RESINS Checkout System Software: Mapping Requirements to Design Features

Jaya G. Nair, C. Radhakrishna Pillai, S. Hemachandran, Dr. P. P. Mohan Lal
ISRO Inertial Systems Unit, Indian Space Research Organisation,
Vattiyoorkavu, Thiruvananthapuram
g_jaya@vssc.gov.in

Abstract:

Computerized Checkout Systems are used for the calibration and performance evaluation of navigation systems. The checkout software needs to realize multiple tasks with different requirements of periodicity and deadlines, to support fool-proof operations with need for minimum intervention by the operators and be available for different missions concurrently without change in the operating procedures. Obtaining a mapping from requirements to specific features in the design enables the effective development and maintenance of the software. This paper discusses such a mapping and describes the mechanisms used for realizing the different design guidelines.

Key words: Checkout Systems, Automation, Safety, Fault-tolerance.

I. INTRODUCTION

Navigation Systems used in launch vehicles need to be calibrated and subjected to many tests in different environmental conditions to qualify them for flight. PC-based checkout systems are used for this purpose. These checkout systems consist of the hardware interfaces necessary to connect the test articles and a checkout program containing the software modules to conduct the various tests. The functional requirements to the checkout software are derived from the test and evaluation plan of the navigation systems. In addition to these, the checkout software is characterized by special challenges related to the environment of use, the complexity of multiple inter-related requirements and stringent demands of quality. These demands can be achieved efficiently and effectively only by addressing them during the design of the software itself. The requirements must be mapped to specific features in the design of the checkout software. This paper discusses how this mapping is achieved in the design of the software for the Checkout System (COS) used for testing the Redundant Strap down Inertial Navigation System (RESINS) for GSLV.

The different types of requirements to the checkout software are briefly enumerated in section II. This is followed by an analysis in section III, to arrive at the design features needed to realize the requirements. The details of implementation of the design features are discussed in section IV followed by implementation details and conclusions in section V and VI respectively.

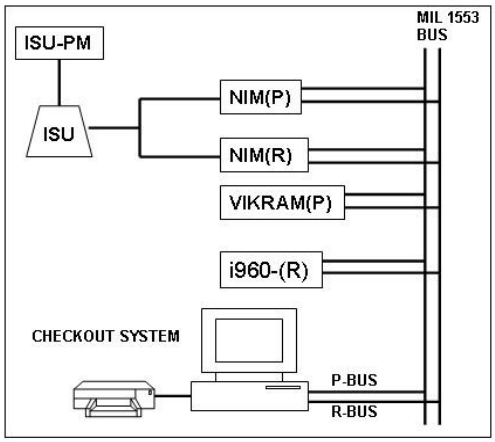
II. REQUIREMENTS TO CHECKOUT SOFTWARE

The requirements fall under separate categories as follows:

1. Functional Requirements

The test article to which the checkout system has to interact is illustrated by figure [1]. It shows Mark-4 ISU connected to Navigation Interface Modules (NIM). The two NIM and two Navigation and Guidance Processor (NGCP) packages which form the onboard processor units, are connected to MIL-1553 buses with redundant connections. The checkout system is also connected to the two buses. Interfaces necessary for powering the packages, for monitoring, data acquisition and providing simulated inputs to the packages are also needed, though not shown in fig [1].

The checkout software must have provisions for powering the packages, implement the monitor-mode (MM) and flight mode (FM) interfaces to the



Fig[1]: Test Articles and Checkout System

processors, acquire and store telemetry data from the different processors in the two buses, and implement the test procedures corresponding to calibration and performance evaluation tests of RESINS. The tests include simulation tests where the checkout system has to simulate the inputs to NIM, tests where checkout system has to simulate the behavior of NGCP on the two buses and power-supply variation tests. Provisions for on-line and off-line data processing, initialization data update, generation of test results and data archival are also functional requirements of the checkout software.

Fig[2] shows the context under which the checkout software operates.

