies were essential to specify complex interactions with other modules, support concurrent foreground and background motions, and handle various fault protections. Studying task scenarios with multi-process functional flow diagrams yielded great insight to overall design perspectives. Since the three modules require three different levels of background motion support, the methodologies presented in this paper provide an excellent comparison. All three modules are fully operational in flight.*

Keywords: flight software, software design, task scenario, sequence diagram, state machine, sample processing.

1 Introduction

The MSL (Mars Science Laboratory) SA/SPaH (Sample Acquisition, Sample Processing and Handling) subsystem [1] acquires rock and regolith samples from the Martian surface, processes them into fine particles through sieving, and delivers small portions of the powder into analytical science instruments, SAM (Sample Analysis at Mars) and CHEMIN (Chemistry and Mineralogy X-ray diffraction). It includes the Robotic Arm, Drill, CHIMRA (Collection and Handling for In-Situ Martian Rock Analysis) [2], ICs (inlet covers), and DRT (Dust Removal Tool). CHIMRA has four actuators, and its important requirement is that another actuator can run concurrently while the vibe actuator is running in the background. There are three ICs, each having an actuator that can run in the foreground with no concurrent motion requirement. DRT has only one brush actuator. Nevertheless, it is required to be able to run in the background to allow a DRT command to change the brush speed or an ARM command to move the arm while the brush is running. Unlike CHIMRA, however, DRT itself does not run two motors concurrently. Incidentally, CHIMRA, IC, and DRT modules exhibit three different levels of foreground, background, and concurrent motion control.

An MSL FSW (flight software) design guideline is to use sequence diagrams and state machines for given FDD (functional design description) and software requirements of a module. This methodology is especially crucial when designing the software involving complex interactions between multiple processes to support concurrent motions and fault protections. Understanding task scenarios with multi-process functional flow diagrams also provides great insight. These design methodologies are successfully applied in designing CHIMRA, IC, and DRT modules.

The architecture and software designs for CHIMRA, DRT, and IC modules are presented in Sections 2, 3, 4, and 5. Successful flight operations are mentioned in Section 6.

2 SSS System Software Architecture

The MSL SSS (Surface Sampling & Science) system software architecture serves to accomplish effective coordination and fault handling of five SA/SPaH mechanism modules and two science instruments. All nominal ground commands for SA/SPaH mechanisms go through the SPAM (Sample Processing and Acquisition Manager) module, which handles individual commands as well as SSS command scripts. SPAM checks ACM (Activity Constraint Manager) conditions and asks ARB (Arbiter) for the resources necessary to complete the command. Once the ARB resource is granted, the command is routed to the corresponding SA/SPaH mechanism module.

3 CHIMRA

CHIMRA is a Robotic Arm turret mounted sample processing system, delivering sub-150 micron rock powders and sub-1 mm soil samples to science instruments in coordination with ARM and DRILL. The CHIMRA FSW module is responsible for handling all motions of the CHIMRA mechanism consisting of four motors: SCOOP,

THWACK, PORTION, and VIBE. There are three types of CHIMRA commands: non-move, foreground move, and background move commands. An example of a non-move command is to dump CHIMRA reports. Basic software requirements for the CHIMRA foreground move commands are to open and close the scoop, thwacker, or portioner, or to vibrate, by running a single motor in the foreground without concurrent motions. The CHIMRA module achieves actual motion control by sending motion requests to the MOT (motor) module. If the motion is supposed to hit a hard stop, a rate servo control request is sent to MOT, and the motion stops when a stall is detected upon hitting the hard stop. If the motion is supposed to reach the goal position without hard stop contact, a position servo control request is sent to MOT.

