Autonomous Payload Operations Onboard the International Space Station

Howard K. Stetson
Teledyne Brown Engineering
Huntsville, AL 35812
256-961-0399
howard.k.stetson@msfc.nasa.gov

David K. Deitsch
Teledyne Brown Engineering
Huntsville, AL 35806
256-961-0467
david.k.deitsch@msfc.nasa.gov

Craig A. Cruzen
NASA Marshall Space Flight Center
Huntsville, AL 35812
256-544-8658
craig.cruzen@nasa.gov

Angie T. Haddock
NASA Marshall Space Flight Center
Huntsville, AL 35807
256-544-6285
angie.haddock@nasa.gov

Abstract—Operating the International Space Station (ISS) involves many complex crew tended, ground operated and combined systems. Over the life of the ISS program, it has become evident that by having automated and autonomous systems on board, more can be accomplished and at the same time reduce the workload of the crew and ground operators. Engineers at the National Aeronautics and Space Administration's (NASA) Marshall Space Flight Center in Huntsville Alabama, working in collaboration with The Charles Stark Draper Laboratory have developed an autonomous software system that uses the Timeliner User Interface Language and expert logic to continuously monitor ISS payload systems, issue commands and signal ground operators as required. This paper describes the development history of the system, its concept of operation and components. The paper also discusses the testing process as well as the facilities used to develop the system. The paper concludes with a description of future enhancement plans for use on the ISS as well as potential applications to Lunar and Mars exploration systems. [1]

TABLE OF CONTENTS

1

[1] IEEEAC paper #1066, Final Version, November 28, 2006

1. INTRODUCTION

In April of 2007, the International Space Station (ISS) laboratory module Destiny will have been supporting science research in Earth orbit for over six years. Many of the spacecraft development and scientific research accomplishments of the ISS program have been documented in the media, professional journals and academic publications. However there are a number of supporting technologies, rarely mentioned that are employed on a daily basis in order to ensure the scientific and engineering data generated by the systems onboard are handled properly. One of these vitally important systems is a software tool commonly referred to as "Timeliner."

Flight controllers at the NASA-Marshall Space Flight Center (MSFC) ISS Payload Operations and Integration Center (POIC) operate the scientific experiments, payload support, data and video systems onboard the ISS. These systems are complex and require a great deal of expertise and engineering know-how to operate effectively. See Figure 1.

Figure 1 - The Payload Operations and Integration Center

During the early years of ISS operations, controllers were routinely sending dozens of commands daily just to configure data systems, activate and deactivate experiments and many other routine, complex, yet necessary activities in order to successfully utilize the ISS as a world-class science facility.

Fortunately, NASA had anticipated that this would be the case and had planned to use the Timeliner User Interface Language (UIL) on ISS. The Timeliner UIL was developed by the Charles Stark Draper Laboratory [1] in 1981 for use in simulating tasks performed by astronauts aboard the Space Shuttle. In 1992, Timeliner was selected by NASA as the user interface language for the ISS, and it was incorporated into the ISS Command and Control Multiplexer-DeMultiplexer (MDM) and the Payload MDM (PLMDM), see Figure 2 [2].

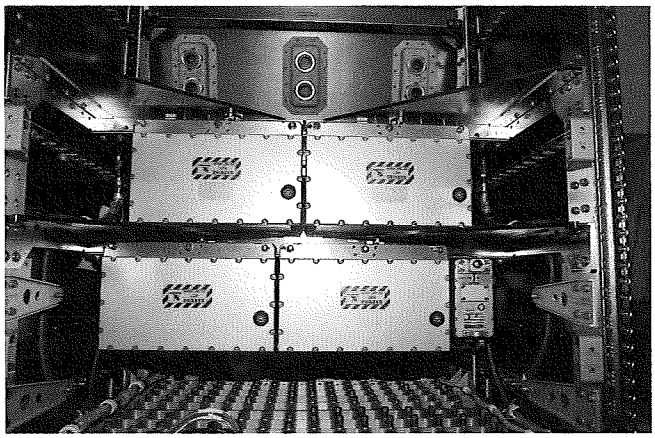


Figure 2 – C&C and PL MDMs Installed in the ISS

Beginning in 1999, software engineers at the POIC, working with Draper, began developing autonomous blocks of software, called Timliner Bundles in new and innovative ways to reduce ground controller workload, add reliability and to a certain extent, put a virtual-controller onboard. From those humble beginnings to the present, engineers, scientists and flight controllers have developed an autonomous and continuously executing system on board the ISS called Higher Active Logic (HAL). HAL incorporates the Timeliner UIL language constructs and is integrated in such a way so as to mimic human decision processes in order to minimize flight controller interaction in routine tasks, while at the same time incorporating a conservative and safe approach to autonomous operations on a manned spacecraft. This paper details the Timeliner design fundamentals, operations concepts for HAL, prototype systems as well as the requirements, testing and review and approval process for such a system.

2. Constructs of The Timeliner UIL

Language Constructs That Model Human Decisions

The Timeliner UIL provides higher level programming constructs that model many human decisions that are made day to day. These constructs relate directly to the decision processes made by ground operators each day concerning such things as traffic control, manufacturing processes, and any other work environments that are procedural. These constructs provide an easier coding paradigm that allows non-computer programmers the ability to follow and understand the execution as it takes place. The English language type constructs allow the actual compiler listings to be used for following the execution of autonomous operations. Ground operators can also scan the compiler listings to predict autonomous operations behavior prior to events taking place. Listed below are some examples of the language constructs and how they can be used to automate daily tasks.

***EVERY** Statement - A Timed Control Loop*

The EVERY statement models the repetitive actions performed on a timely basis. You may turn your outside lights on every night at 8:00 PM, or you may water the yard every day at 5:30 PM. There are a great number of actions that are performed on a periodic time basis and this is when the EVERY statement is used. In the HAL system, the EVERY statement is utilized for monitoring payload power controllers for any change in state and for monitoring whether ISS experiments are providing health and status (H&S) data. Purely event driven cases such as these do not negate the use of time itself to autonomously perform an action. You may wish to perform an action every 5 minutes or on a daily basis or on an hourly basis. Embedding logic code within the EVERY control loop allows further decision making to be employed once the time has been reached for this cyclic action. In the case of power controller monitoring, additional checks are made to determine that H&S data packets are being sent from the just powered payload subsystem and commands are sent to initialize the data processing of the newly arrived H&S data. The HAL system performs the parameter monitoring and commanding nominally performed by ground operations personnel, without being constrained by the communications state with the ground. This allowed the POIC Timeliner developers to move a virtual-ground controller to on-board the ISS and allows the crew to power up or down payload subsystems with less ground interaction.

***WHENEVER** Expression **BEFORE/WITHIN** – A Global Check Forever*

The WHENEVER statement models the situational decision a human makes whenever an event occurs that will have the same action each and every time. By itself, it is simplistic such as when a fire is detected, the fire alarm is sounded,

windows and doors are automatically closed and the fire department is called. Modified by the BEFORE clause, WHENEVER can be more complex by adding an additional condition. Modified by the WITHIN clause, WHENEVER can be time restricted. As used on board the ISS, the intended use of this construct is to start the daily downlink of acceleration data stored within a payload whenever the time reached 2:00PM every day. Embedding logic code within the WHENEVER control loop allows further decision making to be employed once the expression has been evaluated (i.e. is there communications with the ground and if not, is the data recorder in record mode?). The employment of the WHEN statement embedded within the WHENEVER control loop can pause the actions to be taken until the conditions are correct for the actions.

***WHEN** Expression **BEFORE** Expression /**WITHIN** Time Frame – A Timed Local Check*

The WHEN statement models the human decisions while in the process of taking actions. It will simply wait for the condition to become true before proceeding. Modified by the BEFORE clause, WHEN can become more complex, providing an additional check before an action is taken. Human modeling for a WHEN can be a condition encountered or when a specific Time is encountered. WHEN can be made temporary when modified with a WITHIN Clause. Figure 3 illustrates an example of down linking acceleration data daily at 2:00 PM, when ISS has AOS with the ground, or the comm. outage recorder is recording.

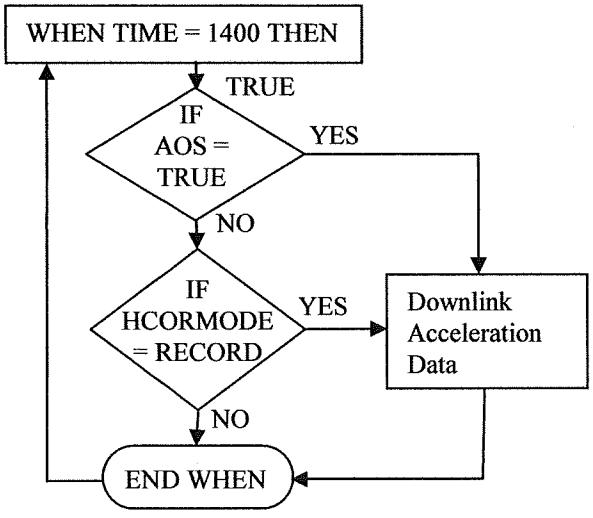


Figure 3 – WHEN Statement Example

Use is flexible enough to provide for the way a specific human "thinks" allowing different logic to behave in the same manner. The use of time in the expression such as WHEN TIME = or any of the sub-components of time such as day, hour, minute, second or even the day of the year allows extreme flexibility for loop control on an event reaction.

OTHERWISE – The Higher Level Alternative

The OTHERWISE statement models the decision branch taken if the WHEN evaluates to false or times out. Since the WHEN is a timed loop, otherwise can set up the conditions for re-entering the WHEN construct repetitively, see figure 4.

