

"TELSIM: REAL-TIME DYNAMIC TELEMETRY SIMULATION ARCHITECTURE USING COTS COMMAND AND CONTROL MIDDLEWARE"

Rodney Davis, & Greg Hupf

Command and Control Technologies, 1425 Chaffee Drive, Titusville, FL 32780, 321-264-1193
davisrd@cctcorp.com, hupfg@cctcorp.com

ABSTRACT

Traditionally, approaches to telemetry “simulation” do not provide the range of behavior dynamics needed for thorough ground systems testing, mission dress rehearsals, and operator training and certification. CCT and NASA have collaborated to provide a telemetry simulation capability to enhance personnel readiness without impacting availability of operational assets. One that can accommodate a full range of realistic nominal and non-nominal mission scenarios in a network distributed operational environment based on a modular COTS command and control middleware and plug-in continuous or discrete-event interactive modeling environment.

KEY WORDS

Simulation, Telemetry, Command, Control, Middleware

INTRODUCTION

Traditionally, telemetry “simulation” is performed using playback of previously recorded data or rudimentary pattern generation. Using this static telemetry strategy for testing is limited, and does not provide the range of data dynamics needed for thorough ground systems testing without expending significant effort to generate test data sets. The limitations of static telemetry simulation are even greater when applied to mission dress rehearsals and operator training and certification. Operational organizations need a capability to enhance personnel readiness without impacting availability of operational assets, one that can accommodate a full range of realistic nominal and non-nominal mission scenarios, in a distributed operational environment.

In collaboration with NASA, CCT has developed the TelSim Real-time Dynamic Telemetry Simulation Architecture to address these operational simulation needs as well as address several architecture quality objectives. TelSim is based on a modular COTS Command and Control Middleware architecture, providing a fully integrated simulation environment that can model the continuous or discrete time behavior of any external system. Models respond to event stimulus

with simple or complex data behaviors that are commutated in to a telemetry stream in real-time as depicted in Figure 1 below. Some of the projected benefits of TelSim are:

- Reduce time to achieve and maintain operator competency
- Improve operator performance
- Provide a means of measuring operator competence
- Provide an engineering analysis tool for refinement of ground systems technologies, processes, and methods of operation

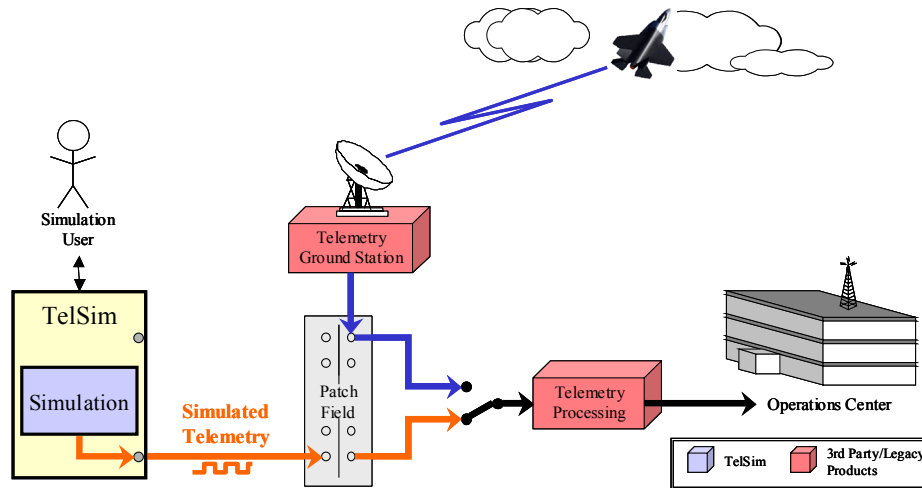


Figure 1 TelSim Concept

The TelSim provides real-time generation of IRIG 106 compliant telemetry, and is capable of fully independent operation that can be plugged-in to any telemetry ground station as a headless server, or with a fully interactive distributed multi-user graphical user interface. The integrated simulation environment is user configurable to facilitate:

- Missing element models for test and integration
- Interactive operations crew training (nominal & non-nominal dynamics)
- End-to-end mission simulations and dress rehearsals

The TelSim modeling engine allows a systems engineer to quickly create interactive models using a point and click GUI or high level scripting language in real-time or predefined. A programming library is also available that supports the integration of 3rd party modeling tools such as MATLAB, or custom developed software models. The environment allows simulations to be scalable in their fidelity so they can operate stand alone, or integrated with other models to create large-scale mission simulations. TelSim also supports telemetry acquisition, processing, and recording for mission playback and replay of post decom data with simulation models.

