Implementing a PLUTO-like script engine for testing the Korea Pathfinder Lunar Orbiter simulator

Dawoon Jung, Hoonhee Lee, Cheol Hea Koo

Korea Aerospace Research Institute, 169-84 Gwahak-ro, Yuseong-gu, Daejeon 34133, South Korea
Email: dwjung@kari.re.kr, lhh@kari.re.kr, chkoo@kari.re.kr

INTRODUCTION

A simulator may need to integrate together models of varying degrees of completeness depending on the mission phase. For example, a reasonably complete space environment model might be available even at the early conceptual design phase, while a payload model might only become available at a later phase. The Space System Model (SSM) concept [1] provides a solution to this dilemma by virtualizing space and ground segment components.

The Procedure Language for Users in Test and Operations (PLUTO) provides a means of scripting test and operations procedures. The language itself is a reference implementation of ECSS-E-ST-70-32C [2]. Such a scripting system, together with an SSM, suggests a flexible way to simulate and validate early mission concepts.

The Korea Pathfinder Lunar Orbiter (KPLO) mission will gather lunar surface imagery and scientific data from lunar orbit. The mission has been under development at the Korea Aerospace Research Institute (KARI) since 2016. As it is still in its early design stages, only a dummy spacecraft simulator is available. To compensate, we developed a script engine that partially implements ECSS-E-ST-70-32C. This script engine reads scripts written in JavaScript and provides built-in functions that map intuitively to PLUTO language constructs. A man-machine interface (MMI) was also developed that allows the user to edit script code with automatic completion, view elements in an SSM tree, and perform basic line-by-line debugging.

We use the script engine to simulate a camera slewing maneuver of the KPLO spacecraft. Our results show that our engine can be used to perform scripted tests even with partially-implemented or non-existent space and ground subsystems and simulators. By adopting JavaScript, we minimized bugs and shortened development time by building on a proven language base. The ease of script development also allowed us to readily test a new algorithm for performing online updates of simulator parameters based on simulated telemetry inputs.

Previous examples of PLUTO engine implementations include Manufacturing and Operations Information System (MOIS) and Integrated Development and Validation Environment for Operations Automation (IDEA) [3]. However, we required a lightweight system that we could customize freely and rapidly integrate into our mission design environment.

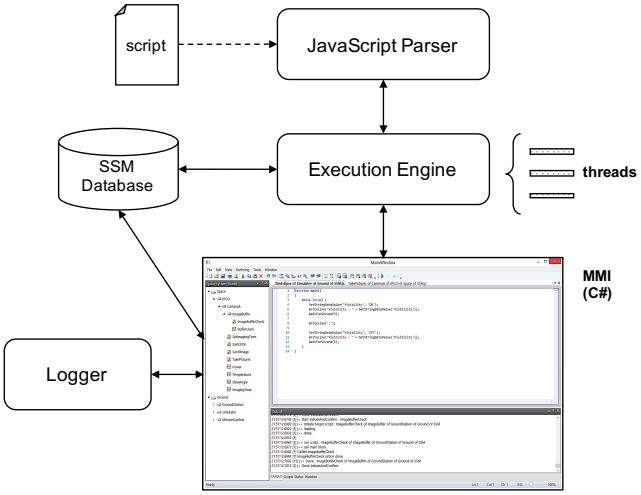
In the Implementation section, we discuss design and implementation details. The Results section presents our results, and the Conclusion section gives concluding remarks.