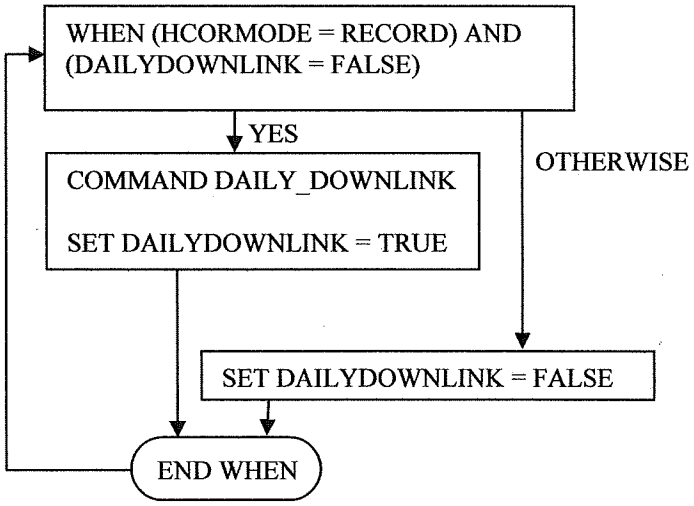


Figure 4 – Re-Entering the WHEN Construct

***WAIT** Time Period - The Alternative End Item*

There are cases where actions do not have end item verification, and it takes a period of time for the action to be completed, and this time period is consistent and can be measured. Such a case may be when starting an engine and waiting for the oil pressure or temperature to be at a pre-determined range. If sensor data for oil temperature/pressure is not available for sampling, you can WAIT a period of time before engaging the engine other than at idle. Knowing that a warm engine without the choke engaged provides optimum efficiency of the engine and that the engine arrives at this optimum within 2 minutes, allows a WAIT to be implemented. The scenario for a turbo charger cool down period after use before shutting the engine down is another example that can be applied today. The HAL system 3 uses WAIT statements between English language text presentations to allow operators the time to read the text before the next message is presented. It also uses WAIT statements between commands for down linking status data from the Minus Eighty Laboratory Freezer for ISS (MELFI) as it takes up to 3 seconds for MELFI to process each dump command. WAIT can be employed whenever a time period is needed before continuing and no end item is available for the previous action or activity.

Command, Telemetry and Crew Interfaces

ISS payload Timeliner systems are compiled on the ground and then the executable files are uplinked to the PLMDM. The complier interface provides fully instantiated (hard-

coded) commands built into the Timeliner executable. On the ground, the ISS payload Timeliner compiler interface pulls each command from the command database, and updates all modifiable command fields with the coded constants provided after type and range checking is performed. Nominally ISS commands are built from generic shells that are updated with parameters during execution. The POIC decided to forego this type of implementation and to provide fully instantiated commands that cannot be updated in real-time in order to eliminate safety concerns and to provide an autonomous system that transmits identical commands each and every time. This paradigm simplified the approval of Timeliner bundle content and test requirements. And although executable file sizes are larger than they need to be, there have been no impacts to PLMDM operations due to the larger sizes.

The payload Timeliner system has a shared memory area within the PLMDM for inter-bundle data exchange. While this adds to the autonomy of the system, the dependency upon memory mapping is very extensive. During execution, a telemetry identifier reference will cause the value within the memory address to be evaluated. Precision is of importance here as floating point values on a 32 bit word may not equate exactly to the number of decimal places being tested against. In this case, a range evaluation works better. The best solution is to utilize 64 bit floating point values to increase sampling precision.

In the case where a sequence writes to the shared memory, data exchange between bundles becomes a simple SET statement in one sequence/bundle, and a looping sampling code segment in another sequence/bundle sampling for a particular value being written to memory. The receiving sequence/bundle then clears the memory location to indicate it has received the value and is being processed.

The Timeliner system also has built in crew interfaces as part of the language specification. MESSAGE statements present text to operators, allowing status to be presented in ASCII text to ground and on-board consoles. The WARNING statement has the same functionality, but includes an optional function that will pause execution of the sequence and issue the warning. Execution must then be re-started with a command from the operator. The payload Timeliner system does not utilize the MESSAGE and WARNING statements since ASCII writes to memory are used instead. The ASCII messages become telemetry at that point and are sent to the ground.

The CONFIRM statement presents messages to operators and waits for a confirmation (Yes/No) response. CONFIRM can be used to verify the completion of a manual operation when no telemetry response is available. The PAUSE keyword can also be used with CONFIRM with the same functionality of stopping the executing sequence. The QUERY statement presents text and expects numeric input from the operator. Again, the PAUSE keyword can be used to hold execution of the issuing sequence. Fully autonomous insertion of data into executing sequences is provided by the payload system via shared memory without pausing and allows either the operator or executing sequence the capability to insert data. Finally, the RESPOND command is used by ground operators to provide input for the crew. In this way, ground operators can reduce crew interaction.

3. Initial Operations Concept and Proof of Concept

Initial Coding and First Bundles

When the first elements of the ISS were arriving on orbit, developers started making their initial plans for utilizing Timeliner for payload operations. While Timeliner had been developed with spacecraft applications in mind, up to this point, it had not been fully tested on the space qualified processors that would fly on the ISS. This not only left the detailed design work, but also the trailblazing task of software and hardware integration for the payload Timeliner developers [3].

The first step was to get the command interface working. The ISS architecture is designed on a tier structure of MDMs. The MDM(s) are linked together by an IEEE 1553 bus architecture. While this architecture is robust for command and control, it does present problems when dissimilar processors are linked together through the interface, such as the way Motorola and Intel processors are used on ISS. The word order of memory representation is different for these processors, which creates several mapping problems for passing of data from one processor to another. This is further complicated with byte swapping of data at each Bus Interface Adapter (BIA). Data, therefore, must be byte swapped with respect to the destination processor. This creates several problems in a tier structure of three or more levels. As data is propagated through processors, byte swaps occur depending on the number (odd or even) of dissimilar MDM processor cards that it passes through. This challenging hurdle was overcome by closely working with the various Interface Definition Documents that describe the ISS command and control structure, as well as through analysis and testing.

As mentioned above, one of the primary safety constraints placed upon payload Timeliner bundles was that they were only to use commands that are fully instantiated and embedded in the executable. This means that the bundles can only send predefined and fully tested commands, rather than allowing Timeliner to create updateable commands based upon variable components. This created a non-trivial job of verifying every command in the bundle. However it

also made it very straightforward to verify that no hazardous commands would be sent inadvertently from the PLMDM.

Timeliner commands are sent via a similar command path as commands sent from the ground. However, since the Timeliner commands originate from onboard, they do not go through the same set of processors. Therefore, commands originating from the ISS PLMDM have to be pre-byte swapped for execution. The final product is a byte swapped executable with fully instantiated commands imbedded within the bundle. This difficult development was made possible through the extensive use of Timeliner Data Access Run Time Libraries.

The first Timeliner bundles on the ISS were simple. One supported a payload called the Microgravity Acceleration Measurement System (MAMS). This experiment is a sensitive set of accelerometers housed within EXPRESS Rack 1 in the US Laboratory; see Figure 5 [4]. This payload has the requirement, when active, to downlink its memory buffer to the ground every 24 hours. Initially this was a scheduled set of commands sent by a controller on the ground. The developers at MSFC quickly realized this was a perfect application for Timeliner. The final implementation included a single kick-off command sent from the ground at a best time with regards to other operations onboard and communications coverage. This single command then started the MAMS_DAILY_DOWNLINK bundle on the PLMDM. This bundle sent a list of commands that executed a data dump, a "wait" for a period of time, a second dump to ensure all data was captured, and then a final command that erased the MAMS data buffer so a new set of data could be collected. While not completely autonomous, it definitely paved the way for more onboard automation in support of ISS science operations.

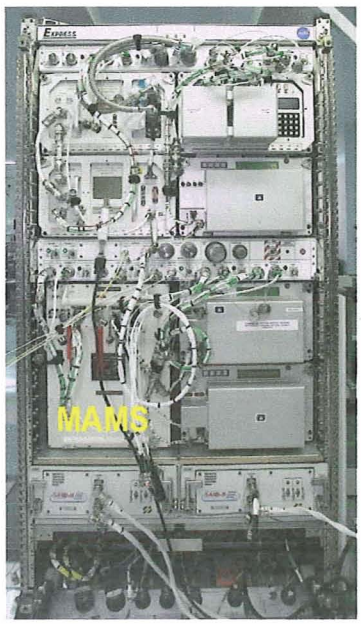


Figure 5 - EXPRESS Rack 1 and MAMS Payload

Another bundle supported an entire payload rack during its power-up and power-down sequence. The Microgravity Science Glovebox (MSG) rack, also in the US Laboratory, see Figure 6, required eight separate commands to start monitoring of its H&S data after power was applied. Early in the ISS program, each of these commands had to be manually sent from the ground. Again the Timeliner developers at MSFC recognized that controller workload could be decreased while increasing operational reliability by creating an autonomous function on the PLMDM that could send the required commands when the MSG rack was powered on as well as the corresponding commands when the rack is powered down. Ultimately the designers created the MSG_STARTUP and MSG_SHUTDOWN bundles. These sent the correct series of commands from the PLMDM based upon the power status of the rack to either start or halt rack data. All of these bundles, although basic with respect to the systems that are utilized today, laid a foundation for the new functionality to come. These were the first of their kind to be executed on crewed spacecraft.

Figure 6 – Microgravity Science Glovebox Rack

The next step was to receive and use this information on board to create more robust bundles, but first more integration problems had to be overcome. To read telemetry or memory, addresses must be shared or known to the Timeliner compiler. A file is generated at compile time, the address map file, which defines the start address, start bit and length to the executor on the MDM. To verify these addresses was a huge task. There was not just one group that had these addresses, and in the end a new process had to be developed to get these addresses. The byte swap issue and Intel - Motorola memory representation was a problem, and is still an issue even today. But these can be overcome

with testing and understanding of where the data is coming from, and how many MDM bus interface adapters the data is going through.

Once the telemetry issues were understood, more complex bundles could be created. But the developers also realized that having many stand-alone complex Timeliner bundles operating on a single MDM would soon become difficult to manage from the ground. Thus, a system was needed onboard to manage these bundles, the Higher Active Logic or HAL concept was beginning to take shape [5].