BODY

The TelSim concept originated out of the NASA payload test community that needed a reconfigurable test article for preparation and checkout of complex spacecraft; a flexible tool that could support verification of mission databases, ground system configurations, data processing, and command and control software and displays. It was also a design goal to implement a capability that could be extended beyond telemetry to support other communications protocol simulations, such as avionics and ground support equipment buses, without significantly altering architecture. It was important for us to define an architecture that could be used to build complete new systems (i.e. includes ground station, data processing, simulation and training, operations and control center, etc.).

Rather than embarking on a significant custom software development effort to build TelSim from the ground up, a middleware based approach was selected that allowed the development team to significantly leverage COTS technology, while also providing significant latitude for customization and extension for meeting unique requirements. The initial R&D effort required definition of an overall SW architecture that would support seamless integration of all aspects of telemetry processing using modular and distributed components. The result was an architectural pattern. Once this framework was defined, the primary objective was to implement a robust dynamic telemetry simulation capability that could take advantage of the approach. The bulk of the R&D effort was creating a generic commutator that could support any class I or II IRIG telemetry format. The remaining effort focused on a configuration approach that integrated existing components of the COTS command and control middleware.

There are significant benefits from using a modular distributed SW arch and interoperable COTS components that communicate via common middleware. For example, telemetry data acquisition, transmission, processing, simulation and modeling all use common SW components. Minimizing the number of disparate components reduces complexity and enables a common configuration approach. Modular distributed architecture is scalable and flexible, allowing more efficient use of personnel and allocation of computer resources. Adapting to new requirements and incorporating new features is more efficient, reducing O&M costs and increasing system lifespan.

COTS Command and Control Middleware

The Command and Control Toolkit™ (CCTK) from Command and Control Technologies Corporation (CCT) is middleware for creating custom control systems. Shown in Figure 2, CCTK was chosen for the TelSim architecture framework because of its integrated real-time simulation engine and modeling language. The combination of configurable real-time parameter services, command and control visualization, and simulation tools made CCTK an ideal platform for building an interactive telemetry simulator. CCTK also included support for telemetry acquisition/decommutation, meaning most of the infrastructure for describing telemetry formats and data processing algorithms already existed and would simply need to be extended for simulation.

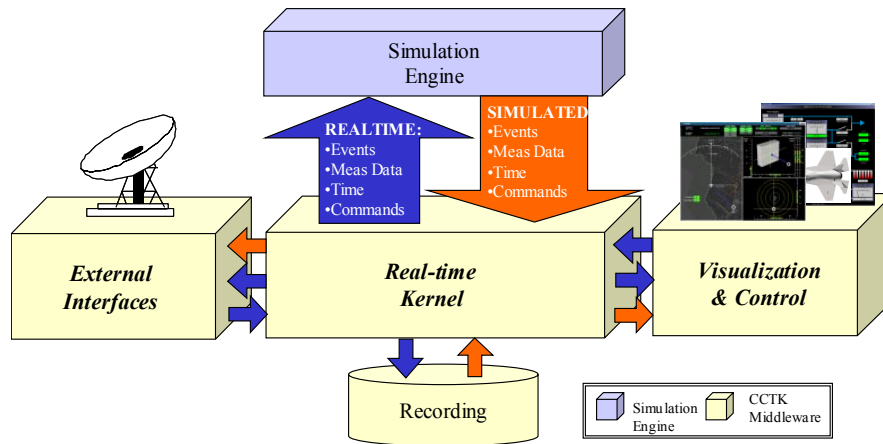


Figure 2 Command and Control Middleware Concept

CCTK provides the following major functional features for the TelSim Telemetry Simulator:

