May 1st, 2:00 PM - 5:00 PM

Paper Session II-A - Operations and Maintenance Requirements Specifications - Automated Buy-Off System

Todd Flato
Riptide Software Incorporated

Barry Rubel
Riptide Software Incorporated

Operations and Maintenance Requirement Specification – Automated Buy Off System (OMRS-ABOS)

By

Todd Flato
and
Barry Rubel

Riptide Software Incorporated
P.O. Box 360857
Melbourne, Florida 32936-0857
todd.flato@riptidesoft.com
(321) 773-9233 Office
(321) 779-9943 Facsimile

Problem Domain

Project Engineering is the Kennedy Space Center (KSC) organization responsible for monitoring the progress of milestone-closure as part of Space Shuttle processing, also known as a flow.

An Operations Maintenance Plan (OMP) is a plan used to process a Space Shuttle as it is in a flow for a single mission. Each requirement-line-item in an OMP is known as an Operations and Maintenance Requirement Specification (OMRS). Each OMRS must be verified and closed separately by a configuration management person and an engineer.

An Operations and Maintenance Instructions (OMI) document, which is a type of Work Authorization Document (WAD), consists of sequences of steps of tasks performed on a Space Shuttle used to satisfy a set of OMRS's. Performing steps in an OMI is a prerequisite for an OMRS to be verified and closed.

Project Engineering monitors verification-closure status, i.e. open or accomplished, of OMRSs with software known as Operations and Maintenance Requirement Specification Flow Planning (OMRSFP). The manual verification-closure of an OMRS in OMRSFP is known as buy-off.

Ground-based checkout-systems are used to checkout the Space Shuttle at KSC. The Checkout and Launch Control System (CLCS) is an example of a new checkout-system replacing the Checkout and Control Monitor Subsystem (CCMS). Checkout-systems consist partly of software to perform checkout of the Space Shuttle. Checkout console operators utilize checkout-systems to perform automated and semi-automated tests on Space Shuttle systems.

During a test, checkout-systems software detects major OMRS events for Space Shuttle flight-hardware. Major events include:

1. Pass/verify event of an OMRS
2. Events that would require the Space Shuttle processing personnel to recheck flight-hardware, e.g. removing flight-hardware and reinstalling it.
3. Fail/invalidate event of an OMRS.

There are about 8000 OMRSs per flow, i.e., shuttle processing to support a mission. The required OMRSs need to pass before advancing the Shuttle from the current processing location to the next, i.e., from the Orbiter Processing Facility (OPF) to the Vehicle Assembly Building (VAB), from the VAB to the pad, and from the pad to launch.

Operational Scenarios (Legacy)

This Operational Scenarios (Legacy) section describes how the legacy applications are used for Space Shuttle processing.

Operational Scenario 1a: Checkout Operation (Legacy)

A system engineer arrives at the firing room console for testing. The system engineer reports ready to perform a checkout to the Test Conductor on station. The system engineer references the OMI required. The system engineer activates the CCMS software required to perform the checkout. The system engineer commands the CCMS console for the checkout. CCMS interrogates Space Shuttle systems. Space Shuttle systems return return-data to the CCMS console. CCMS indicates as to whether Space Shuttle systems' return-data meets OMRS specified-data requirements. The system

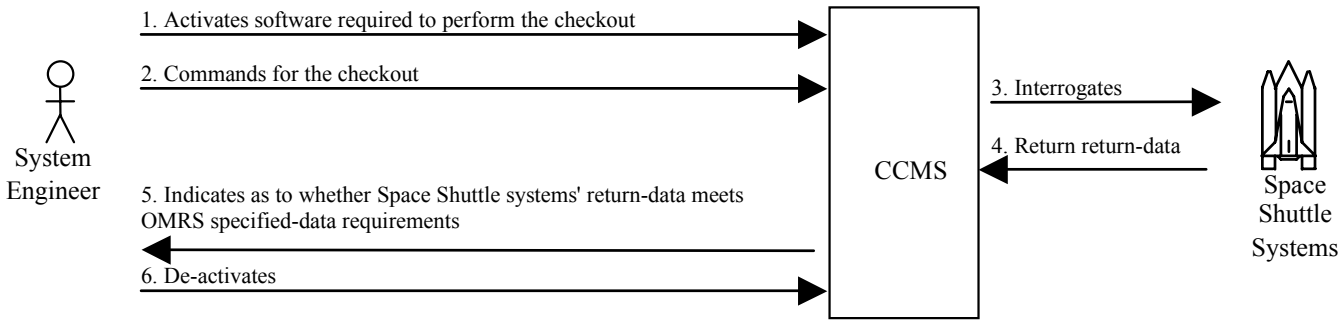

Figure 1 Use Case: Checkout Operation (Legacy)

engineer records the checkout conclusion in the OMI. The system engineer de-activates the CCMS software required to perform the checkout.

