# TELEMETRY SYSTEM AS A NETWORK TEST APPLIANCE: A SYSTEMS, TEST AND SOFTWARE COLLABORATION

**James P. Knuff, Eric S. Greene**
**Raytheon Missile Systems**

## ABSTRACT

An automated missile testing environment, reliant on telemetry data, demands automated control of telemetry devices. Software reuse across many missile products (Wikipedia)[i] and different lab environments requires a software control product that has a simple interface and an ease of modification across different telemetry device vendors. This paper describes a software application that integrates telemetry control/status into automated test and provides a simplified GUI to expedite manual testing. Results from this application show telemetry overhead time reduced by 74%, with a rapid payback on our investment of less than six months.

**KEY WORDS:** Automated Test, Telemetry, Test Executive

## INTRODUCTION

As a missile moves through its development cycle and its subsystems are integrated together, accessibility to performance data becomes more limited. When a fully integrated missile round is tested, data access may be restricted to just telemetered data. Even during earlier phases of testing, telemetry data may be used to evaluate software changes.

A problem arises in creating a telemetry services application that is usable in an automated test environment and is suitable for simple manual testing. Creating such a capability reduces hardware button actions and significantly quickens overall test cycle time. Another benefit is that a knowledgeable telemetry operator is no longer required during testing.

A typical telemetry system used in our Hardware In the Loop test labs (Figure 1) contains a telemetry station/card, RF receiver, digital data recorder and one or more PCs to support real-time control, telemetry data extraction and data visualization. All system devices are connected to a telemetry PC via a common Ethernet network.

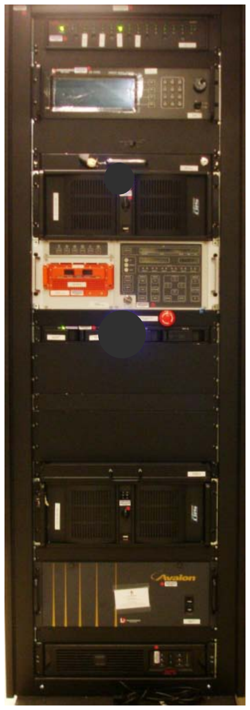


**Figure 1: Telemetry System**

We created this client/server Telemetry Services Executive application (TMSE) to reside on a networked telemetry system PC. Users communicate with the TMSE via a simple set of commands issued across a network TCP/IP connection. Each command invokes actions that are statused back to the user.

The controlling Test Executive software (client) resides anywhere on the network. It communicates with the TMSE using telemetry service messages inserted into its test script and provides additional software code to evaluate returned status. Our GUI also resides anywhere on the network and uses the same command/status messages.

## TELEMETRY AND MISSILE DEVELOPMENT

Since well thought out telemetry content provides a quick performance assessment, it is useful even in early product development. It might be surprising to some that missile telemetry plays a role in many different test environments.

During product development and testing, several different types of test labs are utilized (Table 1), each with diverse telemetry system capabilities. These include Computer In the Loop (CIL), Hardware In the Loop (HIL) and our Round Level missile test labs.

Table 1: Missile Test Environments

| Test Lab | Typical Types of Testing | Telemetry Equipped | Telemetry Utilization |
|---|---|---|---|
| Software Evaluation Station | Build up and integration of flight software | Yes | Low |
| Computer In the Loop | Closed loop subsystem testing , Hardware Integration, Software Qualification | Yes | Low |
| Hardware In the Loop | Flight hardware, Simulated environments, motion, flight test support | Yes | Moderate |
| Round Level | Flight hardware, Simulated environments, actual testing | Yes | High |

As an example, our CIL lab involves early prototype hardware or hardware simulants, excited primarily through computer driven scenarios. As the hardware design matures and moves up to our HIL lab, CIL labs are still used to quickly and efficiently test software modifications. Simple changes to test scenarios and initial test conditions allow us to Monte Carlo software changes under a wide array of operational conditions.