Three standard downlink telemetries for MER/MSL [3] are EVR (event reporting), EHA (engineering, housekeeping, and accountability) channelized telemetry, and DP (data product). EVRs are time-stamped messages of formatted output functionality similar to printf(). CHIMRA generates EVRs to inform states, activities, as well as fault occurrences. EHA channel data are conveyed in packet streams that are transmitted to ground. CHIMRA EHA channels include the fault status and four mechanism states. Data products are files with their associated metadata stored in nonvolatile memory. CHIMRA produces four data products: summary (commands and various states), motion (foreground motion telemetry), bg_motion (background motion telemetry), and anomaly reports (IPC messages). CHIMRA first writes the data for the reports in their corresponding history RAM (random access memory) buffers during the course of command executions. Later, HST (History Collection) module can be asked to the dump the report to a nonvolatile flash memory so that they can be downlinked to Earth. CHIMRA supports automatic dump. When a history buffer is filled more than a specified fraction of its data buffer size, CHIMRA automatically requests HST to the dump the report. CHIMRA also provides a non-move command to dump selected reports.

An example of a sequence diagram for a foreground motion command can be found in the middle portion of Figure 1. Upon receiving a SCOOP_OPEN foreground command from SPAM, CHIMRA checks ACM conditions. For instance, if CHIMRA MOTION_ERROR is TRUE, it is unsafe to perform foreground motion. If it is safe, start the motion by sending a continuous motion request to MOT together with its associated callback function. During the motion, MOT gets the new motor data, executes the callback function at 64 Hz, and writes the motor telemetry to the motion history buffer. After the motion execution in the MOT context is complete, CHIMRA notifies ACM that the foreground actuator is not in use. Thereafter, CHIMRA optionally requests the HST (History Collection) module to dump summary and motion reports. Finally, a command reply is sent to SPAM.

