The Space Congress® Proceedings

1991 (28th) Space Achievement: A Global Destiny

Apr 24th, 2:00 PM - 5:00 PM

Paper Session II-B - The Launch Processing System of Tomorrow

Gerry Duggan
Harris Space Systems Corporation, Rockledge, FL

Shaman Mullich
Harris Space Systems Corporation, Rockledge, FL

Charles Westerfield
Harris Space Systems Corporation, Rockledge, FL

The Launch Processing System of Tomorrow

Authors: Gerry Duggan
Shaman Mullick
Charles D. Westerfield

Harris Space Systems Corporation
P.O. Box 5000
Rockledge, Fl 32955
(407) 633–3800

ABSTRACT

Preparation and launch operations on the Space Shuttle and its payloads are highly complex. The current Checkout, Control, and Monitor System (CCMS), installed in the mid–1970s to provide prelaunch testing, launch sequencing, and control of the Shuttle, is now approaching the end of its useful life. To meet the increase in launch processing requirements for both the Shuttle and Space Station, NASA has responded to the need to replace this aging system with a new system which incorporates the advantages of modern state–of–the–art real time computers, displays, software, and communications. This effort, known as the Core Electronics System, is being implemented by a NASA/Harris team. The objective is to develop a Generic (or Core) test system, applicable both to Shuttle launch processing and to Space Station integration and test, which will also serve as the basis for future NASA test systems. The Core project will replace the CCMS at KSC and, in parallel, develop and install the Test, Control, and Monitor System (TCMS) for the Space Station. The Core System will serve space exploration well into the twenty–first century. This paper discusses the Core architecture and the benefits it provides to the space community.

THE LPS OF TODAY

Progression of LPS Capability Over the years

Launch processing technologies have evolved considerably over the last 30 years. Table 1 shows the progression from early to current to future launch needs and capabilities. Preparation for launch of the Space Transportation System (STS) is currently accomplished through the use of the Launch Processing System (LPS). The LPS was developed in the mid–1970s and evolved from the need for a rapid Shuttle turnaround to meet the projected launch rates. The operational goal of the LPS is a successful launch. All test activities are focused on providing safe, trouble–free launches while maintaining schedule and avoiding costly delays.

Table 1 The Progression of Launch/Checkout Capability over the last 30 years

Chronology	Examples	Mission Attributes	Test Architecture
Yesterday (1960/70's)	Early missile launches	Single test article, sequential test	Centralized computer, dumb terminals
Today (1970/80's)	Shuttle/payloads	Single Test article, parallel tests, rapid launch turnaround	Distributed computers, dumb terminals, shared network
Tomorrow (early–1990's/ 2000's)	Space Station/Shuttle	Multiple test articles, independent parallel test, multi–stage integration, rapid reconfiguration and turnaround	Pooled Computers, smart/graphics terminals, shared networks, reconfigurable resources

Today's LPS

Today's LPS consists of three elements: the Central Data Subsystem (CDS), the Record and Playback Subsystem (RPS), and the Checkout, Control, and Monitor Subsystem (CCMS). The CDS provides data management, test application software development, and system build functions for the LPS while supporting a program library, historical data, and pre– and post–test analyses. The RPS provides the mechanism for capture and playback of all unprocessed instrumentation data. The CCMS commands and monitors the vehicle and Ground Support Equipment (GSE) during launch processing operations.

The current CCMS has served NASA well during its checkout mission, but there are some imminent needs for launch processing which the CCMS does not meet. These include: a shorter turnaround time for test program development, the checkout of Space Station Freedom articles, simultaneous checkout of multiple test articles, and hence, improved test performance and increased system capacity. Modification of the current CCMS is not prudent due to its limited reconfigurability, custom software, obsolete hardware, and the limited responsiveness of its Test Build process.