## NEEDS

Significant problems exist in marrying telemetry to product testing. Fast, automated test cannot coexist with time-consuming manual button pushing without slowing down the whole test cycle. Also for many engineers, telemetry setup and operation is an acquired skill requiring a considerable amount of on-the-job-training. This skill barrier is a major hurdle.

Furthermore, directly integrating automated telemetry into our labs has proven difficult since our missile programs use different automated Test Executive applications (commercial and homegrown). Requiring direct code integration, across this variety of applications, is an onerous proposition.

Yet the need for automated telemetry control is still important. Therefore, through careful conceptual and system design we targeted solutions to these and other major problems.

## TMSE SOFTWARE DESIGN ELEMENTS

We facilitated our solution with these TMSE software design elements[ii] (Hayes), (Pettichord)

- Use of a simple network message command/status set
- Independence from the test context, test environment and test hardware
- Test scripting/setup is independent of test type
- Modularity and Portability
  - No re-hosting the TMSE application onto another network PC
  - No direct integration of TMSE into the Test Executive code
- Test cycle time reduction
  - Automated test control over a full feature set
  - An aggregated result summary file for quicker post-test evaluations
  - Telemetry files pushed to specific PC/folder for quicker post-test evaluation
- Simplified GUI eliminates a tester's need for specialized telemetry skills
- Reusability by design (National Instruments)
- Maintainability via simplified internal data structures and command/status syntax

We also focused on reusability. By using a standard network command and status set, the TMSE is able to function irrespective of the telemetry hardware services utilized. The TMSE uses a Front-end, Back-end software design (Figure 2) that isolates the client/server command and status communication from the specifics of the particular telemetry hardware devices.

In this approach the explicit commands conveyed to the Back-end do not change with either device type or vendor. Likewise, the explicit status syntax returned back to the Front-end does not change. The Back-end contains all specific vendor commands required to control/status each device across the network. Both sides are bridged via data structures and housekeeping functions that run inside the application.

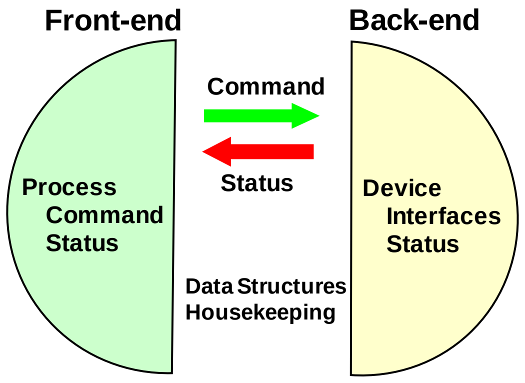


Figure 2: Software Design

When a new telemetry device vendor is selected, no Front-end changes are required, just a new device specific Back-end component. This component would contain an internal command/status handler plus its device specific network commands.

## SERVICES

Our design provides a feature rich application that supports a wide range of telemetry systems and telemetry data users (Table 2). Some of these services are described here.

Table 2: Provided Services

| Service | Description |
|---|---|
| Coordination | Common Configuration file, Command & Status messages |
| Telemetry | Setup, Start/Stop, File naming, Status |
| Data Exchange | Setup, Start/Stop, Status |
| RF Receiver | Setup, Status |
| Data Recorder | Setup, Start/Stop, File Upload, Status |
| Archival | File upload, TM/Recorder file transfer |
| Report Building | Aggregated: File name, Test result, Test context |
| Error Logging | All command/status messages plus errors |
| GUI | Full manual control, Simple action/colorized button set, Data and Result panels |

**Coordination of Services:** The TMSE uses a Test Configuration File to coordinate and control the test activities/services utilized by the Test Executive. It contains the specific telemetry related test details unlikely to change from test to test, reducing setup communication. This configuration file also contains the IP and port information required to coordinate network communication.

As an additional service the TMSE reports available drive space (TM station/card, Recorder, archival) before test execution. Prior to testing, the Test Executive can command the use of all or some of the services provided by the TMSE.