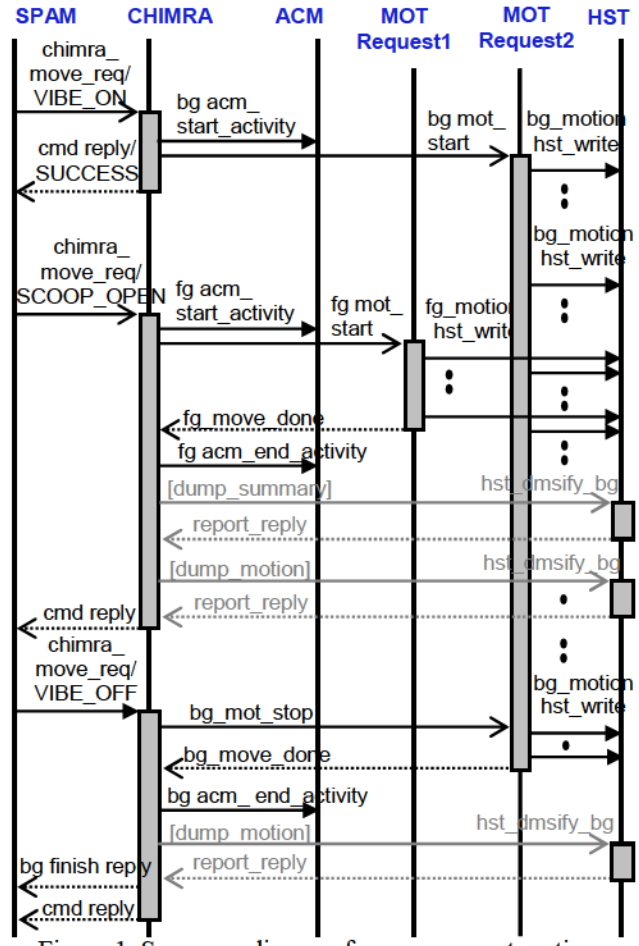

Figure 1. Sequence diagram for a concurrent motion scenario

Another important FSW requirement is to enable concurrent foreground motion while the vibe is running in the background. In fact, the whole sequence diagram of Figure 1 illustrates the nominal scenario of this operation. Upon receiving a VIBE_ON command, CHIMRA checks ACM conditions. If it is safe to perform background motion, start the background vibe by sending a continuous motion request to MOT together with its associated callback function. After a specified time delay from the motion start, return a command reply with SUCCESS to SPAM so that SPAM can send a CHIMRA, ARM, or DRILL command while the vibe is still running. During the background vibe motion, MOT gets the new motor data, executes the callback function at 64 Hz, and writes the motor telemetry to the bg_motion history buffer. While the background vibe is running, a foreground command such as SCOOP_OPEN can be executed. Upon receiving a VIBE_OFF command, CHIMRA stops the vibe by sending a mot_stop request to MOT and then notifies ACM that the background vibe is not in use. CHIMRA then sends both a bg_finish reply and a VIBE_OFF command reply with SUCCESS. Note that the background VIBE_ON command returns two replies: a command reply after the vibe is started and a bg_finish_reply after the vibe is finished.

Figure 2 shows a multi-process functional flow diagram, where two concurrent processes of CHIMRA foreground and background motions are displayed in parallel. CHIMRA sends continuous motion requests to MOT together with the callback functions. Each motion callback function is executed in MOT context at 64 Hz, transitioning over three states starting from RUNNING to STOPPING and then back to IDLE at completion. The callback function updates the motion or bg_motion history buffer and processes the CHIMRA-specific algorithm including the algorithm's internal state transition. The callback function also sends the background telemetry of small vibe motion data to CHIMRA at 1 Hz if the vibe is in motion. This background telemetry is used to update the vibe EHA channels during the motion.