Operational Scenario 1b: SPDMS-OMRSFP OMRS Verification Entry (Legacy)

The system engineer arrives at a Shuttle Processing Data Management System (SPDMS) SPDMS-OMRSFP. The system engineer activates the SPDMS-OMRSFP software required to perform OMRS verification entry. The system engineer commands the SPDMS-OMRSFP software to update the OMRS verification status based on the checkout conclusion. SPDMS-OMRSFP indicates the successful update of the OMRS's verification status. The system engineer de-activates the SPDMS-OMRSFP software required to perform OMRS verification entry.

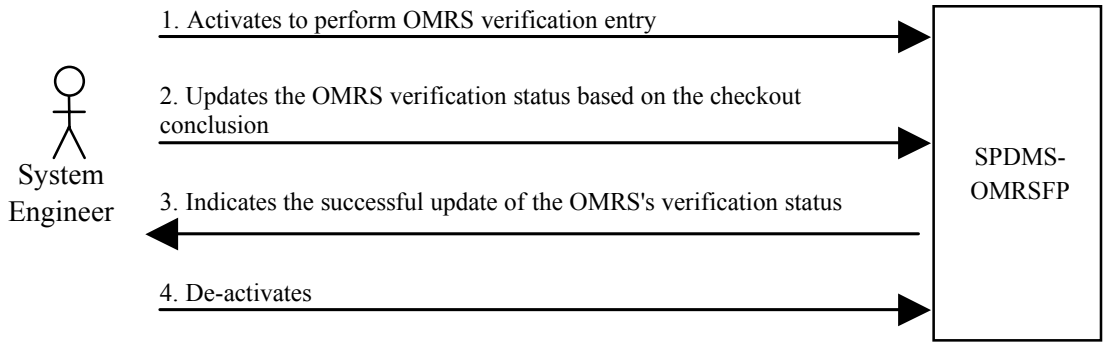

Figure 2 Use Case: SPDMS-OMRSFP OMRS Verification Entry (Legacy)

Solution Domain

The Operations and Maintenance Requirement Specification - Automated Buy Off System (OMRS-ABOS) is the application that provides an integrated software capability to automatically buy-off those OMRS's that can be verified by checkout-systems software. OMRS-ABOS provides communication between OMRS producers, such as CLCS, and OMRS consumers, such as Project Engineering.

Operational Scenarios (OMRS-ABOS)

This Operational Scenarios (OMRS-ABOS) section describes how the solution applications will be used for Space Shuttle processing.

Operational Scenario 2: Checkout Operation (OMRS-ABOS)

A system engineer arrives at the CLCS console for testing. The system engineer reports ready to perform a checkout to the Test Conductor on station. The system engineer references the OMI required. The system engineer activates the CLCS software required to perform the checkout. The system engineer commands the CLCS console for the checkout. CLCS interrogates Space Shuttle systems. Space Shuttle systems return data to the CLCS console. CLCS application software evaluates as to whether Space Shuttle systems' data meets OMRS specified-data requirements. If an OMRS event occurred, the CLCS application persists the OMRS-event-data to the SDC. The CLCS console indicates the checkout's OMRS event. The system engineer records the checkout evaluation-result in the OMI. The system engineer de-activates the CLCS software required to perform the checkout.