**Telemetry:** Direct control over the telemetry receiving/recording process (load telemetry project, name a telemetry file, running the telemetry card/station) plus monitoring the telemetry stream (Lock).

**Telemetry Data Exchange:** During test execution, the telemetry station/card is capable of broadcasting telemetry variables out to network users. This service controls the delivery of different telemetry variable sets to different numbered network ports.

The Test Executive is a consumer of real-time telemetry data, where it can monitor, evaluate and execute test paths using telemetry variable content. Via this service, data can be used to provide a Quick Look Summary of the current test or a Statistical Quick Look Summary across multiple test runs.

**RF Receiver:** This service will setup the receiver hardware consistent with the information in the Test Configuration File. During or after test execution the TMSE allows Change Messages from the Test Executive, which can alter the receiver settings (e.g. frequency).

**Report Building:** At the end of each individual test, the Test Executive can send the TMSE a text statement that conveys the test result. Each statement is associated with the telemetry file name, test context information and then aggregated into an overall Test Report file. This report allows post-test data reviewers to rapidly identify the particular files of interest for their evaluation.

**Data Archival:** As a background task, the TMSE will transfer the telemetry, recorder and Test Report files onto a large capacity, special-purpose archival drive and the telemetry file onto a high performance RAID drive for data visualization.

## SOFTWARE DESIGN

The TMSE application is network-centric with emphasis on hardware configuration flexibility and a simple, yet informative command/status message set for communication between the TMSE and Test Executive/GUI. As a TCP server, the TMSE is either in a "listen state" or in a "connection state" with a single client. Once the client connects and receives a handshake, the TMSE will wait for a command. When the client sends a command, the TMSE will process it and return a status message indicating the result. This command/status interaction will continue until the client transmits a termination command or disconnects.

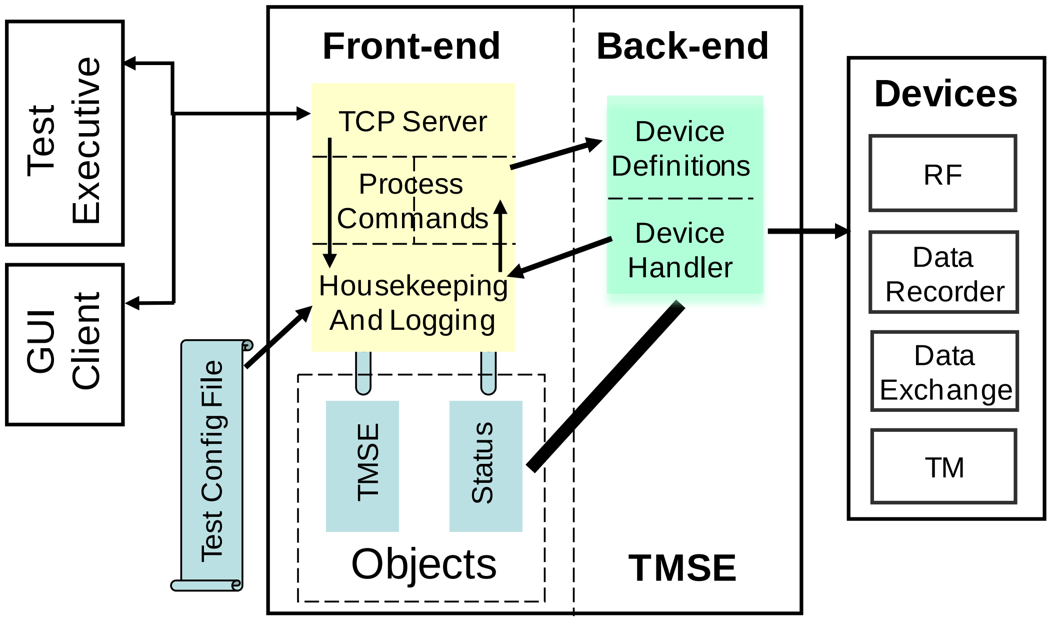
**Requirements:** This application was developed in VB.NET 2008 for a Windows environment using .net's interoperability. It can run on any system with .net 3.5 framework – also supported by Mono on most non-Windows operating systems. The client must be able to transmit and receive TCP/IP packets to/from the server. Once the client is configured and the TMSE server is running, an operator will not have to personally manage any telemetry hardware. All telemetry devices must be controllable across a network connection