- Built in applications for visualization and control such as user configurable quick look displays, messaging, and health monitoring.
- Application development environment that includes C/C++ CCTK programming library, scripting environment based on Tcl.
- GUI builder for creating dynamic graphical control and monitoring displays.
- An integrated simulation environment that supports user application development, missing element modeling, and user training scenarios.
- Master time acquisition, synchronization, distribution, and general-purpose user timer.
- Modular data processing scheme that allows processing algorithms to be dynamically chained together and extended with user custom modules.
- XML configuration databases for management of interfaces, commands, measurements, and resources in an open, extensible manner.
- Digital command, measurement, and message recording, and retrieval for post mission data reduction and analysis.
- Network support services for peer-to-peer and client server operation

CCTK is a generic, re-configurable tool-suite that can perform a variety of control and monitoring tasks.

Modeling Environment

The CCTK simulation function provides a flexible means of creating simulations that can scale from simple single sensor manipulation to fully interactive behavioral models. Models created in this environment execute concurrently and work interchangeably with real-time applications and external interfaces as shown in Figure 3.

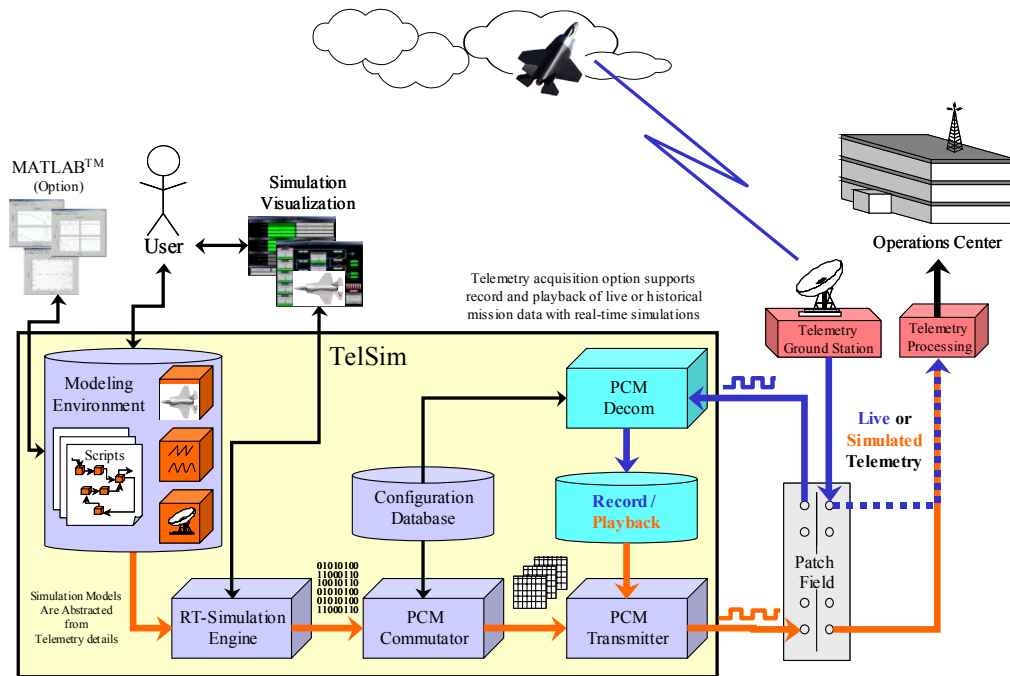


Figure 3 TelSim Architecture

Simulations can be “played” such that operators, mission application software, and visual displays are unaware whether they are interacting with live or simulated data/commands. The Simulation Engine allows a systems engineer to quickly create interactive models using a point and click GUI or high level scripting language. A programming library is also available, as part of the CCTK Development Environment, that supports the integration of 3rd party modeling tools or custom developed software models. The environment allows simulations to be scalable in their fidelity so they can operate stand alone, or integrated with other models to create large-scale mission simulations.