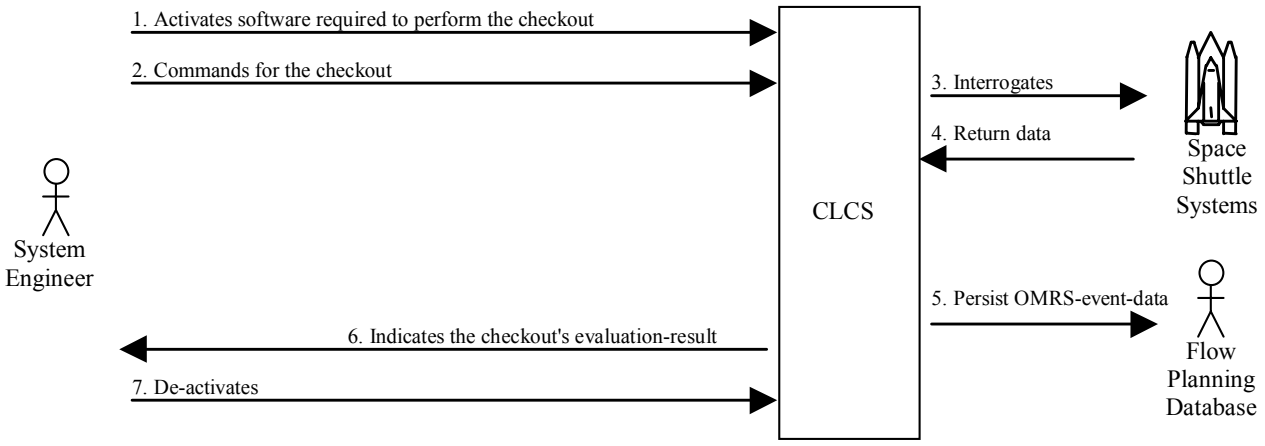

Figure 3 Use Case: Checkout Operation (OMRS-ABOS)

System Architecture

This section provides a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components.

Upon realization of an OMRS event, CLCS application software simply calls a pass or fail operation on the appropriate OMRS model, e.g., V41AN0.010-A. This model then writes the OMRS-event-data to a Function Designator (FD). The FD's new value is transmitted to the Shuttle Data Center (SDC) via the CLCS Service Network.

At periodic time intervals, the OMRS-ABOS Application Server calls the SDC Application Programming Interface (API) to retrieve OMRS-event-data that occurred since the last time the OMRS-ABOS Application Server called the SDC API. The OMRS-ABOS Application Server receives the CLCS-generated FD data, extracts OMRS-event-data from FD data, and then stores the OMRS-event-data in the OMRSFP-database-staging-table.

OMRSFP retrieves the stored OMRS-event-data periodically from the staging table and writes the data to the same database table populated during manual entry, i.e., the legacy use case (figure 2).

When the application starts and restarts, the application calls the SDC API to retrieve OMRS-event-data that occurred during the time that the application was unavailable. CLCS-generated FD's are then received by the application.