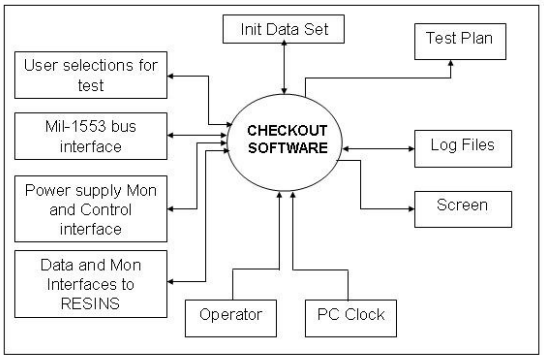


Fig [2]: Context diagram for Checkout Software

2. Special Requirements

Certain challenges to the RESINS Checkout system that translate to special requirements to the checkout software are enumerated here.

The navigation systems for different missions have broadly the same test requirements but there are differences in many specifics. These include changes in the format of data, software used in navigation processors, test conditions, methodology used for computation of test results etc. The RESINS checkout software must be available to support the different missions concurrently.

Different units of checkout systems need to be deployed to cater to tests at different test stations including environmental test-beds. Differences in the configuration of the checkout hardware including allocation of I/O channels between the different units need to be supported seamlessly.

It must be possible to operate the checkout software for long hours without failure as some of the calibration tests involve a number of steps which must be completed in the same day.

The checkout systems are operated by people who are not necessarily RESINS experts. Hence the system should be equipped to support fool-proof operations with minimum intervention required from the test personnel.

To reduce the time of test, it must be possible to efficiently use the checkout software and provide all required facilities for decision making in the checkout system itself. The software must provide necessary alert messages to the operators as and when required.

The software should be able to accept different configurations of the test articles, simulating the behavior of the missing packages accordingly.

The checkout systems are typically used for many years which translate to having a long maintenance phase after the initial design and deployment. Any failure and reallocation of hardware interfaces during the maintenance phase must be absorbed and it must be possible to support updates to the software and setup as and when needed.

III. DESIGN FEATURES NEEDED TO MAP THE REQUIREMENTS

The broad requirements enumerated in section II are analyzed in detail to arrive at a set of

design guidelines to the checkout software. Some guidelines are laid down in order to realize the functional requirements of the software in the most efficient way. The other guidelines are aimed at achieving the special requirements.

The checkout software should be **correct** and **complete** with respect to meeting the complex functional requirements under all conditions of load. The maximum load on the system with respect to timeliness and amount of data to be handled occurs at the time of tests involving simulation of inputs to NIMs, updated every 20 msec. The strict timing requirements must be met at all times.

To concurrently support RESINS packages of different missions, the checkout software must be able to reconfigure itself to the differences in the test requirements between missions. The feature of **reconfiguration** must also solve the need to work in different checkout units with potential differences in hardware interfaces. The software must also reconfigure itself in the face of reallocation of hardware interfaces during the long maintenance phase.

To cater to the years of usage of the checkout system, the software must have the feature of easy **maintainability**. The software must be scalable and it must be easy to understand and make changes.

The feature of **self-containment** is needed to ensure efficiency of usage and also to act as the decision-making system during the performance evaluation of RESINS. The software must provide OK/NOT OK messages about the status of checkout system itself and SUCCESS/FAILURE messages about the status of performance tests conducted on the test articles.

The feature of **automation** is needed to support fool-proof operations in the checkout. The test procedures must be automated to the extent that minimum intervention is required from the operator. Automation also means that software should have features for **security**, **safety** and **fault tolerance** without the need for conscious action from the user. The software must also have the feature of **user-friendliness.** The system should be easy to operate and should provide a consistent interface for the user, across the many hues of its use.

The need to operate continuously for long hours without failure necessitates that the software must have features of **robustness**, **availability** and **reliability**.

IV. DESIGN OF MECHANISMS TO REALISE THE FEATURES

The mechanisms employed to realize the different features are explained here.

1. Correctness and Completeness

The checkout software needs to implement many actions with varying degrees of criticality and periodicity. Complex tests need simulation of inputs to the test articles and acquisition of data from multiple interfaces with millisecond periodicities. To achieve correctness in the face of hard real-time deadlines, the checkout software is designed as a real-time system using a hierarchy of co-operating processes with different priorities and scheduling policies.