Data and commands exist within the simulation environment as time based discrete samples that are normalized and available for distribution throughout CCTK. Simulated data appears to the rest of CCTK as though it is originating from real-time external sources (Telemetry, Radar, etc.)

As illustrated in Figure 4, the CCTK simulation engine supports a real-time, or user defined time base. It can run concurrently with real-time tasks, facilitating integration with live operations such as missing-element integration testing; or use simulated time as it’s primary event mechanism, sequencing behaviors at predefined times with configurable time propagation rates.

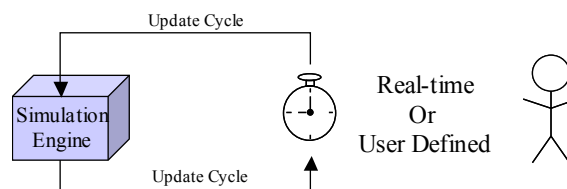


Figure 4 User Configurable Simulation Time

The environment provides tools for defining simulation behaviors and waveforms to associate with data (measurements), including:

- Constant
- Sine
- Saw Tooth
- Random
- Data file
- Jitter
- Square
- Complex expression
- Ramp
- User custom

Data sample rate, amplitude, offset, period, bias, delay etc. are user configurable. Data are organized in to named *groups* that trigger behaviors in response to events such as time, commands, or other group's behaviors. *Groups* exhibit control and/or instrumentation behavior of systems; e.g. when the power supply group receives the *power-on* command, initiate group behavior (ramp voltage and current to nominal no load level); or at CDT = 000:00:00.00, start radar tracking ephemeris data file.

Simulations are composed from three basic building blocks: Measurement Modules, Command Modules, and Event Modules. Each module, described in Figure 5, produce behaviors in response to input stimulus. Behaviors can be discrete events, or continuous.

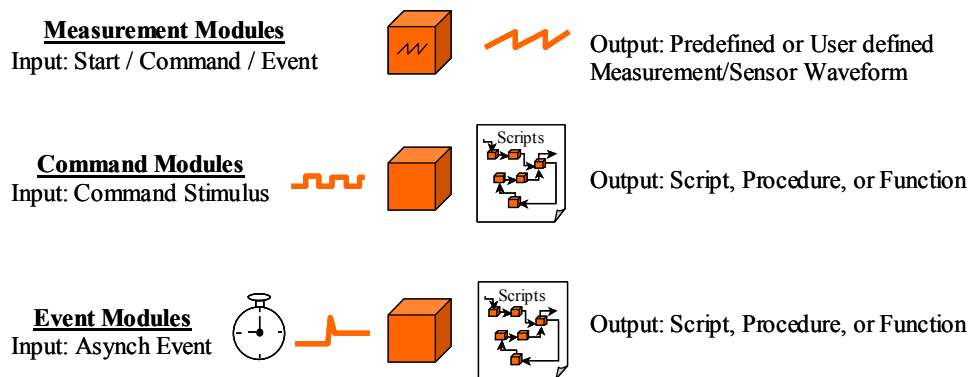


Figure 5 Simulation Engine Module Types

The following simple simulation script starts six standard waveforms simulations and runs 1 Hz for 30 seconds then stops. The resulting simulation waveforms are shown in Figure 6.

```

Sim_NewMeas -type rand -name "Analog Sensor #1" -upper 100.0 -lower 0.0 -update 20

Sim_NewMeas -type ramp -name "Analog Sensor #2" -initial 10.0 -final 83.5 -period 6

Sim_NewMeas -type saw -name "Analog Sensor #3" -period 1 -amplitude 20.0 -offset 50 -force true

Sim_NewMeas -type sine -name "Analog Sensor #4" -period 3.0 -amplitude 60.0 -bias 0.0 -offset 0.0

Sim_NewMeas -type square -name "Analog Sensor #5" -period 5.0 -upper 90.0 -lower 50.0

Sim_NewMeas -type delim -name "Analog Sensor #6" -delim "," -tags VALUE -file /cct/simfile.dat -repeat true

Sim_Run 1 30