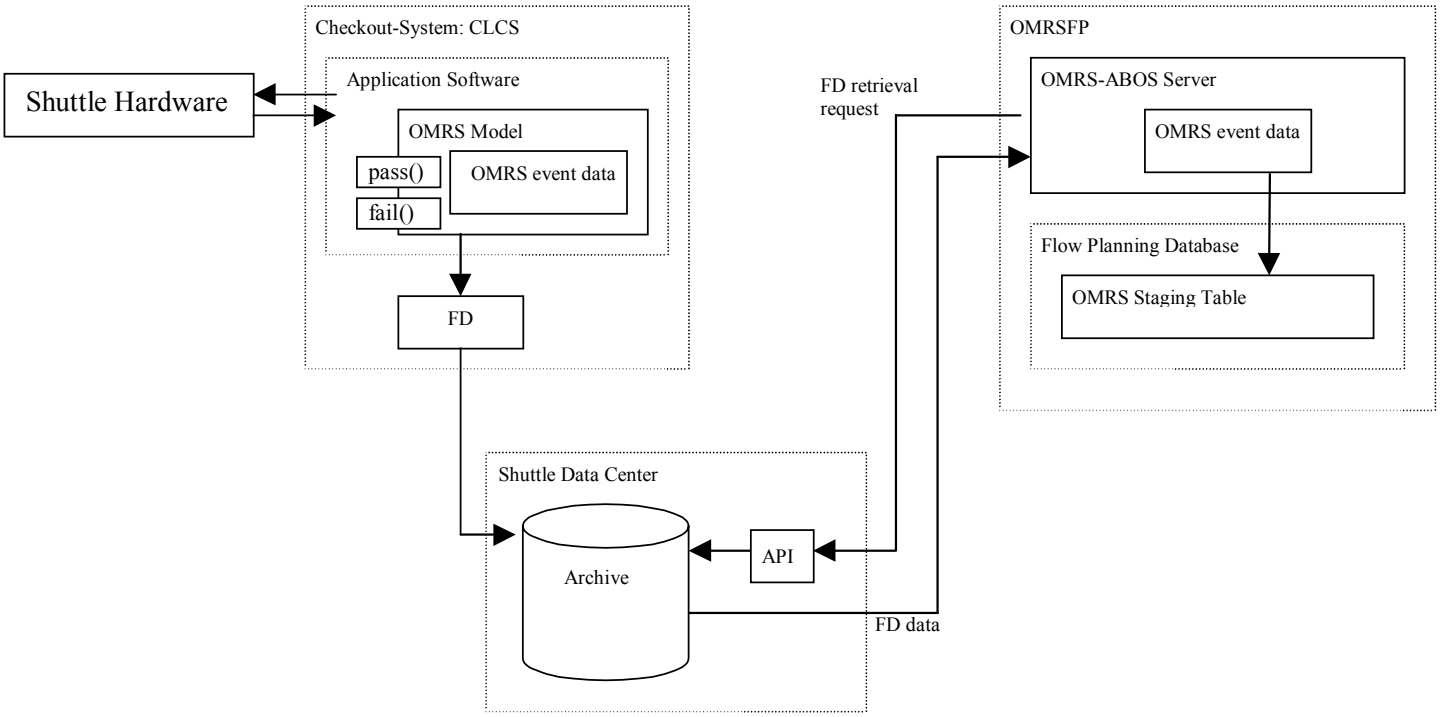

Figure 4 Platform Model: OMRS-ABOS

Platform Architecture

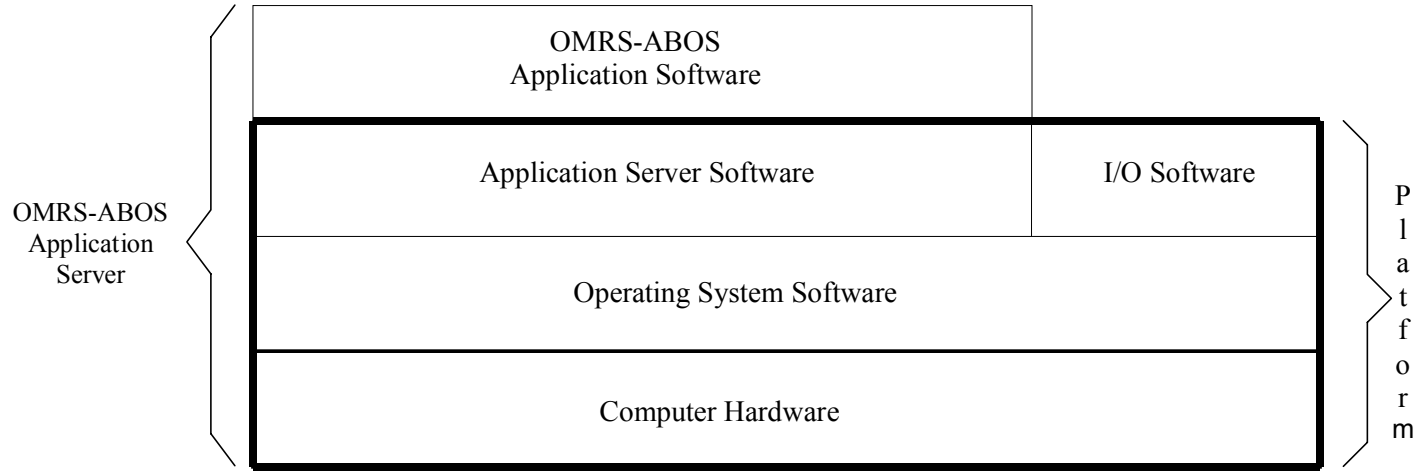

Figure 5 Conceptual Model: OMRS-ABOS

The computer hardware consists of one Compaq ProLiant DL380 Model DL380R01 P667-256K with 512MB PC133MHz Registered ECC SDRAM DIMM Memory, four 18.2 GB Wide Pluggable Ultra2 Drives, one Compaq NC3134 Fast Ethernet NIC 64 PCI Dual Base 10/100 network card, and one

Compaq Hot Plug redundant Power Supply Module. The computer hardware is designed for use in conjunction with DataCenter Operations (DCO). The operating system software consists of the Microsoft Windows 2000 Server operating system. The application server software consists of BEA Systems WebLogic Server 6.1 (non-clustered).