Lessons Learned

The LPS was technologically advanced when delivered and NASA has learned many valuable lessons from their launch checkout experience on today's system. To promote long life, the architecture should be open, based on industry–wide standards (which were not mature at the time of the LPS development), and implemented with generic, modular components that can be used to support any launch or checkout mission by the addition of mission–unique interfaces. The number of pooled modular elements at a facility should be selected to reflect the maximum number of concurrent tests needed to support schedules at that facility. Multiple concurrent tests should be supported through the assignment and configuration of pooled modular elements to serve a particular test. As another lesson learned, the design approach must be geared toward satisfying the end users by including them as a part of the design team, keeping them involved throughout the entire development cycle, and embedding into the design an understanding of their prime objective: to checkout, control, and monitor test articles in a safe, accurate, and timely manner.

THE LPS OF TOMORROW

NASA has defined a new checkout system. This new LPS retains portions of the old LPS, while defining a new CCMS (CCMS–II) to retain the best of the LPS and eliminate or reduce the recognized limitations. CCMS–II includes generic checkout capabilities, an open architecture, reusable hardware and software, easy reconfiguration of hardware or software, industry standards applied throughout, and a modular design. This architecture takes advantage of the common system requirements for the STS and Space Station checkout. Commonalities include: a Human Computer Interface (HCI); the ability to run automated test procedures; acquisition, distribution, storage, recording, and playback of real time measurement data; and configuration control, fault isolation, and status determination of any equipment.

This new LPS definition is the basis for the Core project being engineered jointly by a NASA/Harris team. The Generic System represents a common "core" upon which unique test and checkout systems are built. The Core Project defines two systems for checkout: the CCMS–II for STS activities and the Test, Control, and Monitor Subsystem (TCMS) for Space Station activities. These two systems and each future system will be built from the modules of the generic Core architecture.

Based on lessons learned on the LPS of today, the LPS of tomorrow will accommodate improvements in the following areas: (1) multiple missions, (2) growth and change, (3) multiple concurrent testing, (4) management of shared and subset resources, (5) multiple users, (6) system performance, and (7) missing element simulations. Each of these improvements are described individually in the following sections.

ACCOMMODATE MULTIPLE MISSIONS: GENERIC SYSTEM CONCEPT

Experience gained by NASA/KSC on past programs such as LPS, Generic Checkout System (GCS), and Partial Payload Checkout Unit (PPCU) confirmed the notion that launch, prelaunch, and postlaunch test/checkout systems share a large measure of sameness in the functions they perform. And further, generic modular components of hardware and software can be developed to perform these common functions. Having the generic components available as a base, mission–unique components can then be integrated into future launch processing systems to meet the special needs of the mission. Harris Space Systems Corporation (HSSC) is implementing the generic concept to be extended into two unique systems: (1) CCMS–II, a replacement for CCMS of LPS for the Shuttle Program, and (2) TCMS, for the Space Station Program. Once accomplished, ***future system applications can avoid the development cost for the generic components*** already implemented on the Core project. Figure 1 depicts the generic system concept.

Generic modular components are organized around interconnecting networks to provide generic test/checkout services for the mission. Any test/checkout system can be realized with some combination of the modular components shown.

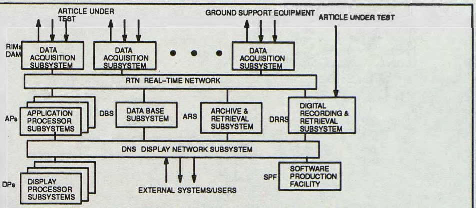

FIGURE 1 GENERIC SYSTEM ARCHITECTURE

The Data Acquisition Subsystem (DAS) contains Data Acquisition Modules (DAMs) and Remote Interface Modules (RIMs) that interface directly with the Test Article and/or GSE, acquiring and processing data from the source in the downlink direction and through–putting commands in the uplink direction. ***Mission–unique interfaces are satisfied by the addition of special interface cards to the generic DAMs.***

The Real Time Network (RTN) provides data and command routing and storage services that interconnect DAMs to Application Processors (APs) and Archival Recording Subsystems (ARSs). As the name implies, the APs process user–written test application programs that analyze downlink data and generate uplink commands. ***Mission–unique data and command processing are thus contained in the Test Programs themselves.*** Multiple application program languages are supported by the generic system including GOAL, UIL, Ada, C, and LISP.