4. An Onboard Autonomous System – Higher Active Logic

Requirements Development and Testing

The concept for HAL was driven by constraints of bundle management, processor utilization limits, and of course, crew and system safety. Thus the system became "event driven." The architecture of the ISS command and control system is based on locations or devices or Remote Terminals (RTs). Each RT location has a Remote Power Controller (RPC) that determines if the unit is powered. Allowing a master Timeliner bundle (HAL_MAIN) to monitor the status of these RPCs (powered or unpowered), tuned out to be a very reliable and safe method to manage which specific payload bundle should be active. If a particular payload RPC is "open" or unpowered, the associated bundle is removed from memory. Conversely if the RPC is "closed" or powered, the bundle is installed. With this control mechanism in place, autonomous monitoring or commanding is only possible when a RT, or payload, is powered. This addressed one of the primary safety concerns of the ISS program managers [5].

HAL was assigned 1288 words of specific memory locations on the PLMDM to use for shared values. The tools necessary were now in place to make a workable system on board the ISS. At this point in time, Timeliner had the functionality to read telemetry, send commands and then create and update HAL defined memory. The HAL memory was mapped exactly like all telemetry, that is all memory was divided into Payload Unique Identifiers (PUIs) and assigned addresses, start bit, length and data type. This was not only readable memory but writable too. Several different types of data were defined ranging from one bit parameters, integers, floating point, and double floating points to ASCII. This enabled the HAL master bundle to inter-communicate with RT level bundles or share memory.

By using shared memory, HAL was being designed to actually monitor bundle sequences, and execute and report back if the system is operating properly or improperly. Also this can be used to request other services from HAL by designating a service request word. Much could be said about the HAL design and will be covered later. The important thing to note is that specific memory for HAL is being used and maintained by the HAL system.

All of the data in memory HAL is using is being sent to the ground. Therefore, a more dynamic system of monitoring can be accomplished. Each RT that is configured into the HAL system is assigned a suite of memory. Some can be used for HAL monitoring and request handling and some can be used for RT specific functionality. There is memory assigned to each RT that is ASCII and is 60 words long. This is a first in space operations, actual text can be updated and sent to the ground. This enables ground operators to view messages directly instead of using extensive look up tables that are numerous and difficult to maintain.

With a system in place within the frame work of ISS procedures and guidelines, automation can take place. The first bundle to have true automation was the "AP_HAL_PLMDM."

This bundle was to have functionality to recover the PLMDM and monitor all RT(s) for power, H&S, and send the appropriate commands. It was envisioned that the ground operator would only have to install HAL_MAIN and all would be recovered and configured appropriately. It would also continuously monitor all power and H&S from all RT(s) and any subordinate payloads until the HAL system is stopped.

The next step was to prove this would work. HAL 1 was designed and tested to generate messages, in lieu of commanding, to prove autonomous commanding could be achieved. After months of monitoring with no errors HAL2 was developed to send commands. To date HAL3 is running as designed. The ground operator simply installs HAL when the PLMDM is operational and continues on to other flight controller tasks. The HAL system was critical in a recent consolidation of ground controllers; it basically automated a complete ground position [5]. Table 1 below summarizes the progression of the HAL system since its initial loading onboard the ISS in April of 2005.

Timeliner Approval Processes for ISS Operations

The process of getting a new Timeliner Bundle on-board ISS typically takes six to nine months from initial concept to first execution. This is the amount of time required for a detailed review of requirements, development process, testing and review cycle. Initially, the developer presents a concept of operation for the new bundle to the Automated Procedures Control Panel (APCP) at MSFC. The APCP reviews all payload Timeliner concepts for operational feasibility, design and test plans. Upon approval from the APCP, the developer starts the development cycle.

Table 1. HAL Version Functionality

Version and Date of 1st Execution	HAL Functionality
HAL 1 (4/2005)	• Autonomous Bundle installation • Autonomous Bundle halt and removal • Autonomous H&S processing for MSG rack • H&S message for all other payloads (proof of concept) • Ku band AOS/LOS messaging (proof of concept) • Single command function for MAMS science data downlink
HAL 2 (9/2005)	• All HAL 1 functionality • Single command configuration of PLMDM • Autonomous H&S processing for all payload racks & subrack payloads • Shared memory for higher level status downlink • Shared memory for inter-sequence communication • Ku band AOS/LOS written to shared memory • Single command function for enabling & inhibiting Ku High Rate Transfer Profiles • Inter sequence communication for HAL configuration control • Ground request for HAL configuration control
HAL 3 (6/2006)	• All HAL 2 functionality • Autonomous Monitoring for PEHG & APS radiation hit detection • ISS Lab Freezer configuration commanding

Development initially consists of making sure all telemetry and commands necessary for creating the bundle and sequences are available for the Timeliner compiler. Once this has been completed, the first compilation(s) can take place. The Timeliner UIL is unique in identifying compilation errors. These errors are defined as a "cuss" or "cusses". After all the "cusses" have been removed and a clean compilation is made, all of the supporting files are automatically created by the Timeliner compiler. These files are the address map file "bundle_name.tla" and the executable "bundle_name.tlx." Of course, these must be properly byte swapped, if necessary. These must be placed on the PLMDM mass storage device for execution. The Timeliner Executor resides on the PLMDM and reads these files into memory for execution when directed to, by ground commanding or onboard commanding. The facilities that have a Functional Equivalent Unit PLMDM are limited resources. Some examples are Payload System Integration and Verification function, Payload Rack Check Unit and Space Station Training Facility. These facilities and others are limited resources; therefore, they must be scheduled in advance for testing. The scheduling and availability of these resources are a large factor in the time needed for development.

After development, a comprehensive review and approval process is in place prior to executing the new software onboard to ensure the design is safe, does not use more computational resources than allocated, and follows proper operational practices. This process involves multiple organizations including the Payload Operations Center at MSFC, specialists at JSC, as well as experienced Timeliner developers at Teledyne Brown Engineering and engineering elements of the PLMDM vendor to ensure all parties involved are aware of and concur with the new system.

5. Future Applications of Autonomous Systems Onboard Crewed Spacecraft

Intelligence Levels

The Timeliner UIL easily provides a built in software organization that lends itself to intelligence layering. The Executor/Sequence/Bundle packaging of Timeliner can be applied to the multiple layers of status data when the computer architecture is a tiered structure, such as used on the ISS. Sensors and effectors provide status data that can be collected and used as input into specific intelligent software. Each specific programmed function can then provide its monitoring and commanding status. Timeliner sequences provide the low level intelligence with a collection of sequences providing the low level monitoring and command response functionality. The Timeliner sequence status as well as sequence execution status provided in shared memory (i.e. HAL system 2), allows intelligent execution data to be available for "higher" level sequences that are executing in a higher intelligent bundle of sequences. This higher level bundle may be executing within the same processor, or it may be executing elsewhere on the network. The intelligence leveling of bundle functional capability and processors is dependent upon the type of system being autonomously operated. Adding shared data areas for each Timeliner Executor creates the ability for sequences to pass data and request actions of other sequences proven with the ISS payload operations HAL system. The addition of an "activity" intelligence layer of

Timeliner bundles, coupled with a real-time re-planning system specific for the activity, finalizes the overall design of the autonomous operations.

Timeliner sequences at all levels can be controlled by planning parameters broadcasted to memory locations for which functional and activity sequences are programmed. Functional and activity sequences can broadcast their status data simply by writing into shared memory shared by local sequences and broadcast to other processors/executors.

The HAL system 3 utilized shared memory to provide English text status during its execution. The text is displayed and played back during operations, and could be converted to voice. The HAL system 2 also used shared memory for ground operator and inter-sequence requests. The HAL system 2 utilized a strict asynchronous request handler, one request at a time, but each bundle can have one, allowing simultaneous requests within the system. The Timeliner system provides built-in "man-in-the-loop" query, input and confirm capabilities allowing full human interaction during activity execution. This allows "man-in-the-loop" at any execution level and at each execution point in the automated system. Layering Timeliner Executors provides higher level knowledge (status data), and programming to auto-recover at each functional point, reducing the human requirement.

HAL System Shared Memory and Bundle Organization

The HAL system was provided with 1288 words of memory accessible by all bundles within the Timeliner execution environment. Utilizing this shared memory, sequences can monitor "request" words for requests that can be made by other sequences or the man-in-the-loop. Data is also provided with the request to further define the particular request for the executing sequence. Status data is then returned based upon the operation of the request. It is this mechanism that can be employed for activity execution status, re-planning and for providing updates resulting from the re-plan. The device status data is available to all bundles, see figure 7.

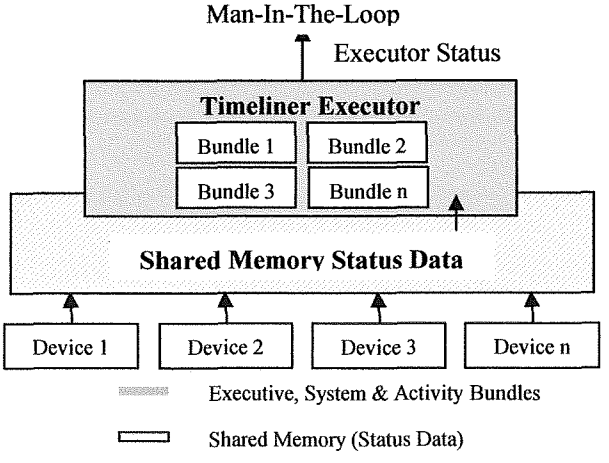


Figure 7 – Single Tier Integrated Timeliner Structure

This allows Timeliner bundles a "Level 1 Intelligence" capability in that each bundle has available to it, the knowledge of each device's status but is "unaware" of any other Executor's status or functional execution status. Higher level activity execution can still be achieved by organizing the bundles with tiered functionality, providing the higher level execution status necessary for an "activity" to be executed and re-planned.