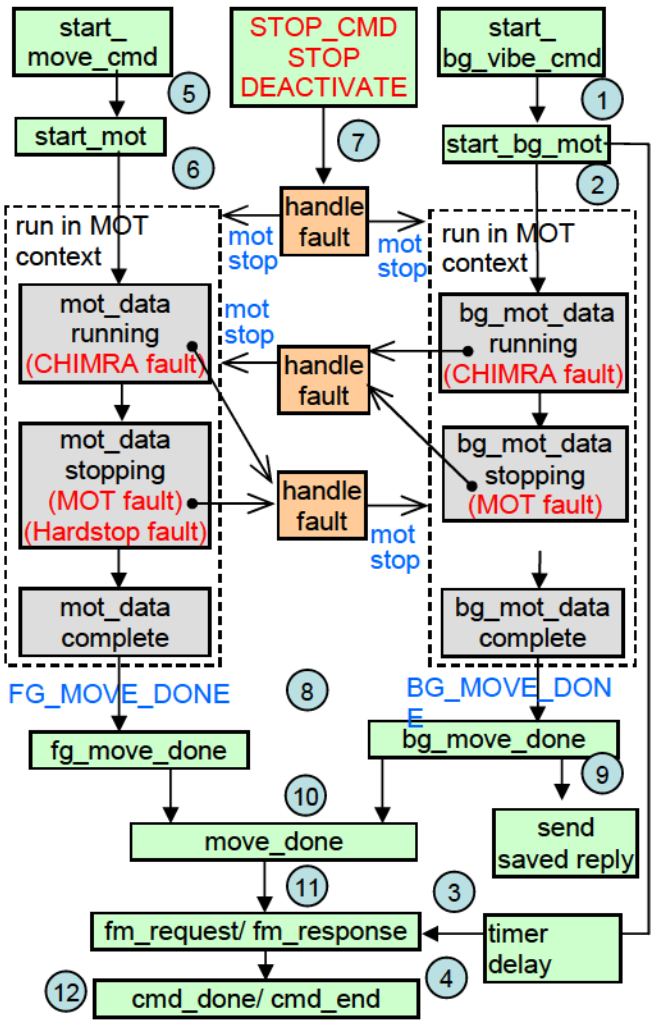

Figure 2. A multi-process functional flow diagram

Fault detection and handling are also depicted in Figure 2. During the RUNNING state, external faults and CHIMRA-specific faults are checked. External faults occur when CHIMRA receives a STOP_CMD, a STOP request from SPAM, a DEACTIVATE request, or an out-of-range thermal state update from THERMAL. During the STOPPING state, MOTOR faults are checked by examining the safety and error data fields of the motion request telemetry. When the disposition data field indicates that the motion is complete, a HARD_STOP fault is also checked. A HARD_STOP fault is declared if the final motor position is not within the specified tolerance. Any fault occurrence causes both the foreground and background motions to stop. MOTOR and CHIMRA motion faults set both the ACM GOAL_ERROR and MOTION_ERROR to TRUE, while non-motion and external faults set only the ACM GOAL_ERROR to TRUE.

Finally, task step numbers are added to illustrate task scenarios in Figure 2. Let us study a fault scenario caused by an external fault. Upon receiving a background VIBE_ON command, CHIMRA performs Steps 1-4 to start the vibe actuator in the background. This corresponds to the first portion of the sequence diagram of Figure 1. While the vibe is running, a foreground motion command is received. CHIMRA performs Steps 5-6 starting a motor in the foreground. Before the foreground motion is complete,

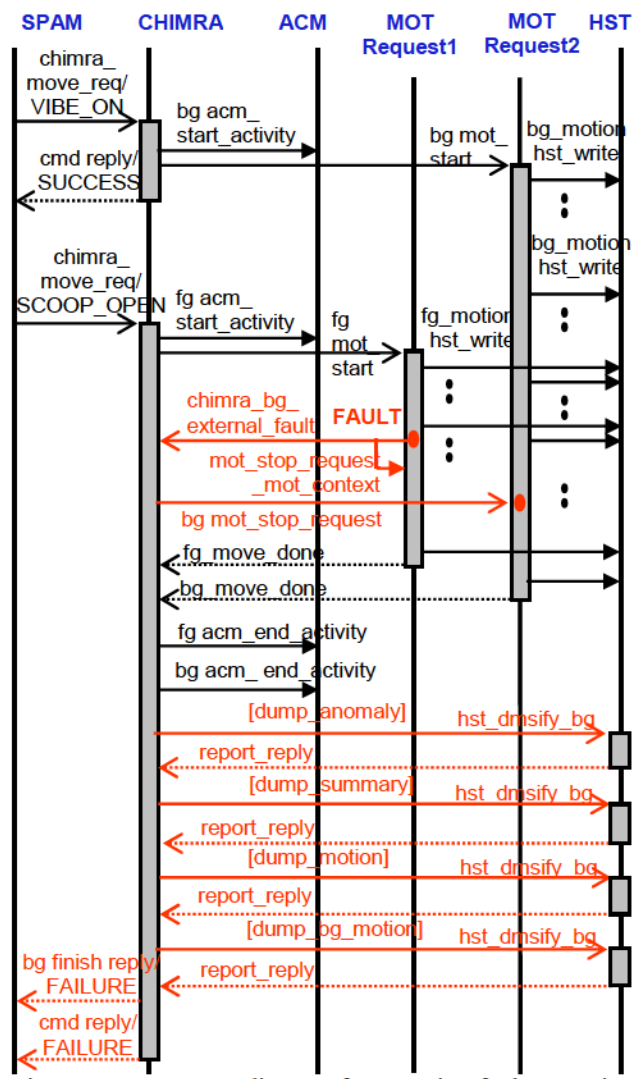

Figure 3. A sequence diagram for a motion fault scenario

an external fault occurs, for instance, by a STOP_CMD (Step 7). The external fault causes both foreground and background motors to stop. When the background vibe finishes (Step 8), a bg_finish_reply is sent (Step 9). Only after both motors are stopped, CHIMRA proceeds to the next step (Step 10). The angles of several sample pathway vectors relative to gravity are updated by using the turret frame attitude obtained from FM (Frame Manager) (Step 11). Finally, data products are dumped and a command reply is sent (Step 12).