The ARS records the processed data for both on–line and post–test retrievals and analyses. Terabyte storage is achievable through the use of optical disk platters. Data thus recorded is physically mounted in "jukeboxes" within the Database Subsystem (DBS) where it may be retrieved by requesting users and operators. The Digital Recording and Retrieval Subsystem (DRRS) records and retrieves raw (unprocessed) data directly from the test article interfaces.

Users and Operators gain visibility and control over data processing and retrieval operations through Display Processors (DPs) interconnected to APs and ARSs via the Display Network Subsystem (DNS). As with test application programs, ***the users define mission–unique graphical displays for dynamic viewing of data associated with or analyzed by the user–written test programs.***

ACCOMMODATE GROWTH AND CHANGE

A major lesson has been learned on past programs: ***technological advancements cause rapid obsolescence of commercial products.*** Supplier support often wanes within a few years of introducing an "exciting new product" in favor of their latest and greatest offering. In order for future systems to yield a 30–plus year life, NASA/KSC has added to the generic system concept some antidotes against the effects of obsolescence, growth, and change:

a. A multi–network architecture to accommodate the interconnection and future addition of modular system elements

b. Observance of open system standards to accommodate the use of heterogeneous products for hardware and software implementations

c. Allocation of system functionality to generic physical modules that can be added to the system as needed to accommodate present and future expansion needs

d. A layered software architecture to augment the open system standards to further isolate the impact of change to the layer boundaries

e. Sufficient performance margins to accommodate both design tolerance and growth.

Multi–network Architecture and Open System Standards

Figure 2 depicts a template of the multi–network architecture.

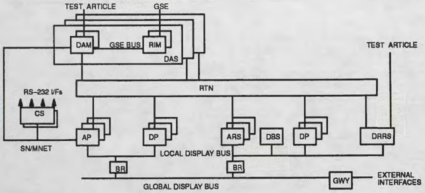

FIGURE 2 MULTI–NETWORK ARCHITECTURE

The RTN is a custom design patterned after the existing Common Data Buffer of the LPS with extended performance to accommodate 16 independent concurrent tests and extended porting to permit up to 256 subsystem interconnections.

The DNS has several network components: (1) an Ethernet local display bus to accommodate local clusters of APs, DPs, ARSs, and DBSs; (2) a Fiber Distributed Data Interface (FDDI) global display bus to interconnect the local buses; and (3) various bridges (BRs) and gateways (GWYs) to interconnect internal and external networks.

The SN/MNET is an Ethernet service network (SN) and maintenance network (MNET) that provides file transfer services and accumulates maintenance–diagnostic data for front–end subsystem elements (DASs). Attached to this network are multi–port communication servers (CSs) having RS–232 interfaces into all system modules for running diagnostic tests from a central operational position (a designated AP/DP).

The Ethernet and FDDI networks provide IEEE standard link–level communication protocols supported by TCP/IP upper level protocols for the near term, migratable to Open Systems Interconnect (OSI) standard protocols in the future. For the customized RTN, standard interface protocols are provided using TCP/IP (later OSI) for the upper levels and direct interfaces at the link level.

In addition to the open system communication standards, the Generic System also specifies the use of the IEEE POSIX Operating System for all commercial general purpose computers in the system.

Modular Additions of Heterogeneous Elements

Heterogeneous computing elements can be added and combined in the multi–network architecture so long as each element observes the open system network and operating system standard protocols. Further, ***as a heterogeneous element becomes obsolete, it can be replaced with a technologically superior version to meet future demands without major rework to salvage its application suite.*** Thus, the open system architecture assures vendor independence and provides for growth, expansion, and interchangeability of elements. The result is a system with considerably extended lifetime and reuse to serve other mission environments.

Layered Software Architecture

To further assure planned growth of the system, the software architecture is organized into four layers to insulate against the rippling effects of changes to the software. The layering is depicted in Figure 3.

The layers are separated by standard interface boundaries used to communicate between layers. Thus, ***software changes that may occur within a layer are inhibited from propagating change to its neighboring layers*** because the interface between them has been formalized and standardized.