Increasing the levels of intelligence

Expanding the single tier design to a layered architecture, with broadcasting of all lower level data tiers up to higher-level tiers and broadcasting higher-level data down through the tiers, creates a tiered architecture of knowledge and data, which increases the intelligence capabilities at each tier see figure 8. This allows higher level operations functions to be programmed where upper "Executive" bundles have knowledge of all activities and operations being performed at the lower levels.

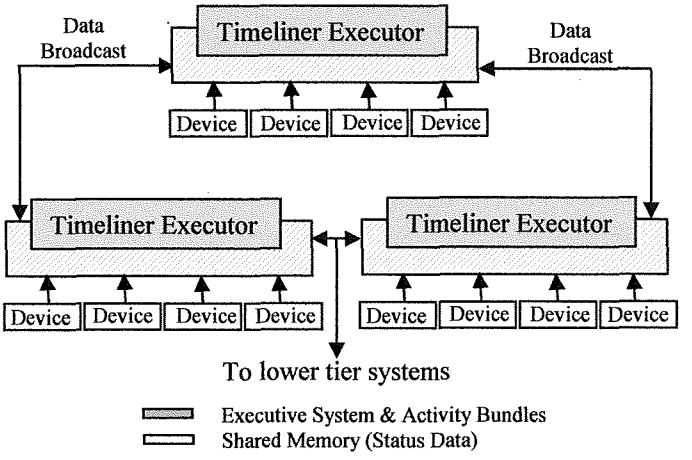


Figure 8 – Multi-Tiered Utilization of Autonomous System

Figure 9 shows a simple conceptual autonomous robotic exploration vehicle. In this example, lower level tiers provide the data for motor speeds, track speeds, bearing temperatures, and other data from the system that provides vehicle movement. The lower tier's command and control is via the guidance, navigation and control (GNC) system [6].

The GNC executive provides a higher level of operations via the Timeliner Executor. The executor at tier 1 provides single command functionality, track system monitoring, autonomous location tracking and navigation. The Remote Manipulator System (RMS) is located at tier 2. Tier 1 data is broadcast to tier 2 allowing the RMS system, knowledge and potential command and control of its location while the RMS is performing an activity. The tier 2 executor controls operation of the manipulator system, again providing single command functionality and autonomous monitoring of the RMS system. The tier 3 (Main Vehicle) executor controls the bundle installation and removal for systems powered or

un-powered and also executes the activity timelines. Tier 3 contains all the status data from the lower level tiers allowing "knowledge" of all lower level systems and devices during execution [6].

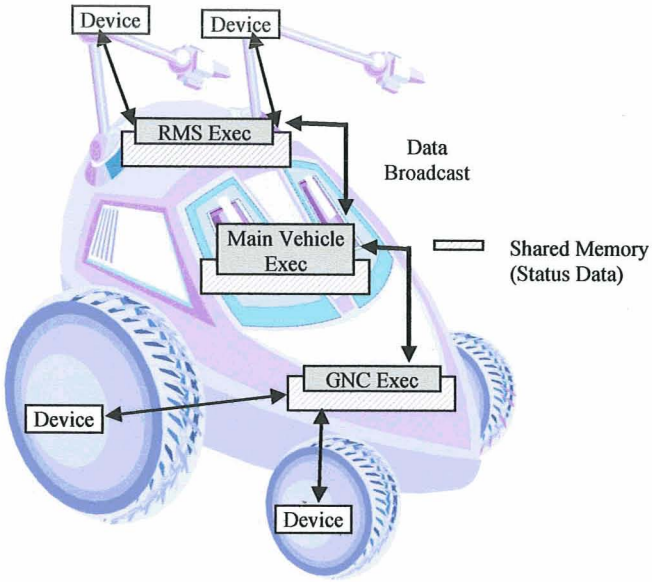


Figure 9 – Autonomous Robotic Vehicle Concept

Real-time Re-Planning the Mission Manager

During sequence execution through the various executor levels, operating parameters are being monitored and utilized for performing local functions or during execution of activity sequences. The return and execution status from all sequences via shared memory and the built in executor/ bundle/sequence status is broadcast to all tiers and also to the "planning engine" driving the automation (figure 11). This allows expert knowledge about the automated target system to be programmed to command and control its operation. The planning engine is programmed to monitor the automation and the target system and execute and monitor "activities" (sequences). Thus, the planning engine becomes the operator of the target. The planning engine is designed specifically for the system being operated and is programmed for the broadcast status data needed to re-plan and broadcast out operational updates for sequence execution at the function and activity level. Intelligent leveling with executors, coupled with expert programming at the various levels, reduces the amount of re-planning when auto-recovery can be achieved more rapidly than with re-planning. The planning engine would know via broadcast status that an auto-recovery was in progress, finished, successful or not successful and may have nothing to re-plan or at a minimum, re-plan only when it had to. The planning engine is broken down by subsystem and activity of the target to be automated. Sub-systems, for example, can be defined small or large, but usually consist of basic power, thermal, communication guidance, navigation, and life support. Each sub-system defined has specific programming for both monitoring, and parameter control of sub-system commanding. In effect, it is a mini-planner. This mini-planner exchanges data between the other sub-system mini-planners as needed [7].

The Activity planner initiates and monitors activities while in execution in the same manner as the sub-system planners. The automated activities command and control the lower sub-system functions and report execution status back to the Activity planner. Data exchange between the Activity planner and sub-system planners allow current status verification of resources, for activity execution. All planning engines contain default initialization data which can be updated as more efficient planning data is realized or as anomalies in hardware dictate a change in plan [3].

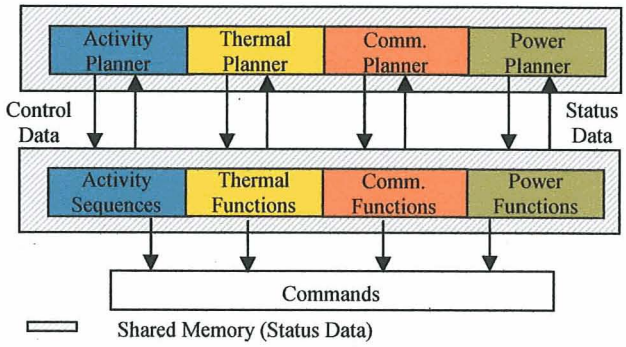


Figure 10 – Mini Planning Engines

Memory interfacing between the planners is further defined for the different support modes the system is executing within as follows:

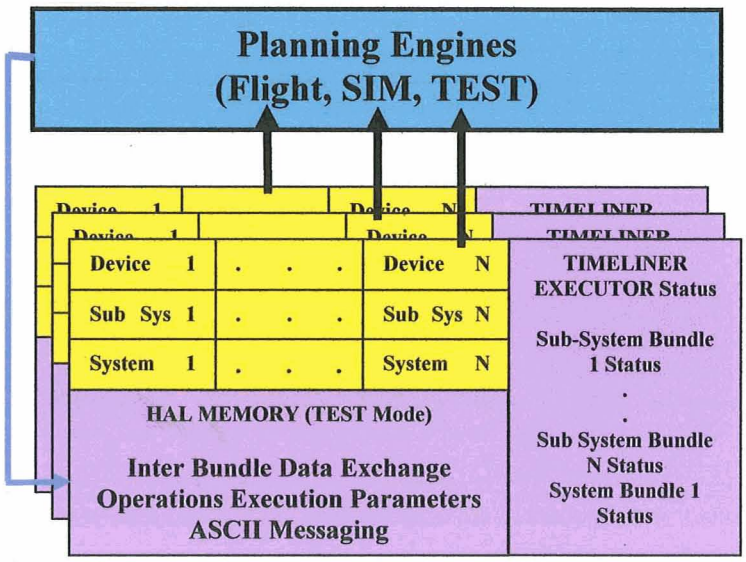


Figure 11 – Planning Engine Memory Utilization

Planning engines have access to the real-time status data of the devices being utilized, the subsystems being operated and the activities operating the subsystems. The front end of the Planning Systems must provide the "knowledge" required for command and control of the devices, subsystems and activities. The HAL 1000 system design utilizes a knowledge based architecture that can be updated as operational constraints and device/system limits are realized in real-time as depicted in figure 12 below:

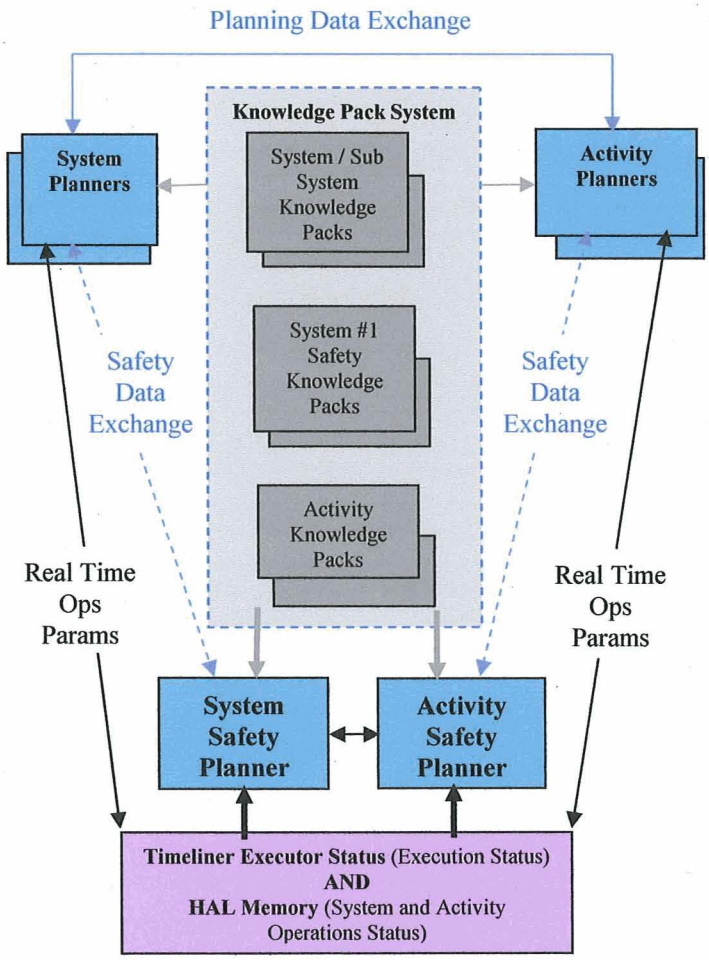


Figure 12 – Knowledge Pack Interfaces

The knowledge packs provide the planning engines with all the device, system, sub-system, safety and activity limits and constraints.