Figure 3 shows the sequence diagram for a task scenario where an internal CHIMRA fault occurs in the foreground motion. The fault causes both the foreground and background motions to stop, and thereafter dump all four reports. Both a bg_finish reply and a command reply are returned with FAILURE status.

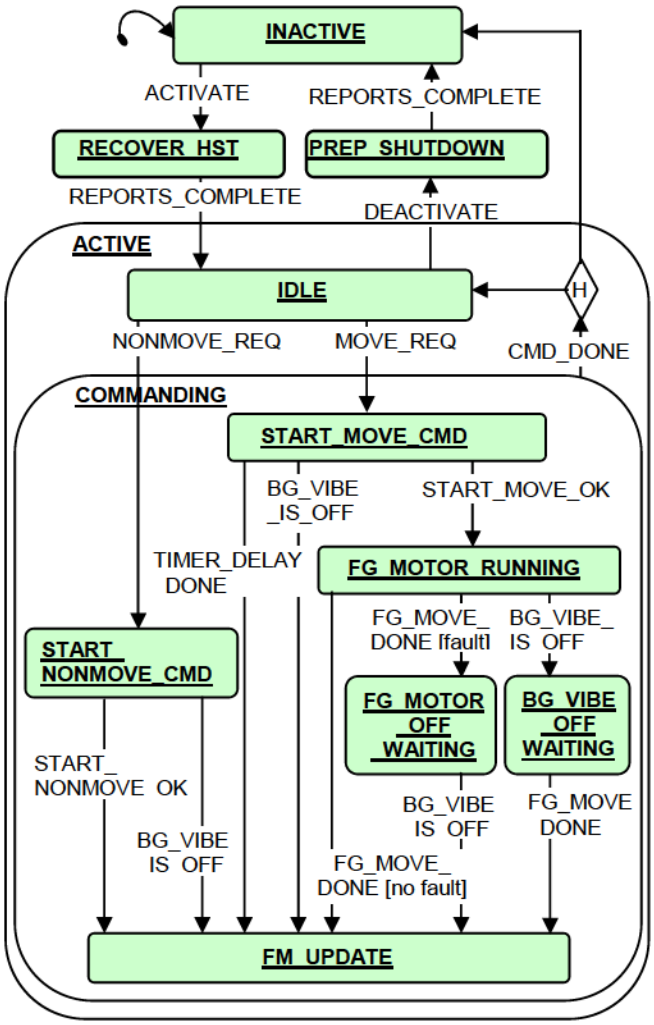

Figure 4. CHIMRA main state machine

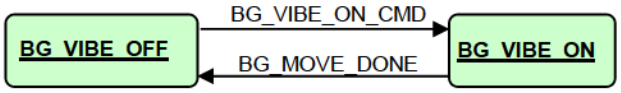

Figure 5. CHIMRA background state machine

Since CHIMRA must support concurrent foreground and background motions, its behavior can be best represented by two state machines. The main and background state machines are shown in Figures 4 and 5, respectively. Hierarchical state machines [4] are employed, so that common message handlings are done in parent states, while specific message handlings are done differently in each state. At the system boot, CHIMRA module is first initialized, clearing CHIMRA state variables and restoring NPM (Nonvolatile Parameter Manager) stored variables and auto-coded parameters. Thereafter, CHIMRA task is spawned and starts the CHIMRA state machines. Thereafter, it runs an infinite loop that checks and handles incoming IPC (inter-process communication) messages [3]. When there are no IPC messages in the CHIMRA IPC queues, the CHIMRA task is blocked. When a message is received in any of enabled CHIMRA IPC queues, the CHIMRA task dispatches the message to the auto-coded handler first to handle CHIMRA parameter commands. If the message is not for the auto-code handler, it is dispatched to the CHIMRA state machines. After the CHIMRA task is spawned, the task is activated as a final setup. Upon receiving an ACTIVATE message, the main state machine transitions from INACTIVE to RECOVER_HST. After the recovery of any remaining history buffer data, the state machine transitions to IDLE.