Choice of operating system and the architecture of the software are critical in achieving correctness.

1.1. Choice of Operating System

To facilitate the design as a real-time system, it is decided to use a real-time operating system (RTOS) and QNX is selected. QNX has fast context-switch and interrupt latency times and many mechanisms for inter-process communication and synchronization.

1.2. Software Architecture

The software is designed to have client-server architecture. The servers are independent, each implementing a particular functionality and expose themselves only through the interface. The main checkout program acts a client to the many servers. The server programs include device drivers for I/O cards, programs for handling processing and storage of high-speed data, implementing the protocol for communicating with the test articles, for printing functions etc. Servers run in the background and the users interact with the clients. Fig[3] shows the list of servers and clients that typically operate during a session of checkout use.

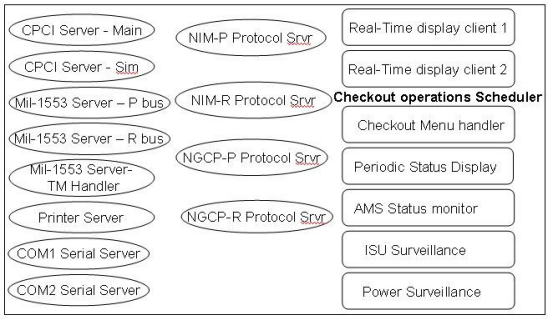


Fig [3]: Servers and Clients

2. *Reconfiguration*

All factors which may undergo change are captured as parameters to the checkout software. The values of these parameters are specified using a set of configuration files. The files capture all the different scenarios of usage with respect to missions, checkout units and test beds. Users for the different projects are distinguished by defining them as different users in the multi-user system. The test-configuration for a particular invocation of the software gets specified by a combination of the identity of the user, environmental settings, command-line arguments and selections made by the user from a list of options shown. Once the test configuration is specified, the software uses the set of applicable configuration files to define the values of all parameters and is able to reconfigure itself. It is possible to completely change the look and feel of the checkout software in this manner.

Reconfiguration applies to fields shown in the Graphical User Interface (GUI) displayed, the menu of options shown, channels used in data acquisition, data processing methodology used, reference values used to check test results etc.

The feature of reconfiguration makes the software adaptable and extensible.

3. *Maintainability*

The capability of reconfiguration across changes is the main tool that helps to achieve maintainability.

Device drivers are designed to implement common APIs independent of the features of any particular hardware. This makes it possible to support failures and updates to hardware interfaces during the long maintenance phase. It is possible for the system to be implemented in cPCI systems or industrial PCs, and is scalable across hardware enhancements. Independence of servers also adds to scalability of the system.

The main client program controls the sequence of operations. The single point of control improves maintainability and understandability of the whole system. The servers are typically implemented in object-oriented programming languages, paving the way for code reuse.

4. *Self-containment*

All the necessary tools for completing the tests are made available in the checkout system itself. Periodic display of the status of the test articles and tests in progress is provided. Tools are used for data processing and analysis with automated checking of results. These mechanisms enable decision-making during tests without the need of any additional equipment.

5. *Automation*

Different mechanisms are designed to implement automation. Automating the steps of test procedures is the basic mechanism. The parameters for the tests are captured in the software so that minimum intervention is needed from the test operator. Tests involving a number of steps are automated using configuration files that capture the parameters of intermediate steps.

Some tests need the test-jig to be commanded for necessary orientation and rotation. Commands to test-jig controller are generated automatically from the checkout system by building an interface and command protocol between the two systems.

Automatic verification of initial conditions necessary for tests, verification of correctness of the operation of intermediate steps, real-time display and checking of critical parameters during tests, data processing and verification of results at the end of tests are some of the other mechanisms used to enable automation. Availability of unambiguous and time-tagged log data which helps in problem analysis is also necessary.

Automation creates an identical condition for the system all along the testing process. This is highly essential to ensure correctness of test results since navigation systems are highly sensitive to changes in the environment of tests.