6. CONCLUSION

Science operations onboard the International Space Station to this point, have been tremendously successful. The implementation of autonomous systems, such as Timeliner and HAL, has been a particularly noteworthy accomplishment. Like any endeavor, there have been setbacks, and progress has not come as quickly as first expected. However, reflecting on the three years of Timeliner supported payload operations; it is easy to recognize that great strides have been made with respect to reducing the workload of ground controllers and increasing operational reliability. Specifically, in the area of autonomous systems onboard crewed spacecraft, Timeliner and HAL have dramatically changed the way these types of systems are utilized by providing systems that can model human decisions [5].

Prior to the ISS program, automation within critical systems was minimized in an effort to increase reliability. As discussed in this paper, the first payload Timeliner bundles, albeit simple, proved that operational reliability could be increased. And thus, they paved the way for more complex bundles, such as the HAL concept. The 6-9 month approval process for the HAL concept allows time for development, testing to ensure the design is safe, testing the computer resources on-board, and ensures that operational procedures are followed. Now that autonomous systems have been proven onboard crewed spacecraft, the natural progression would be to increase all levels of automated intelligence and to eventually allow for fully automated real-time re-planning.

With the Space Shuttle fleet returning to flight and the continued construction of the ISS, the hope is that the steady stream of science payloads will continue. Many of these science payloads will take advantage of autonomous operations provided by the Timeliner UIL. In turn, this automation will help earthbound scientists conduct their experiments in the microgravity environment of low Earth orbit.

REFERENCES

[1] Charles Stark Draper Laboratory Timeliner Webpage: http://timeliner.draper.com

[2] Brown, R.A.; Braunstein, E.; Brunet, R.; Grace, R.; Vu, T.; Busa, J.; Dwyer, W; Timeliner: Automating Procedures on the ISS; Space Ops. Conference, Houston, TX, October 2002

[3] Brown, R.A; Automating Space Operations Using Timeliner and Adept; Enhancing Space Operations Workshop, Houston, TX, May 2005

[4] Craig Cruzen, Richard Gibbs, Steven Dyer, John Cech; "Expanding Remote Science Operations Capabilities Onboard the International Space Station", IEEE Aerospace Conference, March 2003.

[5] Howard Stetson; Concept of Higher Active Logic for International Space Station Payload Operations; Teledyne Brown Engineering White Paper; May 2002

[6] Ricard, M; Sauer, B; Autonomous Mission Planning for Spin-Stabilized Science Satellites; Proceedings 11th Annual AIAA/Utah State University Conference on Small Satellites, Logan, UT, September 1997

[7] Howard Stetson; Developing Autonomous Operations of the Future with Timeliner; Teledyne Brown Engineering White Paper; June 2005

BIOGRAPHY

Howard K. Stetson *is the Timeliner Development Lead for ISS Payload Operations and has over 30 years of experience in software development and engineering encompassing numeric intensive computing, code vectorizing, parallel processing, computer graphics, simulation and modeling, computational fluid dynamics, real-time command and control operations, operations automation and software integration and test. Mr. Stetson has produced two white papers for the Marshall Space Flight Center. The Higher Active Logic System (HAL) for the ISS payload operations computer system and the Automated Multi-Purpose Space Operating System (AMPSOS) designed specifically for long duration space missions. Mr. Stetson is currently in design of the AMPSOS concept to support autonomous operations for space and commercial applications (HAL 1000).*

David K. Deitsch *is a Timeliner Developer for ISS Payload Operations and has over 25 years experience in automation and data processing. Before being a Timeliner developer, he served as a Command Payload Officer for the ISS. This experience was instrumental in designing Timeliner applications for automation and consolidation of ground flight positions for the Payload Operations and Integration Cadre. Mr. Deitsch is currently designing the "Next Gen" concept of software tools for ground operations. He holds a Bachelor of Science in Mathematics from University of Alabama in Birmingham and a Master Of Science in Computer Science from Alabama Agricultural and Mechanical University.*

Craig A. Cruzen *is a Payload Operations Director (POD) at NASA's Marshall Space Flight Center in Huntsville, AL, where he leads the ground control team in performing science operations onboard the International Space Station. Before being selected as a POD in 2003, he served as an ISS Payload Rack Officer (PRO) and Timeline Change Officer (TCO). Duties in these flight control positions included monitoring and commanding payload racks and payload support systems onboard the ISS as well as maintaining daily activity timelines. He holds a Bachelors degree in Aerospace Engineering from the University of Michigan in Ann Arbor.*

Angie T. Haddock *is the Operations Lead for the ISS PLMDM. In this role, she is responsible for leading the development of PLMDM operations, planning, preparing the PROs to support PLMDM operations, assessing PLMDM operations, and coordinating troubleshooting activities. Ms. Haddock has also provided civil service oversight and technical coordination for the development and update of PLMDM software and the impact of PLMDM technical changes to operations. Prior to joining NASA in 2000, she worked for Teledyne Brown Engineering where she trained to serve as a Command and Payload MDM Officer (CPO) for the ISS Payload Operations Integration Center. She holds a Bachelor of Science degree in Computer Science from Athens State University.*

ACKNOWLEDGEMENTS

The authors would like to acknowledge the following groups who have contributed to the success of performing remote science onboard the International Space Station:

- The POIC operations team at MSFC who operate science payloads around the clock on ISS, and at the same time plan for future experiments.

- The ground testing teams at MSFC and KSC who work to ensure that hardware and software launched into space is the best it can be.

- The astronauts and cosmonauts who live and work aboard the ISS and risk their lives in the pursuit of space exploration.

- The scientists and engineers who design the experiments and use the results to improve our way of life on Earth.

Autonomous Payload Operations Onboard the International Space Station

Howard K. Stetson
Timeliner Software Development Lead for ISS Payloads
Teledyne Brown Engineering

Paper Authors

Howard K. Stetson
Teledyne Brown Engineering
Huntsville, AL 35812
256-961-0399
howard.k.stetson@msfc.nasa.gov

David K. Deitsch
Teledyne Brown Engineering
Huntsville, AL 35812
256-961-0467
david.k.deitsch@msfc.nasa.gov

Craig A. Cruzen
Marshall Space Flight Center
Huntsville, AL 35812
256-544-8658
craig.cruzen@nasa.gov

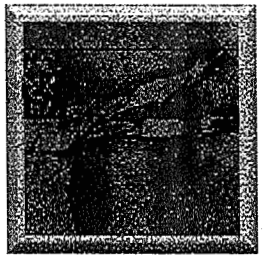
Angie T. Haddock
Marshall Space Flight Center
Huntsville, AL 35807
256-544-6285
angie.haddock@nasa.gov

Outline

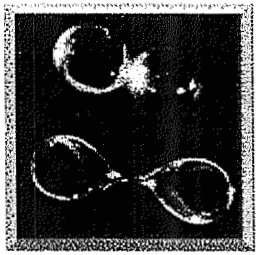
- Introduction
- International Space Station (ISS) Payloads, Payload Operations and Timeliner Overview
- Development and operations of the Higher Active Logic (HAL) Timeliner System
- Potential Timeliner Applications in the Exploration Program
- Conclusions

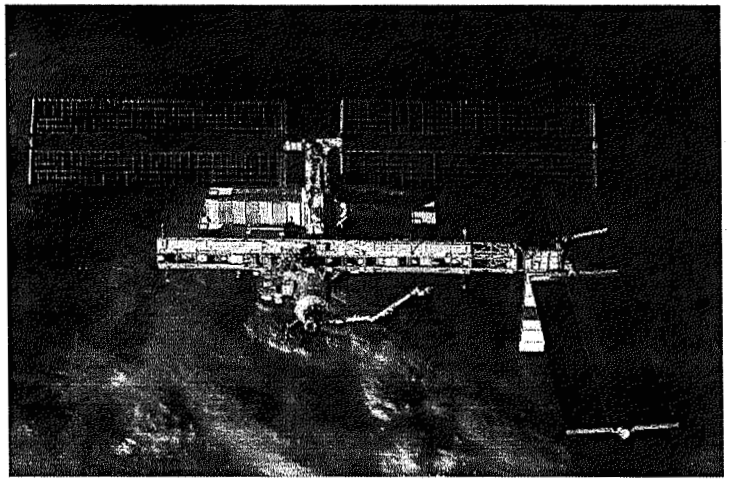
International Space Station:
A World-Class Microgravity Laboratory

Life Sciences

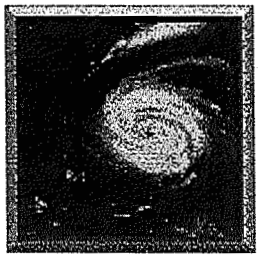


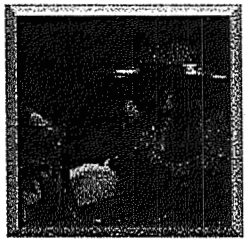
Life Support & Habitation Research



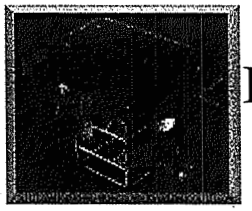


Earth Science





Space Product Development



Engineering Research and Technology

The ISS is a multidisciplinary laboratory, technology test bed, and observatory that provides an unprecedented capability for scientific, technological, and international experimentation in low earth orbit.

The Payload Operations and Integration Center
Marshall Space Flight Center, Huntsville, AL

Flight Controllers in the POIC have conducted ISS science operations 24 hours a day, 7 days a week since March, 2001.