User–written, mission–unique application software resides at the top layer (layer 1). Typically this layer consists of test programs, simulation programs, or test data analysis programs. At this layer, the user–programmer need have no knowledge of the inner workings of the generic system architecture, relying solely on the standard interface calls to the next lower layer for gaining access to system services.

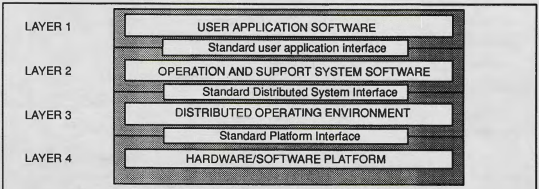

FIGURE 3 LAYERED SOFTWARE ARCHITECTURE

The next level down (layer 2) contains the generic system environment provided by the Core system for user development, operations, analyses, and maintenance activities by which the user and/or associated application programs manipulate the test article. This layer does require knowledge about the functionality of modular elements of the Core architecture, since measurement, command, and link tables need to be built and executed in the Core environment by this layer. However, this layer need not concern itself about the distributed nature of the architecture and the mechanism for inter–process communications and associated distribution of data. This is the domain of the layers below it.

The next level down (layer 3) provides the distributed operating environment that manages the multi–network architecture and provides transparent interprocess and inter–element communications and data distribution services to the layers above it.

The lowest layer (layer 4) provides transparency to the peculiarities of each hardware/software platform to the upper layers, providing a standard POSIX compatible interface, and the vendor specific operating system software plus extensions.

Performance Margins

To accommodate growth, ***sufficient performance margins must be added to each modular system element to account for both design growth and future expansion needs.*** The plan on Core is to arrive at the margin requirements through successive estimation refinements, starting with mathematical analysis, continuing with simulation/modeling in selected areas, and finally through "proof of concept" (POC) testing and evaluation. The first two refinements provide enough confidence to merit initial vendor evaluations and selections. After the initial selections are made, a POC system is implemented to validate system performance and ensure sufficient performance margins for both design and expansion.

ACCOMMODATE MULTIPLE CONCURRENT TESTING

Core will deliver up to 30 sets of equipment over the life of the project. A set is bounded by the equipment attached to an RTN, a pool of modular resources that can be allocated and configured to form test subsets. Core will provide a resource configuration function which will allow the execution of up to 16 concurrent independent tests. The resource allocation and associated test subset partitioning example shown in Figure 4 demonstrates a set having multiple test subset allocations. The resources interconnected by bars with common shading make up a test subset.

Formation of a Test Subset

A Subset Manager forms a test subset by selecting modular subsystem elements from the pool of unallocated resources. Once allocated to the new test subset, the subset is ready to be physically configured into an operational test. This occurs automatically when the Subset Manager gives the system a command to configure the test subset. All associated subset resources are then address–linked and downloaded from the DBS with operating and test execution software. Following subset configuration, an operational readiness test (ORT) is run to ensure the operational readiness of the test subset to perform its assigned mission. The test subset then stands ready for test execution (or test verification) as initiated by the Subset Manager under direction of the Test Conductor. ***A set is specified to accommodate up to 16 test subsets executing concurrently.***

Test Execution

Test execution functions include data acquisition, processing, and recording; closed loop processing; exception processing; reactive sequence command execution; and equipment monitoring. Test data may be distributed to external systems

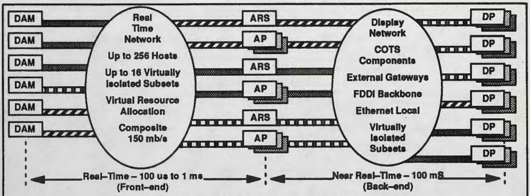

FIGURE 4 PARTITIONING THE SET INTO SUBSETS

and requests for information received and serviced. The safing and monitoring of the orbiter, payloads, and support equipment are also provided. For TCMS, the Test End Item may be the actual Space Station elements and/or a simulation of its missing elements.