```

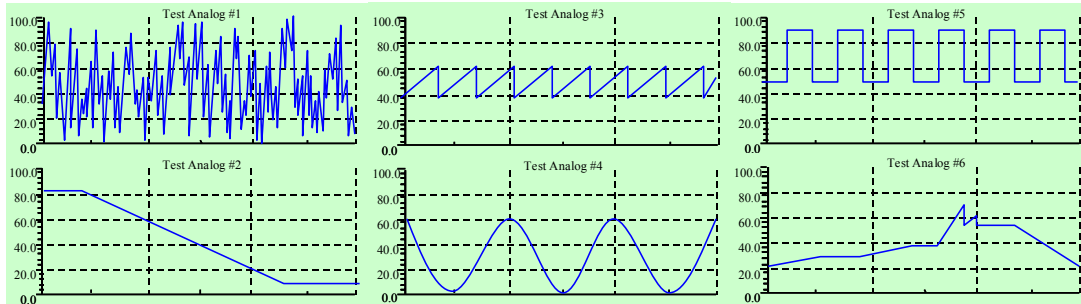


Figure 6 Measurement Simulation Script Output Waveforms

The following simple simulation script uses a command simulation module to toggle between **Fill** and **Drain** Tank Level simulations. Commands can originate from a network, uplink I/F, user GUI, or other simulation models or applications. The command feature combined with the graphic user interface enables the simulation engine to be used for creation of interactive training and/or scenario control. In this example, the *\$currentState* parameter is set to *Fill* or *Drain* by a graphical command widget as shown in Figure 7.

```

Sim_NewCmd -type tcl -name "Tank Fill Drain - Toggle Fill Valve" -script {
    set currentState [Sim_QueryMeas -name "Tank Fill Drain - Fill Valve State" -query value]
    if {$currentState == "Fill"} {
        Sim_NewMeas -type ramp -name "Tank Level" -initial 0.0 -final 99.8 -period 0.5
    } else {
        Sim_NewMeas -type ramp -name "Tank Level" -initial 99.8 -final 0.0 -period 0.5
    }
}

```

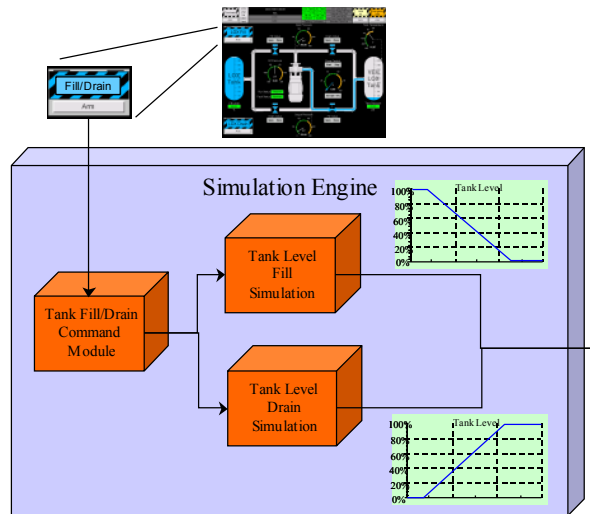


Figure 7 Example Command Simulation

The simulation language is based upon Tool Command Language (Tcl), so all valid Tcl commands and syntax are valid within the simulator. User defined modules based on Tcl, C, or C++ can be integrated with the simulation environment, returning simulated values based on user-defined algorithm. Tcl is a simple-to-learn yet very powerful non-proprietary language. Its

syntax is described in just a dozen rules, but it has all the features needed to rapidly create useful programs on a wide variety of international platforms.

A point and click graphical user interface supports creation of simulations on the fly. Shown in Figure 8, it is a simple alternative to scripting or full language modeling, dynamically creating executable simulation models at run time or off-line. Models can be saved to a file for execution later or outside of the GUI. These files are actual TCL simulation scripts that can be edited outside of the GUI.