IMPLEMENTATION

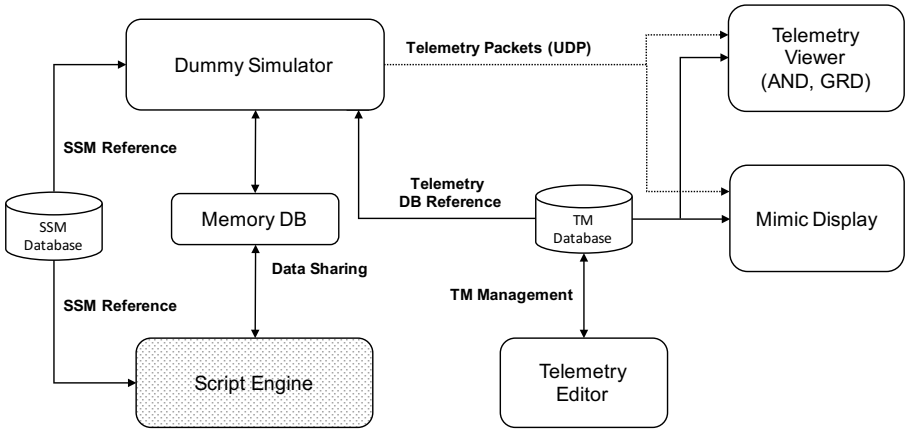
Design

Our script engine uses an off-the-shelf JavaScript interpreter that parses and runs standard JavaScript files (Fig. 1a). Jint [4] was chosen as we required an actively-maintained and stable, ECMAScript 5.1 [5] compliant C# component. We provide a library of pre-defined functions that can be used within a script to partially emulate PLUTO functionality. This is discussed further in the JavaScript - PLUTO mapping section.

An execution engine loads an SSM defined in a database and executes the script via the JavaScript interpreter. We added support to launch scripts concurrently via initiateAndConfirm() on multiple threads.



(a) System components.



(b) Test Architecture.

Fig. 1. System Block Diagrams.

Scripts can be edited and the SSM viewed via an MMI implemented in C#. The editor provides basic syntax highlighting. Multiple scripts can be initiated simultaneously, and also debugged line-by-line. Finally, the MMI displays logging output.

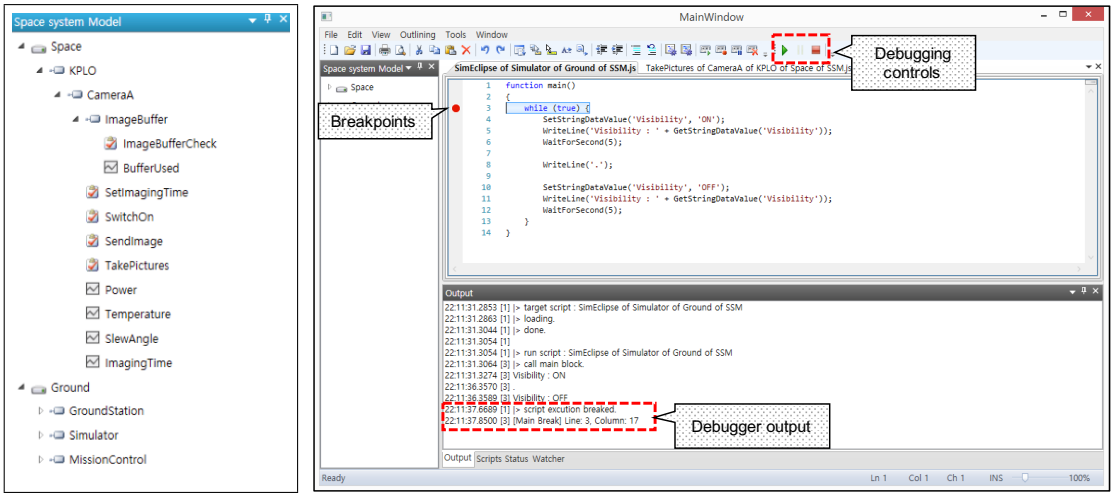
The overall architecture of our test system is shown in Fig. 1b. Our script engine interacts with a dummy KPLO spacecraft simulator using an in-memory database. The dummy simulator reads telemetry definitions from a database and outputs telemetry packets via User Datagram Protocol (UDP) to a telemetry viewer with alphanumeric and graphical displays (ANDs, GRDs). The dummy simulator also displays information on a mimic display.

JavaScript - PLUTO mapping

We wrote a library of JavaScript functions similar in names and functionality to a subset of PLUTO statements. This was done to facilitate compliance with ECSS-E-ST-70-32C during development, and lower the learning curve for users already familiar with PLUTO. Examples of such mappings are provided in the Appendix.