If a non-move command is received in IDLE state, the main state machine transitions to START_NONMOVE_ CMD. After the command execution, it transitions to FM_UPDATE and then back to IDLE. When a DEACTIVATE message is received in the IDLE state, the main state machine transitions to PREP_SHUTDOWN, automatically dumping any remaining history buffer data before transitioning to INACTIVE.

Let us follow the sequence diagram of Figure 1 as an example to study the behavior of the state machines. Upon receiving a background VIBE_ON command in the IDLE state, the main state machine transitions to START_MOVE_CMD. CHIMRA sends a background motion request to MOT, and the background state machine transitions to BG_VIBE_ON. After a specified time delay, the main state machine transitions to FM_UPDATE. Sample pathway vector angles are updated, and history buffers are optionally dumped. Thereafter, the main state machine transitions back to IDLE. In the IDLE state, a CHIMRA foreground motion command is allowed while the vibe is still running. Upon receiving a foreground motion command, the main state machine transitions to START_MOVE_CMD and then to FG_MOTOR_ RUNNING. When the foreground motion is complete, the main state machine transitions to FM_UPDATE and then back to IDLE state. Finally, upon receiving a VIBE_OFF command, the main state machine transitions to START_MOVE_CMD. After the background vibe motion stops, the background state machine transitions to

BG_VIBE_OFF. Thereafter, the main state machine transitions to FM_UPDATE and then back to IDLE.

If a fault occurs while both foreground and background motions are running, the main state machine must wait until both motions are stopped before transitioning to FM_UPDATE. To accomplish this goal, two states of FG_MOTOR_OFF_WAITING and BG_VIBE_OFF_WAITING are inserted between the FG_MOTOR_RUNNING and FM_UPDATE. If the foreground motion finishes first, the state machine transitions to FG_MOTOR_OFF_WAITING before reaching FM_UPDATE state. On the other hand, if the background motion finishes first, the state machine transitions to BG_VIBE_OFF_WAITING. Note that a state transition is possible only by an IPC message, and the CHIMRA task can send IPC messages to itself to facilitate the state transition.

If another command is received in the COMMANDING state, CHIMRA does not defer the message by using disable and enable queues. Instead, for the mechanism safety reason, CHIMRA rejects this message with a FAILURE reply, declares a SEQ fault, stops any actuator motion, sets ACM GOAL_ERROR to TRUE, dumps all three reports, and replies the current command with FAILURE.

4 Inlet Covers

The purpose of the IC (Inlet Covers) subsystem is to keep the inlet funnels of the SAM and CHEMIN science instruments covered except during the delivery of soil and rock powders using ARM and CHIMRA. The IC FSW module is responsible for opening and closing three inlet covers: two for SAM and one for CHEMIN. Each inlet cover has its own individual actuator that does not require any concurrent or coordinated motions.