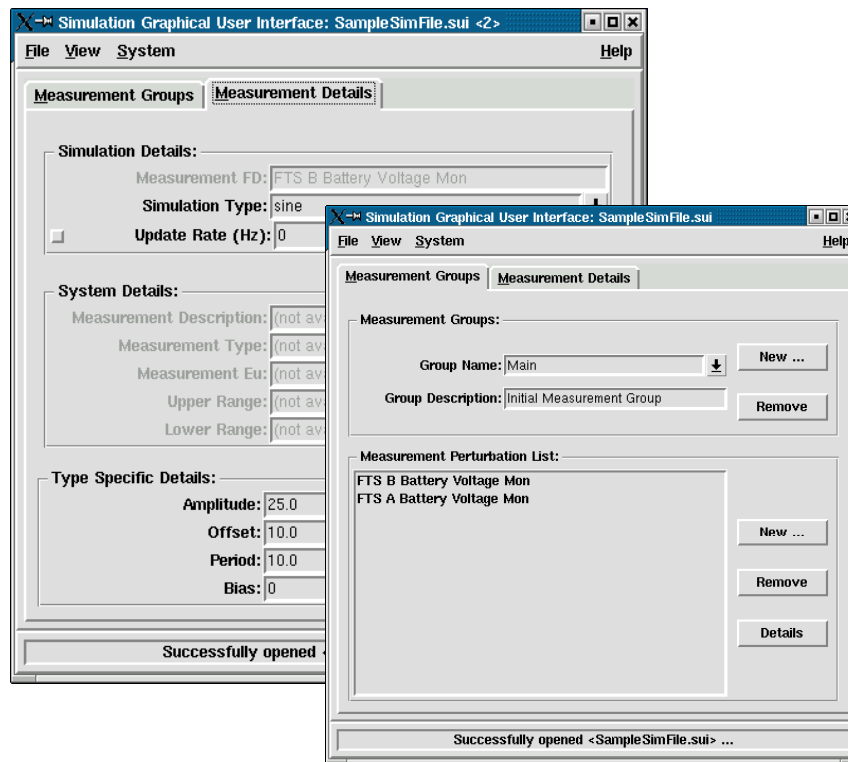


Figure 8 Simulation Graphical User Interface

The example simulation shown in Figure 9, contains a script that plays a delimited data file of radar data. The radar's data is contained in a separate file that is a “\” delimited list of fields. Each line in the file is a record and each field is a measurement value. This script uses the "-type delim" and “force true” options to send the values in a record at the specified update rate of 10Hz.

```

Sim_NewMeas -type delim -name RadarSrc2
-file "/sim/RadarSrc2.dat" -tags [list \
    "R2 - Day of Year" \
    "R2 - Time of Day (tenths)" \
    "R2 - Time of Day (seconds)" \
    "R2 - Azimuth" \
    "R2 - Elevation" \
    "R2 - Range" \
    "R2 - Quality" \
    "R2 - Mode" \
    ""]
#
# Update at 10 Hz
Sim_Run 10

```

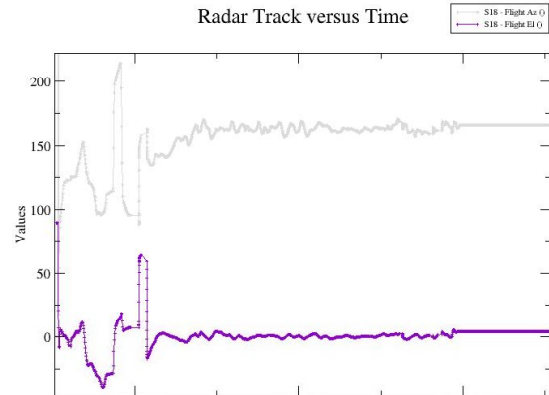


Figure 9 Radar TSPI Simulation Example

The previous TSPI example is extended in Figure 10 to simulate 4 different radars. Each radar vector record is played according to the simulated mission time value (CDT).

```

set radars [list 2 3 5 18]
#
# Simulate the radars
#
foreach radar $::radars {
    Sim_NewMeas -type cdt -name RadarSrc${radar}
    -file "/sim/RadarSrc${radar}.dat" -tags [list \
        "R${radar} - Day of Year" \
        "R${radar} - Time of Day (tenths)" \
        "R${radar} - Time of Day (seconds)" \
        "R${radar} - Azimuth" \
        "R${radar} - Elevation" \
        "R${radar} - Range" \
        "R${radar} - Quality" \
        "R${radar} - Mode" \
        ""]
}
Sim_Run 10