Software Development Process

The software development model offered an efficient process to enable rapid development of requirements. The software development process utilized a quality-designed-in philosophy. Each software component was associated with an appropriate level of test. The process captured user requirements, infused requirements into the design, provided quality control, and developed software as object oriented components.

The software development process made extensive use of industry best practices to ensure consistent high quality production. To further enhance efficiency, rapid prototyping was used.

The application software development process considered current system functionality, proposed operational concepts and developed user requirements. Software was reviewed against the total system architecture. The review process considered the total software lifecycle including sustaining.

A type of software development methodology used for the application development was evolutionary prototyping. In evolutionary prototyping, the goal is to achieve functionality for demonstrating a node of the application to the users for feedback and application evolution. The prototype emerges as the application later in the lifecycle. As with each spiral, functionality is enhanced and implemented. This method benefits the software development process in the following ways: potential problems are identified early in the product lifecycle and the application is executable early in the product lifecycle, with the breadth of the application increasing as evolutionary prototype sub nodes are replaced by released software

A major problem with introducing new technology into a legacy environment is the large investment that exists in systems already in sustaining. Completely reengineering a legacy system with new technology may not be realistic. A threshold does exist, however, where the expected life of a system justifies that it would be better for it to be sustained after being reengineered with new technology. Evolutionary prototyping can be utilized on critical nodes of a legacy system. This approach can be used as a means to inexpensively transition to broader reengineering efforts by incrementally building an evolving framework from which to build. This reduces potential risk and aids in expense and performance control.

Architectural Strategies

Java was chosen as the programming language due to Java's object oriented nature, cross-operating-system capability, and reusability aspects of Java components. Java 2 Enterprise Edition (J2EE) was chosen as the host platform for services required of the Java components developed. J2EE offered timesavings, cost-savings, and access to expert knowledge. J2EE provided extensibility above the Java API Library to include high-level services that facilitate Rapid Application Development (RAD). This was accomplished by an aggregation, the pooling of common problem domain constructs, from across the enterprise, as packages of services. J2EE provided transactional processing services ensuring Atomic, Consistent, Isolated, and Durable (ACID) transactions between application data stores.

The choice of database is primarily constrained by OMRSFP in that the database is implemented in Oracle. The Oracle database product line is very popular and robust. The standard language to access Oracle databases is Structured Query Language (SQL), which is just that, a standard with most database

vendors. Java, J2EE, with SQL allow for migration paths from differing data structures from internal memory data constructs to relational database record sets.

Component architectures facilitate a divide and conquer solution in that the software development team extracts the problem domain constructs as separate software components. J2EE provided a component architecture known as Enterprise JavaBean (EJB). An EJB is a server-side component that provides an interface between the caller and the J2EE server hosting the EJB. EJB's facilitate component reusability and mobility. Components developed for the application will be EJB's. Component-based programming facilitates good cohesion with minimal coupling between components. Maintenance and extensibility of the application are therefore enhanced.

Due to the network-centric aspects of J2EE, distributed data was handled quite easily via the J2EE Java Naming and Directory Interface (JNDI) services. Though the data flow for the application does not include distributed data, as defined by commercial industries, J2EE provided a solution if that were ever a necessity.

The Future

The benefits provided by OMRS-ABOS supports KSC Roadmap Objective 1.1A – Provide Safe, Reliable, Cost Effective Processing of Shuttle and Expendable Launch Vehicle (ELV) Launches. Benefits include reduced clerical labor expenses by eliminating the time required for manual OMRS status entry and increased reliability by eliminating the possibility of human error during OMRS status entry.

The buy off pattern between an information producer and information consumer is not limited to OMRS's. This project is the beginning of addressing the OMRS market at KSC/NASA. Businesses in that market include CLCS and OMRSFP within the project's scope. An additional business in the OMRS market includes the Space Shuttle Remote Manipulator System (RMS).

OMRS-ABOS created a channel for a market. There are more markets at KSC/NASA than the OMRS market. OMRS-ABOS was designed in such an abstract manner such that the architecture is extendible to address additional markets. Accommodations for distributed data communications have been implemented via the encapsulation that eXtensible Markup Language (XML) provides. By utilizing XML as a data transport vehicle, OMRS-ABOS is able to communicate more easily with commercial enterprise platforms.