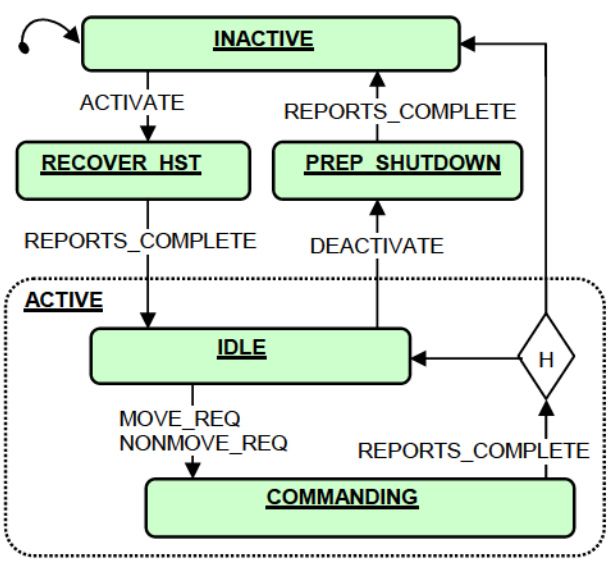

Figure 6. IC hierarchical state machine

IC motion control is essentially the same as CHIMRA foreground motion control. The main difference is that IC uses notify_timed motion requests instead of continuous motion requests because IC does not need IC-specific algorithms to run during the motion control in MOT context at 64 Hz. Only at the end of the motion completion, fault occurrences are checked. For instance, a POS_TOL fault is declared if the final motor position is not within the specified tolerance. Since IC does not need to support any background commands, the state machine is quite simple consisting of only IDLE and COMMANDING states when activated as shown in Figure 6.

5 Dust Removal Tool

The DRT (Dust Removal Tool) has a single brush actuator mounted on the turret at the end of the Robotic Arm. Its usage is to clean off material from natural rock surfaces and from the observation tray mounted in front of the rover. The DRT FSW module is responsible for handling the brush actuator motion. The brush actuator needs to change its speed while in motion. This requires the brush to run in the background so that the CHANGE_SPEED command is allowed while the brush is running. A continuous motion request must be used to change the brush speed on the fly without stopping the motion. Furthermore, there is a software requirement that the Robotic Arm should be able to move while the brush is running. To allow the coordination of the concurrent motions, SPAM needs to use a background DRT brush command. However, unlike CHIMRA, DRT itself does not need to handle two concurrent motions.

Figure 7 shows a sequence diagram for a task scenario of turning on the brush, changing the brush speed, and finally turning off the brush. The DRT state machine is also shown in Figure 8. Upon receiving a background BRUSH_ON command, the DRT state machine transitions from IDLE to COMMANDING state. The software steps to follow for the background BRUSH_ON are essentially the same as those for the background VIBE_ON described in Section 3 with Figure 1: ACM check, continuous motion request with a callback function, followed by a command reply after a specified time delay. Thereafter, the state machine transitions to BG_BRUSH. While the brush is running, a CHANGE_SPEED command is allowed. Upon receiving the command, the state machine transitions to COMMANDING state. The callback function executing in MOT context calls a mot_update function to change the motor speed. After sending a command reply with a specified time delay, the state machine transitions to the BG_BRUSH. Finally, when a BRUSH_OFF command is received, the state machine transitions to COMMANDING, and later the brush motion is stopped. After DRT summary and motion reports are optionally dumped, a bg_done reply and a command reply are sent. Then, the state machine transitions to the IDLE state.