Timeliner User Interface Language

What is Timeliner?

A programming language and execution system that has built in Command and Telemetry interfaces.

Why Timeliner?

- Selected for use on ISS.
- Prior NASA experience on the Shuttle program.
- Complete development and execution system.

What did it give us?

- Decisional based on-board automated command capability.
- Rapid development of autonomous operations.
- Allowed execution status to be programmatically analyzed for higher-level programmatic decision making.

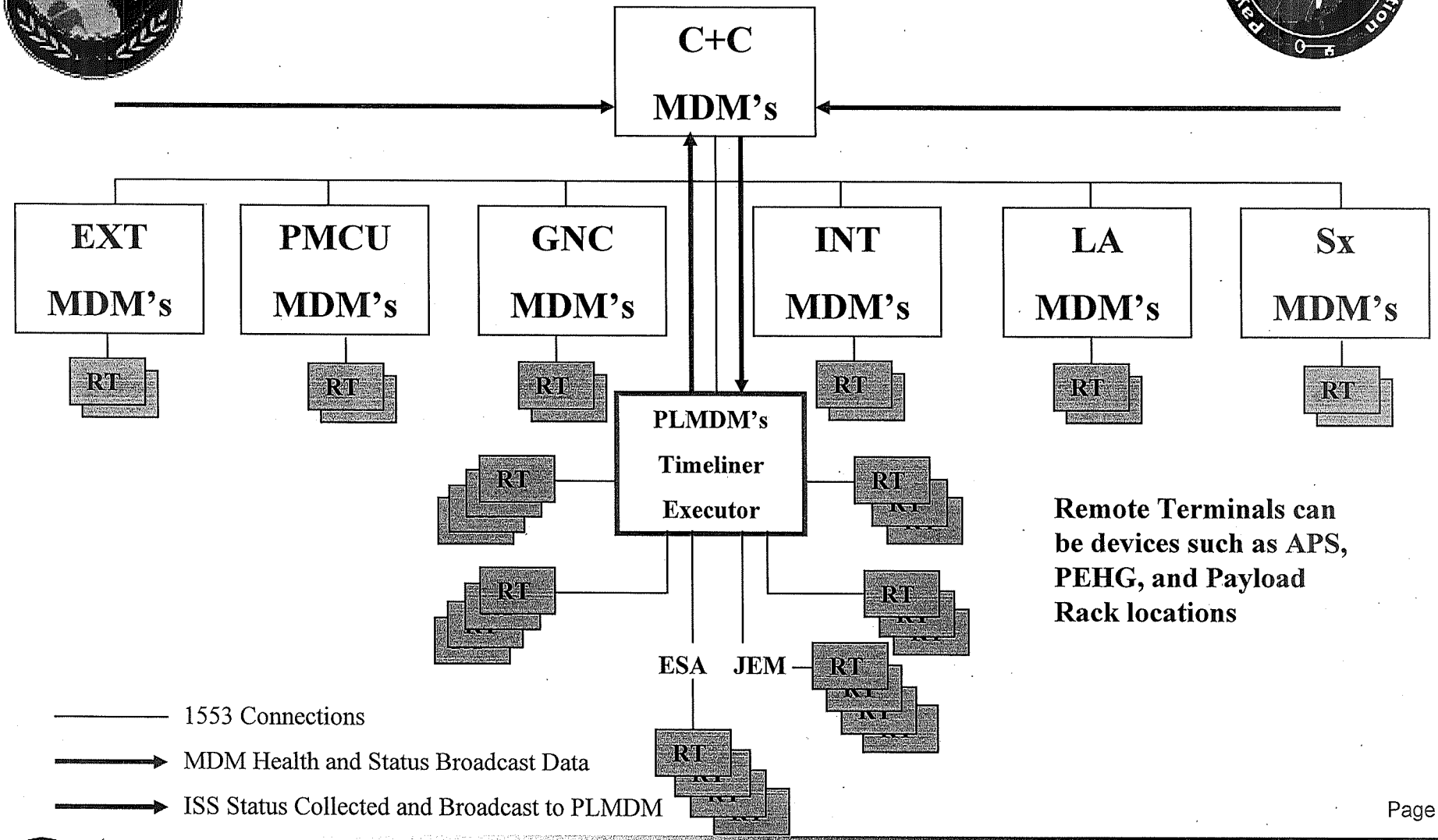
Timeliner System

Two components:

Compiler (Ground Based)
Executor (Onboard Based)

- Compilation involves standard language verification/validation.
- Produces an executable, listing, and memory address map files.
- Compilation resolves command and telemetry references.
- Commands are built into the executable.
- Telemetry is defined as memory addresses, lengths, start bits, and data types.
- The Timeliner Executor executes the software, outputs commands, and reads memory addresses for telemetry values within the code.
- "Programs" are packaged as "Bundles", 10 programs per bundle (executable)

ISS Computer Tier Structure (Sample)



Remote Terminals can be devices such as APS, PEHG, and Payload Rack locations

ISS Computer Tier Structure

- Each MDM collects health and status data from their supporting devices and sub-systems.
- Each MDM in turn, broadcasts the health and status data to the C+C MDM for broadcast and downlink (S-Band).
- A pre-defined collection list if data is broadcast down to the PL MDM.
- Broadcast data is stored in PLMDM Memory and is memory mapped.
- Allows Payload Timeliner situational awareness of system/vehicle status.
- Broadcast data also downlinked via Ku-Band from the PLMDM.

Safety Concerns

After analysis and three separate concepts were designed, the Higher Active Logic System design was promoted. The initial development called for access to power commands restricted from Timeliner access.

Safety issues arise when executing autonomous command and control software within a crewed vehicle.

How do you prevent inadvertent commanding to a remote terminal location while the RT is being repaired or replaced by a crew member? And manual procedures are not deemed sufficient enough for prevention.

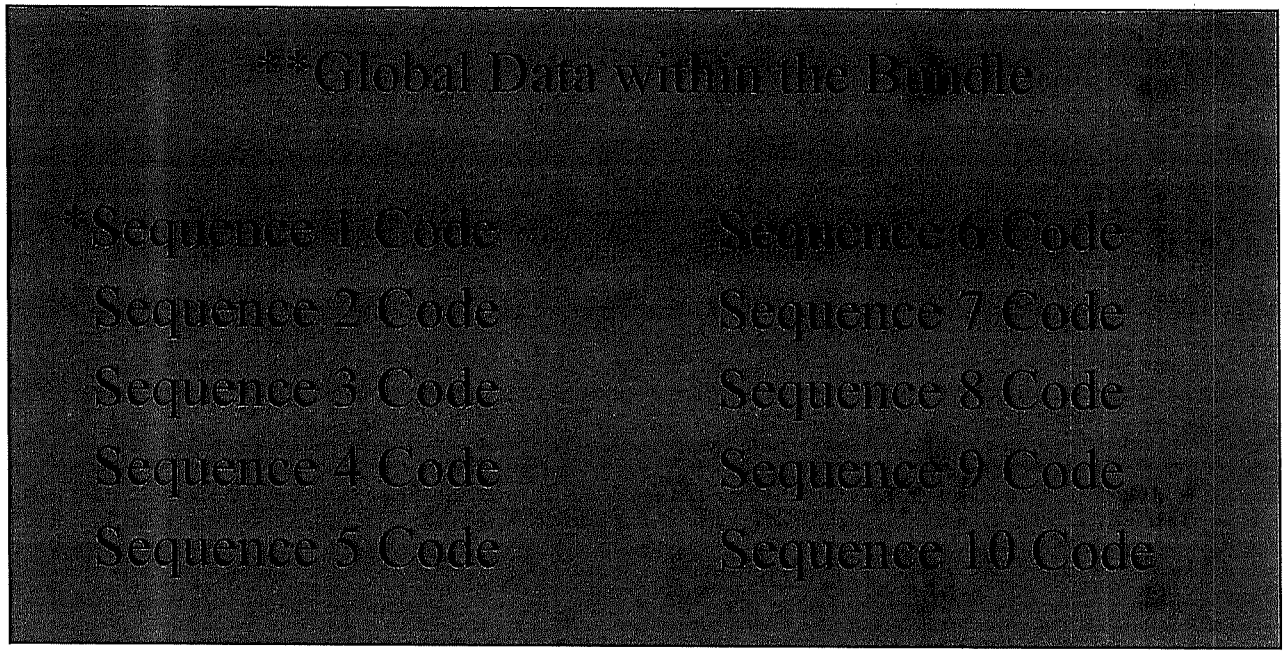
The Higher Active Logic (HAL) System accommodated this by design:

All software installations and removals are performed autonomously as systems are powered and unpowered (event driven) and each remote terminal would be assigned an RT bundle of software.

