Locomotive Control Simulation and Validation System

by
Gregory C. Cala, Ph.D.
Vice President, Operations
Data Science Automation
USA

And

Benjamin A. Rayner
Consultant, Measurement & Automation
Data Science Automation
USA

And

Marcela Maldonado
Consultant, Measurement & Automation
Data Science Automation, Inc.
USA

Category
Simulation

Products Used
LabVIEW 6.1 for Windows
LabVIEW 4.1 for Real-Time Unix
TestStand 2.01
Reflective Memory with Fiber Optic Interface
Custom VME distributed control modules

The Challenge

An international manufacturer of locomotive transportation and control systems must validate the locomotives proper functionality if any control system software modifications are made. An automated, flexible, and productive test architecture is required to replace the current manual procedure that is tedious and requires time consuming, error-prone human intervention.

The Solution

Data Science Automation used NI LabVIEW and TestStand to develop a test architecture that replaced the locomotive status console (a serial device) with an automated virtual console developed with LabVIEW. The entire effort required integration of the locomotive simulator which included a legacy LabVIEW 4 application running under a real-time UNIX operating system.

Background

Locomotive control systems are extremely complex and involve the tight integration of numerous off-the-shelf and custom components in a distributed architecture. These components control the various subsystems of the locomotive through software running on a network of controllers in master/slave architecture. Inputs to the control system come from a variety of

375 Valley Brook Road • Suite 106 • McMurray, PA 15317 • T 724-942-6330 • F 724-942-8390 • DSAutomation.com

devices (sensors, buttons, levers, pedals, etc.). The status of many individual system parameters are indicated with a variety of devices (lights, gauges, digital displays, graphical displays, etc.).

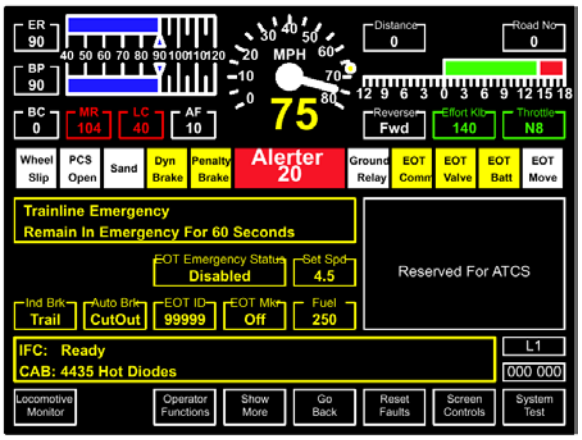
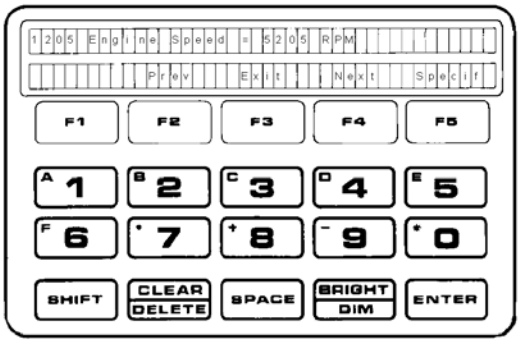


Figure 1. One of the onboard locomotive graphical displays.

The status of all system parameters are available (one at a time) through a custom Diagnostic Information Display (DID) that uses an RS-232 serial protocol to the main locomotive controller. The DID consists of two forty-character display lines and a keypad to allow data entry of the desired parameter ID number.

Figure 2. Diagnostic Information Display (DID)



Simulation and Manual Testing

A simulator was developed previously. It replicates the dynamic response of parameters to changes of others with actual locomotive hardware in the loop (HITL). The simulator runs under Real Time Unix and a LabVIEW 4 application provides a graphical interface to change control parameters and view the current status & response resulting from parameter changes. After a period of time following a set of parameter changes on the simulator, steady state conditions are reached. At that point, the DID is used (as it would be on the locomotive itself) to view parameters individually. A test would involve setting many different operating conditions, using the DID to scroll through thousands of parameters, transcribing the parameter value to a test plan log sheet and comparing the actual to the expected value to determine a pass or fail for each parameter at each operating condition. Each test suite would take many days and was error-prone due to the transcription of values.

Automated Testing

To automate the process, a virtual DID (vDID) application was written in LabVIEW 6.1 and run on the Windows 2000 operating system. This emulates the communications over RS-232 to the main controller of the simulator. A pair of LabVIEW test modules were written to get parameters through programmatic control of the vDID using control references and to set operating conditions by writing to "reflective memory." Reflected memory is networked memory shared by the various simulator components.

Figure 3. The vDID Front Panel showing the result of a request for Parameter ID 1205.

375 Valley Brook Road • Suite 106 • McMurray, PA 15317 • T 724-942-6330 • F 724-942-8390 • DSAutomation.com

Next TestStand was used to configure the sequence of operations required to repeatedly

- Set operating conditions,
- Wait for steady state,
- Request parameter values,
- Compare actual to expected values,
- Determine pass/fail condition,
- Archive results,
- Report results.

There were several classes of parameter values that needed to be handled, including floating point numeric (e.g. temperatures), integer numeric (e.g. Speeds), strings (e.g. On, Open), and bit patterns (e.g. 010010). Each of these required special parsing and interpretation in order to report the result to TestStand in a manner appropriate for that test type. The "Get Monitor Parameter" module was developed to be universally applicable to all classes. The Input Buffer string received from TestStand was used to identify the parameter ID and its class.

Several challenges were overcome during the development of the "Get Monitor Parameter". These primarily stemmed from the absence of detailed documentation for the communications protocol between the DID and the main controller. Inconsistent command formats were particularly troublesome. In addition, when the DID requested the status of a parameter, the controller responded within a fraction of a second with the cached value from the previous request, and two seconds later with the current value of the requested parameter. These hardware limitations significantly reduced the test productivity below what was otherwise achievable with a fully automated simulation program, but they could not be overcome.

Conclusion

Even with the hardware limitations that increased the test time for a single parameter from approximately 0.1 sec to 1.5 sec, test time was significantly decreased from the approximately 30 sec previously required during manual testing. Fortunately, it was still a significant speed performance enhancement, and the ability to run unattended with more reliable results made this automated solution invaluable.

Previously, an entire test suite would take at least 80 hours to complete, and would be necessary for each of 100 code modifications per year, for a total of 8000 man-hours of effort. This was reduced to 4 hours per test suite, or 400 man-hours of effort. As a result, software change requests no longer have to be accumulated over many months in order to justify allocation of the time required for validation of significant or benign changes. An entire suite of tests can now be run unattended. The possibility of transcription errors has been eliminated. Test results are available immediately and can be electronically shared with team members and are archived for future reference.

375 Valley Brook Road • Suite 106 • McMurray, PA 15317 • T 724-942-6330 • F 724-942-8390 • DSAutomation.com