6. Robustness

The program is defined to have different modes of operation. The *test-mode* is the one used by the user. *Diagnostic-mode*, which supports fault diagnosis in the checkout system aids in doing checks on the fly. The checkout program is also designed to have an option *hot-exit*, which helps in exiting from the program preserving the context, and *hot-start*, which inherits the previous context. These features help in maintaining continuous operations even in the face of unforeseen circumstances or failures.

7. Safety

The checkout software controls the sequence of applying power to the test articles and checks power-supply voltages before powering packages. Power-supply voltages, the currents drawn by the packages and critical parameters related to the health of packages are continuous monitored. In case of any anomaly, automatic actions are taken to ensure safety and operator is alerted by audio signals and display. A watch-dog circuit is designed which alerts with loud alarm if the checkout software is not working as intended. Any user-input provided during test is verified before use.

Some of the automatic safe-actions are disabled during critical tests where a number of initial conditions are necessary to be satisfied. Clear indication of any error by display and alarm is the method used in such situations. There is also provision for an emergency switch-off to cut off power to test articles.

Precautions for ensuring safety of test-log and data are also of importance. Automatic re-routing of printer data to a file in case of any error in the printer is an example. The operator is alerted, and once the printer becomes ready, the stored data is printed and then normal printing mode is restored. No user action is required to activate this re-routing.

8. Security

Security is ensured by defining a set of users, categorized by missions, for the checkout system and clearly demarcating the pathname space accessible to each user. The options for hot-start and hot-exit from the program are controlled by passwords. User authentication by username and password, allotment of privileges to users only as required and use of binary data format for configuration files are some of the other mechanisms designed for security.

9. Fault-tolerance

Fail-safe is the condition of fault tolerance required in checkout systems. Automatic safe-actions are undertaken with continuous health surveillance of the test articles throughout the time of tests. In case of anomalies, tests are aborted and the setup brought back to safe conditions.

The client-server architecture of the software enhances the capacity for fault-tolerance and re-configuration as the clients and servers can be updated seamlessly as long as the interfaces remain constant. The network-aware nature of QNX makes it possible to deploy servers in any computer in the network if needed, without changing the software or the operating procedure.

10. Usability

The software has a Graphical User Interface which is menu-driven and intuitive to use. The operating procedures are kept the same across the different missions and test-beds. Automation of test procedures enhances usability. A display window shows the status of tests, the interfaces and health of the packages continuously. Any abnormal condition is highlighted in the display, messages are printed and an audible alarm is raised. The user is provided with the flexibility to group and configure the parameters to be used to calculate results as the test condition demands.

11. Responsiveness

Use of intelligent peripheral interfaces helps to shift time-critical tasks away from the main system. Use of asynchronous inter-process communication mechanisms and real-time features of QNX including fast context-switches and preemptive priority scheduling help in realizing the response times required.

V. IMPLEMENTATION

The checkout software is realized with version numbers and new versions are released to incorporate updates and changes during the long

period of use of the checkout system. The software is subjected to different levels of a formal verification and validation process to ensure correctness and completeness. RESINS checkout software has successfully been in operations for more than a decade supporting various missions.

VI. CONCLUSIONS

Need for concurrent support of missions with requirements which are sometimes at odds with one-another, long periods of usage and fool-proof operations is the hallmark of checkout software. The mapping of requirements to features in the design enables the development of software that not only implements the functional requirements, but is also easy to maintain and use. The different features and methods of implementation employed in RESINS checkout software have been highlighted.

Possible future extensions to the software include building better awareness in terms of the test conditions, automatic generation of a data base of meta-data information, use of semantics in the configuration files and adaptation with respect to changes in test procedures. Building provisions for voice commands, wireless networking and a web-interface to monitor various test beds at a single portal are some of the other extensions envisaged for the RESINS Checkout software.

ACKNOWLEDGEMENTS

Mr. P. S. Veeraraghavan, Director, IISU, is gratefully acknowledged for encouragement and support of the work detailed here. The authors would also like to acknowledge Mr. B.C. Vidwani, DD, LVIS for encouraging us in this work.