ISS Bundle Software Structure

Remote Terminal Bundle File

(Basic Building Block)



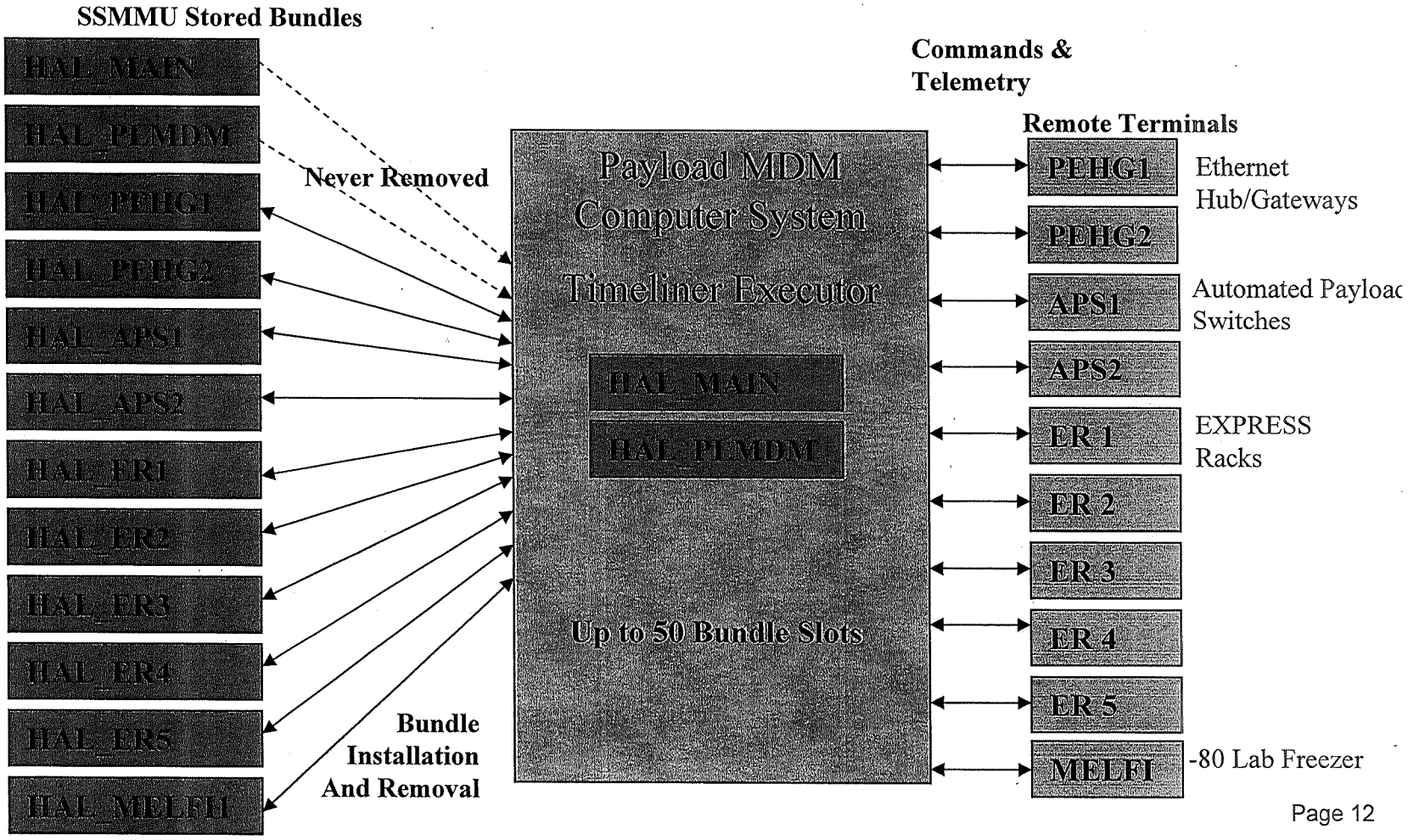
* Sequences are individual executing programs (max = 10 per bundle)

** Global Data is shared between sequences within the same bundle

Bundled Autonomy Software

- Each Remote Terminal on the ISS PLMDM has its own autonomy software (Bundle)
- The HAL System provides for standard functions:
  - Initialization of the RT
  - Safing of the RT
  - Monitoring of the RT
  - Special Functions
- Functions can be manually initiated (Start Sequence).
- Functions can be active automatically upon installation (Monitors).
- Functions can be requested by ground or crew operators.
- Functions can be requested programmatically within the RT's software.
- Functions can be requested programmatically from other HAL's RT software.

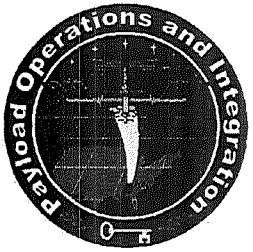
HAL System Timeliner Architecture



HAL System Functions

- Initializes the ISS Payload System with one command
- Installs and removes "autonomy software"
- Performs configuration control of the autonomy level
- Autonomously commands and controls all payload H&S
- Determines and reports the AOS/LOS condition for Payloads
- Monitors and detects radiation effects to the Automated Payload Switches (APS) and Payload Ethernet Hub Gateways (PEHG)
- Initializes, safes, and dumps system parameters for the Minus-Eighty-Degree Laboratory Freezer (MELFI)
- Performs daily downlinks of science data
- Provides monitor and status of autonomous execution (self-check)

Modifications Required To Implement the Higher Active Logic Design



- No changes to the Timeliner System were required.
- Shared memory DRAM allocation where programmatic data exchange could take place

Shared Memory Definition and Usage

- A segment of the allocation used for configuration control.
- Each remote terminal was assigned its segment of the allocation.
- Each segment was defined to provide the following:
  - Incoming Request
  - Request Status Return
  - Last Action Status
  - Last Action Data
  - Last Action Message (60 characters each)
- Each Remote Terminal's software could now take requests from the ground, astronauts or from other RT software. Autonomous actions could now be reported at a higher level and also provide English text status or directions, as well as coded status to operators.

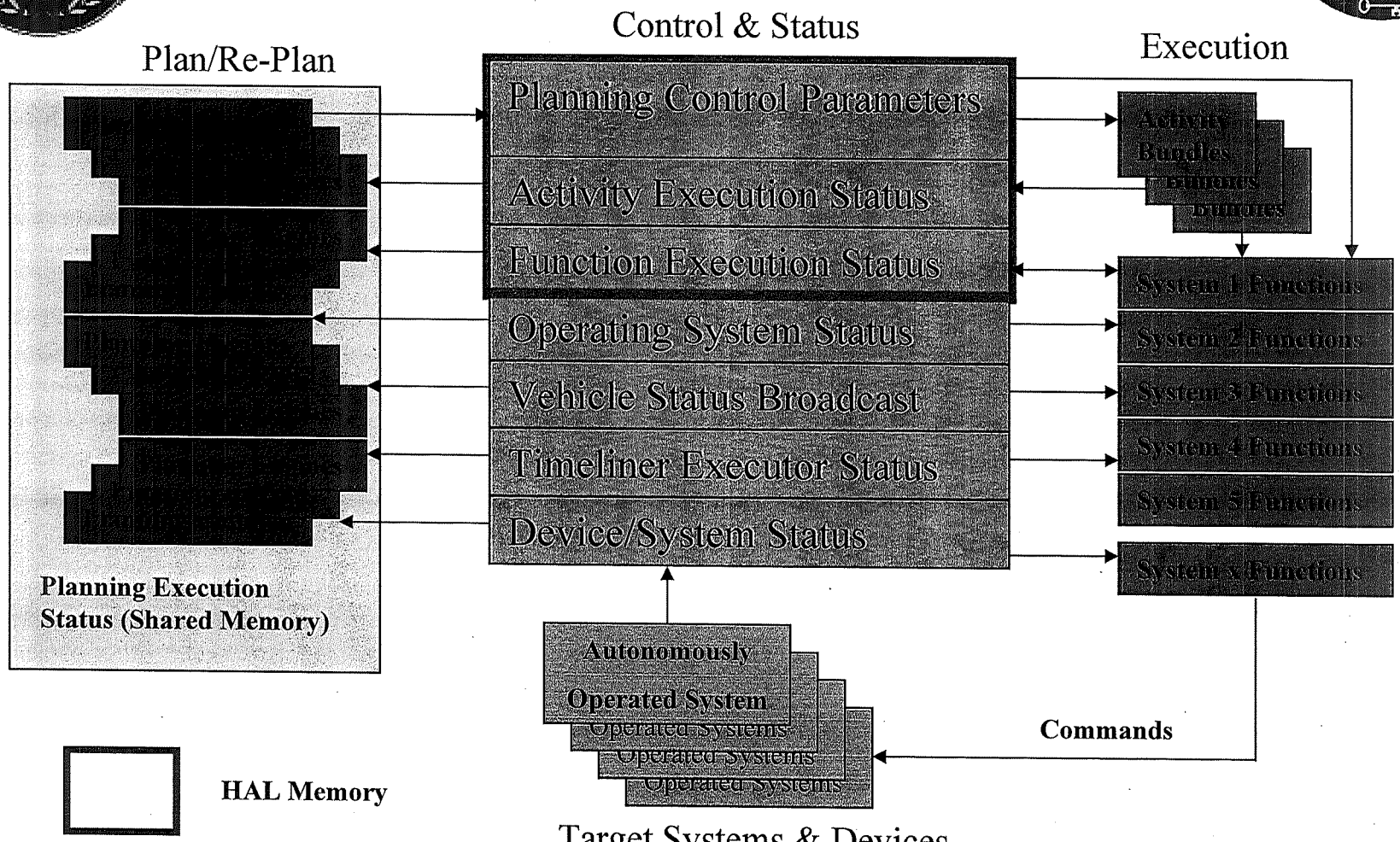
HAL System Configuration Control

- Remote Terminals can be configured into or out of the HAL System.
- Utilizes the HAL Request (Configuration Request) capability
- Determines whether the RT's autonomy software is installed/removed based on the power status of the RT
- Allows an autonomously operated system or subsystem to be operated manually
- Can be expanded to lower-level functions allowing levels of "autonomy" to be defined and configurable in real-time.