**Modularity:** One of our goals was to make future modifications with greater ease. Thus, we used Conway's Law as our model: *focus on the form of what the system is, partitioning it so that each part can be managed as autonomously as possible.* (James O. Coplien)

The TMSE's architecture contains logical boundaries between the Front-end (Figure 3) which handles the commands/statuses and the Back-end modules handling how specific devices respond to these commands. The Front-end software remains unchanged across alternative hardware device configurations. It handles all the housekeeping from test setup through archival. The Back-end contains all specific vendor modules necessary to control/status each device across the network. The Back-end supports new code or current module modifications whenever a new telemetry device is added to the module base. This creates a flexible environment that enables users to modify hardware configurations with minimal software changes – once built a module can be reused with the same device in other test lab environments.

**Objects and Tracking:** To bridge the Front-end to the Back-end, we used two objects that keep track of information and process flow: the TMSE and Status Monitor objects. The TMSE object stores pertinent state information used by the devices such as file paths, connected device types, version information, test names, IP addresses, and various pass/fail flags. This object also contains housekeeping components to track each device's current state. The Status Monitor object is a stack with the latest status on top.

Figure 3: TMSE Architecture



**Communication:** In this design, explicit commands conveyed to the Back-end are immutable and always carry the same definition to each device type. Likewise, a status returned to the Front-end always uses the same verbiage. There are eight base commands (Table 3) which combine to control the hardware from pre-setup to termination. A message sent to the server can contain compounded commands. The commands can change device settings, configure applicable devices, start/stop a test, check status, build reports, and archive files. Each received client command returns a status upon:

1) Successful completion
2) Partial completion (compounded elements)
3) Error

The error message, in conjunction with the given command, will always be device specific to the occurring fault.

Table 3: Simplified Commands

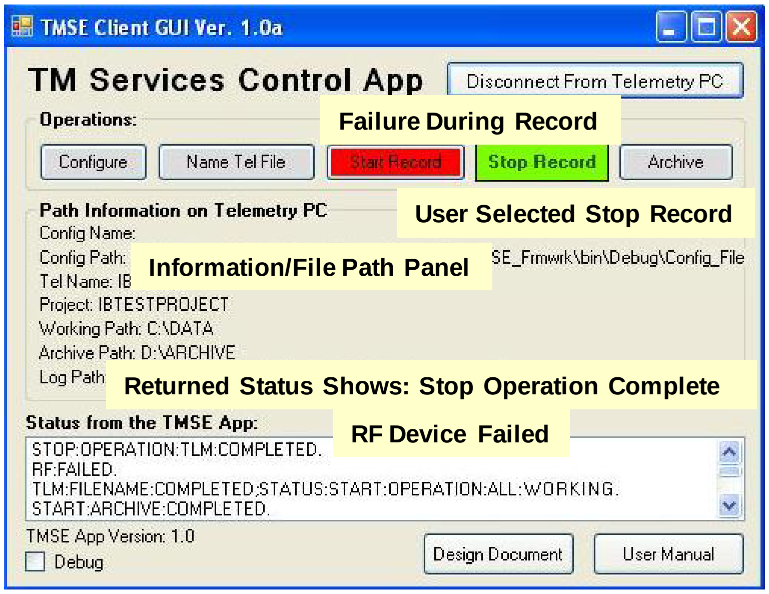
| Command Set | Returned Status |
|---|---|
| START | COMPLETED |
| CONFIG | FAILED |
| STOP | WORKING |
| TERMINATE | READY |
| CHECKSTATUS | MSGUNK |
| CHANGE | NOTCONFIG |
| RESULT | FILENOTFND |
| BUILDREPORT | DISKFULL |

**Built for Reuse:** Our design focus on Conway's Law helped us create a software architecture that inherently facilitates simple reuse. The code required for modification has to be easy or *people won't use it* (Andrew Hunt). Therefore, we set up the Front-end to process commands invoking service actions without issuing particular vendor device messages. For example, if the client uses a specific Data Recorder and needs to change a setting, it would send a Change (service) request to the TMSE. The TMSE Front-end will process this Change Message as an action to the DataRecorder object and not create a specific command to send out to the networked recorder device.