```

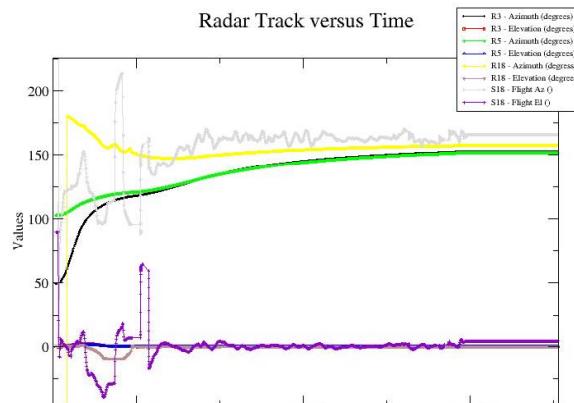


Figure 10 Multi-Radar TSPI Simulation Example

The **group** construct in the simulation engine supports hierarchical organization of the individual simulations modules, such as those shown in the previous examples. Named groups respond to event stimulus with the defined collection of simulation module behaviors. Groups can be orchestrated in to an interacting network of system-of-systems simulations (e.g. propulsion, navigation, power, hydraulics, etc.). CCTK middleware routes real-time simulated data to the PCM commutator, which maintains protocol synchronization and transmission of simulated data according to the defined telemetry format.

Architecture Considerations

The TelSim was implemented using commodity PC hardware, COTS PCM simulator device, and Linux. Consideration was given to use of a real-time OS for performance reasons, but simplicity

and portability were considered to be more important quality attributes, and porting to an RTOS at a later time is feasible should it become necessary. Also, Linux performance latency issues were mitigated somewhat by the modular approach to simulation, commutation, and interface processing all running as independent real-time tasks so there were few interlocking synchronous deadlines to meet. The need for hard real-time support was mitigated further by availability of low cost processing power and continued Linux maturation over the life cycle of the project.

A layered software architectural approach provides separation of concerns and encapsulates significant features. A pipes-and-filters architecture pattern was used to modularize and distribute components. The middleware services participate in the processing or workflow, as well as providing connectivity between data processing modules, external data sources and the user interface. Other architectural considerations include the following: System is easier to validate and certify when the simulation capability is inherently part of the architecture. Architectural approach does not require reconfiguration to inject simulated data. The same primary software components are always used, regardless of whether simulated data is being used.

CONCLUSIONS

The TelSim middleware approach proved to be a sound strategy for creating a complex test article for telemetry simulation applications. Using the command and control middleware saved thousands of development hours and simplified the scope of the overall systems design.

CCT has taken advantage of the TelSim development success and incorporated it in to its COTS command and control product line, where they have continued to mature and extend its capabilities.

Model-based simulation offers a robust alternative to simple data playback for telemetry and range testing and validation. Integrating dynamic modeling into telemetry operations can significantly improve the fidelity of systems test and training by enabling greater dynamic control of mission scenarios, evaluation of failure modes, and range of conditions.

For training, model-based simulation can help create realistic scenarios that unfold according to an orchestrated script, or under the control of a simulation operator that can manipulate model behavior dynamically. Training models can focus on the roles of each operator and push the dynamics that drive human and system behavior and performance. Instructors can simulate various failure modes under realistic conditions to demonstrate and practice response procedures.

REFERENCES

- Range Commanders Council (RCC), IRIG Standard 106-01, <http://jcs.mil/RCC>
- “Software Architecture,” CCTK User’s Manual, Command and Control Technologies Corp., Ver 2.1.
- Brown, K., Davis, R., “Intelligent Launch and Range Operations Testbed,” STAIF 2002