Databases

As already seen in Fig. 1b, our dummy simulator and script engine connect to an SSM database which must be populated offline, for example using a spreadsheet. Once populated, the SSM is displayed in tree form in the script editor MMI (Fig. 2a). In our dummy simulator, we allowed the user to map telemetry mnemonics read from a telemetry database



(a) SSM Tree Viewer. (b) Script debugger.

Fig. 2. Script Engine MMI.

with SSM elements via a mapping MMI. The mapping is stored in an in-memory database; this allows for efficient, safe concurrent updates of mappings even with the simulator and script engine running as separate applications.

Debugging

The script engine MMI allows the user to click in the script editor margin to set breakpoints, and run, pause, and stop scripts (Fig. 2b). Dragging and dropping SSM elements onto a watcher pane causes the current values to be watched. A logging pane displays timestamps, script output, engine status (e.g., stopped at a breakpoint), and thread identification numbers to facilitate debugging. As the language is JavaScript, a separate, lengthy compilation step is not required.

RESULTS

We used our test system to validate a new algorithm for performing operational simulator updates in real-time. Previously, operational simulators had parameters that were fixed at delivery time, resulting in simulator output that drifted away from telemetry during routine spacecraft operations. The premise of this algorithm is to minimize the difference between state reported by telemetry and simulator state, by feeding back telemetry into the simulator and automatically adjusting simulator parameters or output to track telemetry [6].

Listing 1 represents an excerpt from a script that animates a slew angle telemetry mnemonic bound to an SSM reporting data element named SlewAngle. In PLUTO, this element would be referred to as `SlewAngle of CameraA of KPLO of Space of SSM` (for brevity, we only show code for the rising edge of the sawtooth). Listing 2 represents partial code for a simulator model of the telemetry that outputs SlewAngleSim on the ground segment (`SlewAngleSim of CameraA of PAYLOAD of Simulator of Ground of SSM`). Listing 3 is part of a script that periodically reads SlewAngle telemetry, and updates a parameter named SlewAngleOffset of the simulator model if the difference with SlewAngleSim exceeds a threshold. This causes the simulator output to periodically converge with the reported telemetry. Note the use of PLUTO-like SSM reference strings; our script engine parses these in the expected way. Fig. 3 shows that the simulated slew angle (dashed line) tracks the reported telemetry (solid line).

Listing 1. UpdateSlewAngle of CameraA of KPLO of Space of SSM

```
var value = 0.0;
while (true) {
  value += 0.3 + 0.3*Math.random();
  SetRealDataValue('SlewAngle', value);
  WriteLine('SlewAngle ' + value);
  WaitForSecond(1);
}
```

Listing 2. UpdateSlewAngleSim of CameraA of PAYLOAD of Simulator of Ground of SSM

```
var value = 0.0;
var offset = 0.0;
var result = 0.0;
while (true) {
  value += 0.3;
  offset = GetRealDataValue('SlewAngleSimOffset');
  result = value + offset;
  SetRealDataValue('SlewAngleSim', result);
  WriteLine('SlewAngleSimOffset ' + offset);
  WriteLine('SlewAngleSim ' + result);
  WaitForSecond(1);
}
```

Listing 3. UpdateStateIfNecessary of State of Simulator of Ground of SSM

```
function main() {
  function updateSimState() {
    SetRealDataValue('SlewAngleSimOffset of CameraA of PAYLOAD of ' +
                     'Simulator of Ground of SSM',
                     GetRealDataValue('SlewAngleSimOffset of CameraA of ' +
                                      'PAYLOAD of Simulator of Ground of SSM') +
                     (previousState.SlewAngle - currentState.SlewAngle));
  }
  while (true) {
    getSimState();
    getRealState();

    if (needUpdate()) {
      updateSimState();
    }

    WaitForSecond(3);
  }
};
```

CONCLUSION

We demonstrated a JavaScript script engine that partially implements ECSS-E-ST-70-32C and is based on PLUTO. We used the script engine to simulate a camera slewing maneuver, using scripts to generate both reported telemetry and simulator model output. We were able to readily test a new algorithm for performing online updates of simulator parameters based on telemetry inputs. Our results show that our engine can be used to perform scripted tests even with partially-implemented or non-existent space and ground systems and simulators. Currently we do not implement engineering units, activation modes, metadata such as validity period, etc. Further work could focus on fuller coverage of ECSS-E-ST-70-32C and expand the use of the script engine into later design and verification stages.