PROVIDE MANAGEMENT OF SHARED AND SUBSET RESOURCES

Operation and maintenance management tools in the Core architecture support a hierarchical structure shared throughout the system but concentrated at the Set/Facility level and Subset level. The Set/Facility manager is in charge of shared resources and associated Network Management which has standard tools to report network health and associated statistics. The Subset Manager controls allocation of resources to support the management of backup and redundant resources. There are several management tools which overlap both levels of management including System Integrity and System Maintenance.

Set/Facility Management

Set/Facility management is controlled by a designated AP and DP pair that manages all shared or unallocated resources in the facility. A facility is defined as a collection of sets such as Firing Rooms 1, 2, 3, and 4 and the other sets interconnected by the LC–39 network. Shared resources include the networks, external interfaces, and Data Base Subsystem (DBS). This function performs all facility level management, client administration, and subset load/initialization/shutdown coordination with subset managers.

Network Management. The network management function provides network configuration, initialization, performance monitoring, fault monitoring, and maintenance capabilities to the Core System. The Core network design integrates management standards and protocols embedded in the network components into an operational network management system. ***The Network Manager provides centralized control of network operations, using industry–wide management standards that permit heterogeneous network elements to interoperate.***

Subset Management

Each subset monitors and maintains the operational readiness of its own configured resources. The Subset Manager coordinates the load of subsystem software with the DBS and coordinates the initialization of all subsystems. Integrity is maintained by monitoring the health counts for all subsystems (including redundant pairs).

Reconfiguration and Redundancy Management. ***The functionality is provided to reallocate and reconfigure resources in the event of failure or to invoke redundant elements that have been preallocated to a test subset.*** When a failure is detected, the Subset Manager coordinates manual switchover activities or monitors automatic switchover activities. If a redundant pair has been configured, the Redundancy Management function coordinates the switchover from prime to backup (manual invocation) or from active to standby (automatic invocation); if an auxiliary unit has not been preallocated, a resource may be drawn from the available set pool into the subset. When the switchover or reconfiguration has been completed, all subsystems are notified of the configuration changes via system messages.

System Integrity

Health and Status information is collected, recorded, displayed, and processed by the System Integrity function residing on an AP to maintain and monitor allocated resources in the subset. Health and Status collection involves generating and transmitting any status changes, configuration changes, errors, or faults at any level of the subset hierarchy. The data is recorded with a time stamp and resource identification and then reported. Health and Status data may be displayed by any interested client in a hierarchical manner. The data is analyzed for faults and a determination is made whether or not the fault is recoverable and notification is sent to the appropriate client. System Integrity itself is redundant in that two identical copies reside on two separate hardware platforms.

System Maintenance

The System Maintenance function provides the capability to support fault detection, fault isolation, troubleshooting, and trending analysis. These functions are provided on-line or off-line and are available from the subset down to the card level. System Maintenance provides three classes of functions: Subset ORT, Health and Status Analysis, and board level testing. The ORT function is a hierarchical suite of tests verifying that the subset performs as a unit prior to test execution. The Health and Status analysis function retrieves all recorded data and provides a failure history of all hardware resources, including the sequence of events leading up to a failure. This analysis will provide data to aid preventive maintenance scheduling.

ACCOMMODATE MULTIPLE CLIENTS

Lessons learned in the past mandate the early involvement of the end users (Clients) in the system development process. A system is much more likely to be successful if it gains a priori acceptance by the "Clients." The Core System serves a variety of Clients who interact with the system from several different orientations. In order to effectively serve their needs, a detailed study of their job roles and associated tasks was conducted resulting in a successive partitioning hierarchy by orientation, task category, type and classification. Table 2 lists these categories, along with a concise definition of the tasks performed by the various Clients of the Core System.

Table 2 Client Definitions

Client Name	Orientation	Task Category	Definition
User	Test	Preparation	Preparation consists of test article data base ingest and modification. Preparation also includes interpretation of test requirements, development of test procedures and user application programs, definition of required resources, and building of test configurations.
		Execution	Execution consists of control and supervision of test operations
Operator	Support	Administration	Administration consists of account and privilege maintenance, configuration management, and maintenance of system data bases.
		Operations	Operations consists of management and control of Core system resources. Operations also includes preparation and monitoring of subsets in support of testing.
		Maintenance	Maintenance consists of failure analysis, fault isolation, recovery, and preventative maintenance.
Auditor	Process	Assurance	Assurance consists of ensuring the integrity of all system, support, and test functions.
Developer	System	Development	Development consists of the initial design and implementation of the Core system by the CEC.
		Sustentation	Sustentation consists of modification and enhancement of the Core system after delivery.