The particular Test Configuration file invoked, specifies what hardware vender profile to use. Upon first device command, the Front-end's object call will connect to the appropriate Back-end's module for that specific device. This kind of functionality enables a programmer to add new devices with very little Front-end code impacts.  Adding a new device module only requires one hook in the Front-end code, otherwise it is isolated from any other dependency.  This follows Edsger Dijkstra's modular principles where a module is optimal with only one entry point and one exit point (Dijkstra).

## GUI SOLUTION

A GUI provides an adjunct, standalone capability to testers when the automated Test Executive is not acting.  To overcome the skill barrier, users do not need to know how telemetry works or how the specific telemetry hardware needs to be operated.  This GUI resides on any networked PC and features several attributes that enable the user to control and evaluate the delivery of telemetry services.


TMSE Client GUI Ver. 1.0a
TM Services Control App
Disconnect From Telemetry PC
Operations:
Failure During Record
Configure
Name Tel File
Start Record
Stop Record
Archive
Path Information on Telemetry PC
User Selected Stop Record
Information/File Path Panel
Returned Status Shows: Stop Operation Complete
Status from the TMSE App:
RF Device Failed
Debug
Design Document
User Manual

Config Name:
Config Path: SE_Frmwrk\bin\Debug\Config_File
Tel Name: IB
Project: IBTESTPROJECT
Working Path: C:\DATA
Archive Path: D:\ARCHIVE
Log Path:

```
STOP:OPERATION:TLM:COMPLETED.
RF:FAILED.
TLM:FILENAME:COMPLETED;STATUS:START:OPERATION:ALL:WORKING.
START:ARCHIVE:COMPLETED.
```

TMSE App Version: 1.0

Figure 4: Standalone Control GUI

The Operational Buttons offer an easy way to linearly sequence through the delivery of telemetry services.  They dynamically change color to indicate if the associated action has been completed (green), in process (yellow) or failed (red).  Also as actions are completed, new buttons become actionable (light grey).  Panels provide test product locations and returned status.

In the GUI example (Figure 4) a recording failure is indicated (red) followed by a successful stop operation (green).  The Status Panel identifies a RF receiver device failure.

## RESULTS

During conceptual design, we began by thinking about Return On Investment (Sikka).  Our design target was a 60% reduction in test cycle time and a payback of two-years across all of our program's missile labs.

To evaluate our test cycle time reductions[iii] we performed a Taylor Time and Motion Study (L.C. Pigage) (Wikipedia).   In a manual, button-pushing environment, a telemetry operator must monitor the test, communicate with a test director, coordinate, execute an action and then verify it.  During automated test, there is no human motion plus the monitoring and communication are implicit in the encoded test script.

Table 4 aggregates our results across a single one-hundred test cycle Monte Carlo test event. We separated out single occurrence events from those that are repeated every test cycle, plus assumed five test reconfigurations (telemetry related device changes) during a test event. From these results we saw a 74% overall reduction (6.7 hours) going from 9.0 hours (manual) to 2.3 hours (automated). This result considerably exceeded our expectations.