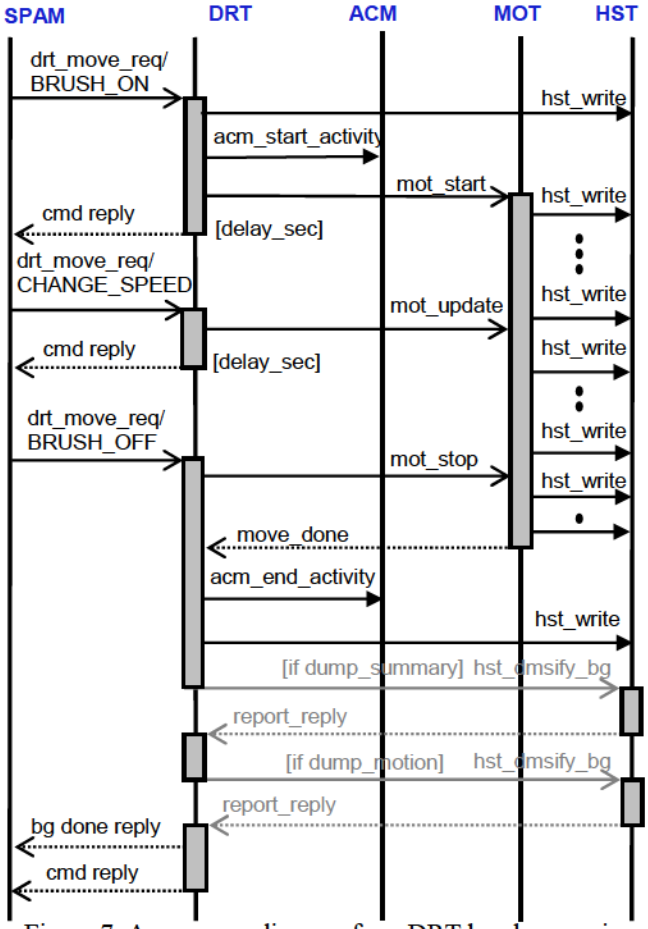

Figure 7. A sequence diagram for a DRT brush scenario

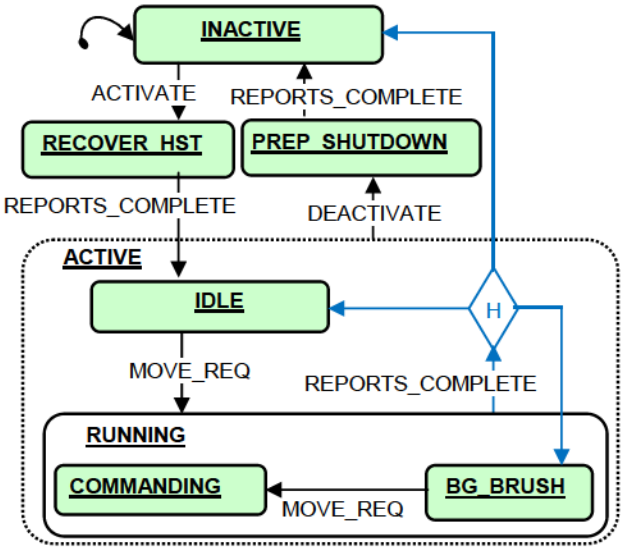

Figure 8. DRT hierarchical state machine

In Figure 8, the history state H is used. After completion of dumping reports in the COMMMANDING or BG_BRUSH state, the DRT state machine transitions to one of the three states, depending upon the history state. For non-move and foreground move commands, the state machine transitions to IDLE (H = IDLE) at the end of the command. For the background BRUSH_ON command, the state machine normally transitions to the BG_BRUSH state. For the DEACTIVATE request, it transitions to the INACTIVE state (H = INACTIVE) in the end.

6 MSL Flight Operations

Since MSL successfully landed in August, 2012, all three mechanisms of CHIMRA, IC, and DRT have been used successfully. CHIMRA vibe and ARM motions were used simultaneously for sample processing. Inlet covers were used for sample drop-off into instruments. DRT brush and ARM motions were used simultaneously for dust removal from a rock surface.

7 Conclusions

This paper presents the MSL flight software designs of CHIMRA, IC, and DRT modules using sequence diagrams, multi-process functional flow diagrams, and hierarchical state machines. CHIMRA was designed with two state machines. DRT was designed with a single state machine but with an extra state compared to IC. All three software modules are new for Mars flight missions and fully operational in flight.

Acknowledgements

The research was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] L. Jandura, "Mars Science Laboratory Sample Acquisition, Sample Processing and Handling: Subsystem Design and Test Challenges," Proc. of the 40th Aerospace Mechanisms Symposium, May 2010.

[2] D. Sunshine, "Mars Science Laboratory CHIMRA: A Device for Processing Powdered Martian Samples," Proc. of the 40th Aerospace Mechanisms Symposium, May 2010.

[3] G. Reeves and J. Snyder, "An overview of the Mars exploration rovers' flight software," IEEE Conf. on Systems, Man, and Cybernetics, Big Island, HI, 2005.

[4] M. Samek, *Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems,* Newnes-Elsevier, 2009.