The Core Human Computer Interface (HCI) is evolving to satisfy the individual preferences and expectations of these Clients. A series of HCI prototypes were constructed based on Client preferences, ranging from controlled interaction structures for the Test Execution environment to more flexible windowing environments for Test Developers. These prototypes were incorporated into the POC evaluations, and iterated based on feedback from the Clients. From these iterations, an HCI design responsive to client needs is being defined. ***The net result will be a superior user interface design that satisfies a wide range af clients while still accommodating their individual needs.***

IMPROVE SYSTEM PERFORMANCE

The LPS of tomorrow will provide improvement in system performance and operational efficiency for both the Test Development and Test Execution Environments.

Test Development Environment

The Test Development Environment incorporates the concept of segmented builds within a test configuration. That is, the total system software build is partitioned into reasonably independent segments during the development phase, with each segment being developed, compiled, and built separately. These segments are integrated into a total System Build when the system is configured for the Test Execution phase. Basically, ***this breaks the total system build process down into much smaller building blocks, thus improving the turnaround time for any particular build segment.*** Once loaded into the Execution Environment, the test configuration build can be edited on–line to make last minute additions, deletions, or modifications to build segments without requiring the rebuilding of the entire test configuration.

Test Execution Environment

The Test Execution Environment incorporates performance improvements in the real time acquisition and distribution of data. DAMs are specified to process 10,000 measurements of polled data per second as well as PCM data streams operating at one megabit per second. ***Total throughput for a test configuration can be 50,000 measurements per second, but the RTN will handle up to 16 times this throughput to accommodate 16 concurrently running test configurations.*** In addition, the RTN will provide a latency of no more than 1 millisecond fully loaded and 250 microseconds lightly loaded.

PROVIDE SIMULATION OF MISSING ELEMENTS

TCMS for the Space Station Program has unique and challenging requirements for real time simulation. Since the space station is to be assembled on–orbit in space, the test–checkout mission extends the concept of simulation from just "program verification" to include "missing element simulations." That is, ground test and checkout of the present launch element in the Space Station assembly sequence must include the capability of simulating the missing elements that are either already on orbit (past launch elements) or will arrive on orbit later (future launch elements). The concept calls for government furnished Flight Equivalent Units (FEUs) to simulate the missing flight processor elements and for TCMS–supplied processors to simulate missing sensors and effectors that drive the FEUs. Thus, while the Test Environment is checking out the present launch element, the Simulation Environment is simulating missing elements having real time interfaces with the present launch element. ***The TCMS physical architecture to support simulation is identical to that which supports test,*** with the following distinction: as the TCMS simulation processors are executing user–written simulation programs, the TCMS test processors are executing user–written test programs. As might be expected, a high degree of software commonality exists between the test and simulation environments. The same repertoire of languages are supported for developing and executing test or simulation programs. Because of functional commonality, ***the software support services provided for the development–build process as well as the execution process incorporate common software components for both the simulation and test environments.***

SUMMARY

Launch processing requires real time data acquisition, monitoring, and control technology. As new launch vehicle and test articles are developed, the checkout system must adapt. The Core generic, open system architecture reduces system complexity and allows for efficient management of equipment configuration, easy reconfiguration, expandability, upgradeability, and reusability for new missions. Each component is selected and designed to provide sufficient margins for future system growth and to reduce the cost of ownership through improved operations and maintenance. Commercially available components are being identified, evaluated, and selected in all feasible instances in order to reduce program cost and risk. ***The LPS of tomorrow will be a superior system, having benefited from lessons learned on the current LPS. The Core System is a leader in this new way of building launch checkout systems. This new approach is vital to achieving NASA's goal of a safe, reliable, cost effective, operationally efficient launch system with a life expectancy of 30–plus years.***.