HAL System Success

- First to execute autonomous commanding on a manned vehicle.
- Proved the system architecture can meet safety requirements.
- Proved autonomous commanding can be accepted as the norm.
- Proved repeatability in autonomous operations.
- Proved ground operator requirements can be reduced.
- Proved autonomous functionality can be configured and controlled.
- Proved a Higher Operations Layer can be applied over an existing command and control system.
- Proved complex operations can be implemented in a single command.
- Proved automated software can be updated in a rapid fashion.
- Proved a higher-level operational status can be obtained:
  - Leads to autonomous plan execution
  - Leads to autonomous replanning

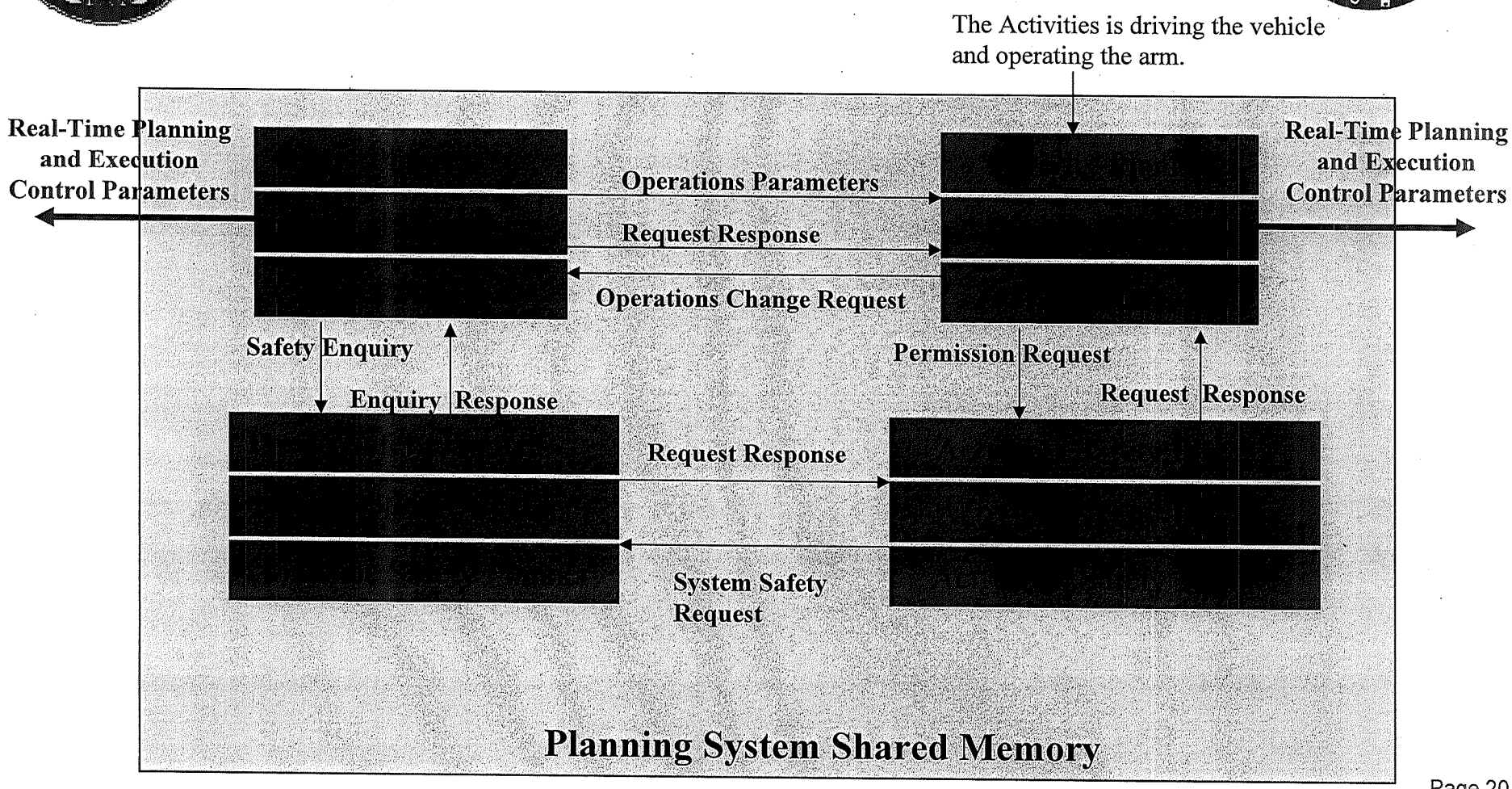
Memory Definition for Plan Execution



Memory Definition for Plan Execution

- Plan an automated activity and execute the activity.
- Monitor - re-plan and affect the automated execution.
- Planning Control parameters specify the execution limits of activities and also the lower-level functions.
- Activity Execution Status provides the details of the current autonomous activities.
- The Planning Engines now have access to all device, system, and execution status required for planning.
- The Planning Engines interface with each other via Shared memory (data exchange).
- The Planning Engines become the operators.

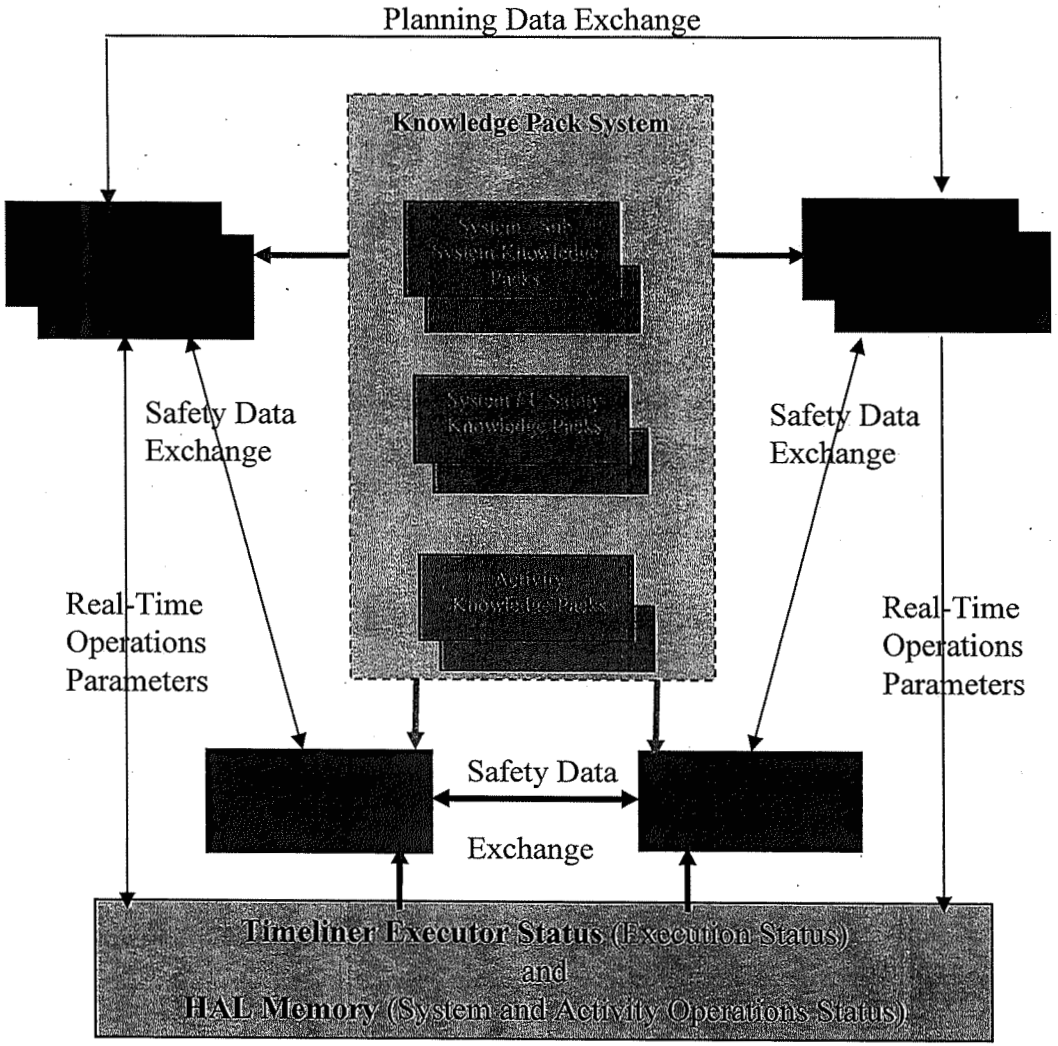
Planning Engine Definition
Robotic Vehicle Example



Planning Engine Definition

- Activity Planners verify with Safety before proceeding with steps within the activity.
- Activity Planners request changes to systems/subsystems.
- Activity Planners monitor the activity and function execution status.
- System Planners monitor the systems status and continuously publish the limits and constraints under which the system is to be operated.
- System Planners implement change requests from Activities upon validation of the request (Re-Plan).
- System Planners verify with Safety before response to real-time events/status and Activity requests.
- System Planners notify Activity Planners of system events
- Activity Safety Planners can interface with System Safety Planners when a system safety check is needed by an activity before proceeding.
- Integrates Safety rules, hazardous condition identification, Flight Rules, and Payload Regulations into the automation.

Knowledge Pack Interface



Knowledge Pack Interface

- Provides device operating environment, parameter identification, limits, and conditions
- Provides subsystem/system parameter identification, operating environment, limits and conditions
- Provides Activity identification, correlates the activity to systems and devices used, and provides activity bundle/sequence identification and successor/predecessor identification
- Provides safety rules, flight notes, payload regulations, and system/subsystem safety limits
- Provides interface point for entering specific planning knowledge (The location knowledge pack is an example)

Summary

The Timeliner system has been used to provide an intelligent operations layer on top of the existing command and control system On-board the ISS.

The Timeliner system has been used to provide on-board operations command and control to replace remote operations.

The Timeliner system provides the versatility to execute autonomous operations on the ground and then migrate the autonomous operations to on-board.

Opens the door for planning/re-planning of autonomous execution.

Acronyms

AOS	Acquisition of Signal
ANITA	Analyzing Interferometer For Ambient Air
APCP	Auto-Procedures Control Panel
APS	Automated Payload Switch
ASCB	Avionics Software Control Board
C+C	Command and Control
DRAM	Dynamic Random Access Memory
ECR	Engineering Change Request
ESA	European Space Agency
EXPRESS	Expedite Processing of Experiments for Space Station
HAL	Higher Active Logic
H+S	Health and Status
ISS	International Space Station
JEM	Japanese Experiment Module
LOS	Loss of Signal
MELFI	Minus-Eighty-Degree Laboratory Freezer
OCR	Operations Change Request
PEHG	Payload Ethernet Hub Gateway
PLMDM	Payload Multiplexer / Demultiplexer
PODFCB	Payload Operations Data File Control Board
PSCP	Payload Software Control Panel
RT	Remote Terminal
TDRS	Tracking & Data Relay Satellite
TORP	Timeliner Operations Review Panel