ACKNOWLEDGMENTS

This work was performed by KARI under a contract with the Ministry of Science, ICT and Future Planning through the “Development of Pathfinder Lunar Orbiter and Key Technologies for the Second Stage Lunar Exploration” project (No. SR17026).

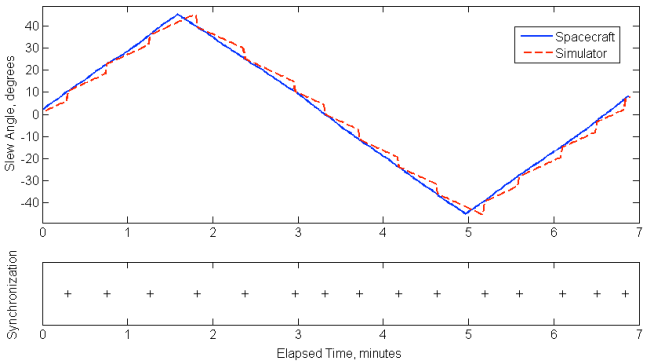


Fig. 3. KPLO Slew Angle Model Update Simulation.

REFERENCES

[1] ECSS Secretariat (ed.), Space engineering — Ground systems and operations - Monitoring and control data definition, ECSS-E-ST-70-31C, 2008.

[2] ECSS Secretariat (ed.), Space engineering — Test and operations procedure language, ECSS-E-ST-70-32C, 2008.

[3] Pearson, S., Trifin, F., Reid, S., and Heinen, W. Integrated Development and Validation Environment for Operations Automation, Proceedings of SpaceOps 2012.

[4] Ros, S., Jint Source Code, https://github.com/sebastienros/jint

[5] European Association for Standardizing Information and Communication Systems (ECMA). ECMA-262: ECMAScript Language Specification, Edition 5.1, 2011.

[6] Lee, H., and Jung, D. Conceptual Design and Implementation of an Integrated Database for Automatic State Synchronization between Spacecraft and Simulator, Proceedings of SpaceOps 2016.

APPENDIX

Examples of JavaScript library functions provided by our script engine and equivalent PLUTO statements.

Table 1. Pluto Blocks vs JavaScript Functions.

PLUTO Block Type	JavaScript Function
DECLARE	declareEvents( )
PRECONDITION	precondition( )
MAIN	main( )
WATCHDOG	watchdog( )
CONFIRMATION	confirmation( )

Table 2. Initiate Functions.

JavaScript Function	PLUTO Equivalent
InitiateAndConfirm(script)	initiate and confirm script
InitiateAndConfirm(script, params)	initiate and confirm script with params end with

Table 3. Local Event Functions.

JavaScript Function	PLUTO Equivalent
CreateLocalEvent(eventName, description)	declare event eventName described by description
FireLocalEvent(eventName)	raise event eventName
IsRegisteredEvent(eventName)	None (use to check for event existence)
IsEventFired(eventName)	None (use WaitForTime() and IsEventFired())

Table 4. Reporting Data Functions.

JavaScript Function	PLUTO Equivalent
IsValidDataKey(key)	get validity status of key
GetRealDataValue(key)	get value of key (as Real)
SetRealDataValue(key, value)	set value of key with value (Real)
GetAbsoluteTimeDataValue(key)	get value of key (as absolute time)

Table 5. Utility Functions.

JavaScript Function	PLUTO Equivalent
SetContext(ctxt)	in the context of ctxt do
EndContext(ctxt)	end ctxt
WaitUntil(expression)	wait until expression
WaitForSecond(xx)	wait for xx s
Write(string)	inform user string
DebugWriteLine(string)	log string