Table 4: Taylor Time and Motion Study

| Event | **Manual Time (sec) | *Automated TMSE Time (sec) | Description |
|---|---|---|---|
| Setup For Test | 60 | 5 | Define/Coord test settings (once) |
| Configure | 118 | 15 | Execute setup (once) |
| Prepare Record File | 11 | 2 | Name file and path |
| Start Record | 3 | 2 | Start TM and recorder |
| Stop Record | 3 | 2 | Stop TM and recorder |
| Archive | 12 | 2 | Initiate telemetry file archive |
| Reconfigure | 35 | 5 | Change setup parameters (optional) |
| | | | |
| 100 Test Cycles (sec) | 3,253 | 845 | Assume 5 test reconfigures |
| 100 Test Cycles (Hrs) | 9.0 | 2.3 | |
| *Automated TMSE – execute | | 74% | Reduction: TMSE/Manual |
| **Manual – monitor, communicate, execute, verify | | | |

Looking at the cost/investment side of the equation, we summed up the time required to develop the TMSE at less than 150 single-person workdays. Given the demands of our design goals, we spent a very significant 19% of that total defining, designing, reviewing and demonstrating this product.

To evaluate our Return On Investment (Table 5), we considered a utilization scenario[iv] in which two labs with six total test stations were performing automated testing. Then assuming a test station utilization of 25% (labs are multi-tasked) running five days a week, for fifty weeks we were able to estimate the total time saved, based on our 6.7 hours per test event.

Table 5: ROI Calculation

| Elements | Aggregated Hours |
|---|---|
| Time Saved (Hrs) per One-Hundred Test Cycle Event per day | 6.7 |
| Six Test Stations | 40.2 |
| 25% Test Station Utilization | 10.1 |
| Five days a week for one year (50 Wks) | 2512.5 |
| Development cost (Hrs: 150 days * 8 Hrs/day) | 1200 |
| Payback Period (Months) | 5.7 |

Under this utilization scenario, our payback time is less than six months. This estimate did not include the labor savings from eliminating telemetry support personnel.

## CONCLUSION

The TMSE application allows a client to use a telemetry system as **just another networked test appliance**. It represents a major improvement in: ease of telemetry use, reduction in test cycle time, standardization of control and interface, plus elimination of the telemetry skill barrier. TMSE's architecture facilitates reuse, by simplifying the required modifications for different hardware vendors and integrating easily with different automated Test Executives.

The results presented here exceeded our expectations. An investment in this type of capability will yield a very significant reduction in overall test cycle time plus a very rapid payback. Our next planned activity will be adding a service that allows the Test Executive to command the TMSE to immediately startup our post-test data visualization tool with a telemetry file of its choosing.

## REFERENCES

Hunt, Andrew and Thomas, David, The Pragmatic Programmer: From Journeyman to Master, 1st Edition, Addison-Wesley, 1999.
Dijkstra, Edsger. "Go To Statement Considered Harmful." Communications of the ACM 11.3 March (1968): 147-148.
Hayes, Linda G. The Automated Testing Handbook, Software Testing Institute, 2004.
Coplien, James O. and Bjornvig, Gertrud, Lean Architecture for Agile Software Development, 1st Edition, Chester: Wiley and Sons, 2010.
Pigage, L.C. and Tucker , J. L., Motion and Time Study, Vol. 1st. University of Illinois, 1954.
National Instruments, Software Defined Test Fundamentals, 2010.
<ftp://ftp.ni.com/evaluation/ate/software_defined_test_fundamentals_guide_preview.pdf>.
Pettichord, Bret, Seven Steps to Test Automation Success, 26 June 2001.
<http://www.io.com/~wazmo/papers/seven_steps.html>.
Sikka, Vijay, Maximizing ROI on Software Development, Auerbach Publications, 2004.
Wikipedia, Raytheon Missile Systems, 20 Feb 2011.
<http://en.wikipedia.org/wiki/Raytheon_Missile_Systems>.
—. Time and motion study,13 April 2011.
<http://en.wikipedia.org/wiki/Time_and_motion_study>.

## END NOTES

[i] Raytheon manufactures a large number of missile/projectile products most of which utilized telemetry during their development.
[ii] Invoking a complete set of design considerations was the most valuable contributor to our results
[iii] Accurately evaluating test cycle time reduction is critical to establishing ROI
[iv] This utilization scenario would be typical of early product development.