An Environment for Incremental Development of Distributed Extensible Asynchronous Real-time Systems

Charles K. Ames, Scott Burleigh, Hugh C. Briggs
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

Brent Auernheimer
Department *of Computer Science, California State University, Fresno, CA 93740-0109*

Abstract

Incremental, parallel development of distributed real-time systems is difficult. Architectural techniques and software tools developed at the Jet Propulsion Laboratory's (JPL's) Flight System Testbed (FST) make feasible the integration of complex systems in various stages of development. In particular, two techniques are used: a strict layering architecture for organization of independent subsystems, and a distributed, low overhead, asynchronous messaging system. These techniques were developed in a few user-months and have proven their usefulness in a spacecraft integration test and simulation environment.

introduction

A goal of the Flight System Testbed (1'S'1') is to generalize and opt im ize system-level spacecraft interfaces in support of rapid prototyping and integration testing [FST, ['194]. Using software to simulate the spacecraft and environment al arbitrary levels of abstraction is the theme of this paper.

A snapshot of a spacecraft under development includes a combination of hardware engineering 1110(ICIS, breadboards, brass boards, and software simulations. The FST development environment includes techniques for the smooth replacement of software simulations with hardware or flight software as it becomes avail able. Th is facilitates complex hardware-in-the-loop simulations.

1 n part cular, an arch itecture was developed to support rapid prototyping of planetary spacecraft systems. This involves encapsulating core on-board services required of any spacecraft (pointing, command handling, telemetry storage, etc.) and simulating a flight-like environment. Simulating the motion of a spacecraft or the output of a camera in real-tilnc is a substantial task. Our approach is to collect and integrate simulation systems that model the in-flight environment.

Our simulation architecture is a layered ball's-eye, or onion pattern reminiscent of traditional uniprocessor operating system design [Tan87]. The center is the system under test. The first layer out is the interface driver layer. 'I'his shaded layer presents services to the core system that will remain constant as outer layer simulations are replaced with real devices. For example, in Figure 1 the core cock under test might be a spacecraft attitude control system, and tbc first layer out might provide the core system with a `gyro_get_state()` function that is initially implemented as a remote procedure cal l (RPC) to a real-time dynamics model simulating spacecraft motion. Th is function might later be implemented to send a message across a bus to spacecraft rate sensors.

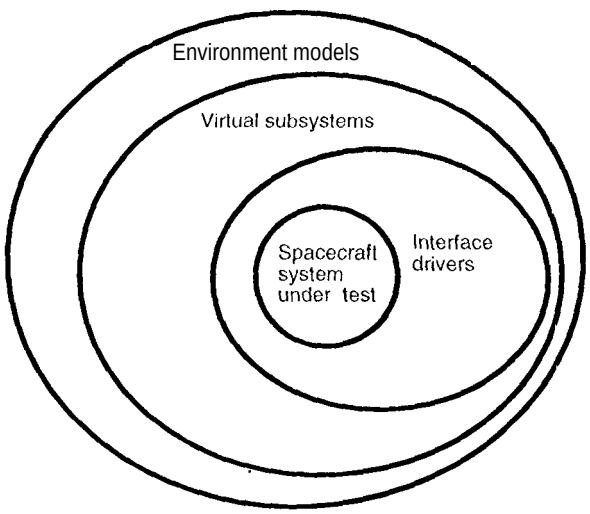

Figure 1. The layered model.

The outermost layers are virtual subsystems and environment models being used to fool the core system. For example, during spacecraft system test, the spacecraft at the core sends thruster commands to turn ("torque") itself, and reads sensors to determine the current attitude for comparison to its desired attitude. It then issues another set of thruster commands to comet remaining error. Thruster commands are "executed" by the dynamics model and sensor outputs arc produced by the dynamics mode]. Because of the inner layers insulat ing the core of this architecture, the flight software cannot

tell that it is in a test environment and not on interplanetary cruise.

Software development within this architecture is supported by an asynchronous messaging system developed at JPL. This system, Tramel (Task Remote Asynchronous Message Exchange Layer), is based cm tbc techniques underlying Remote objects Message Exchange [ROME]. Tramel provides application software written in C with a simple and highly portable, platform-independent abstraction for data communication among UNIX processes, VxWorks tasks, and Posix threads, In effect, each Tramel-literate process/task/tllrcad (called a "zone") attaches itself to an abstract network of processes (an "application universe") that insulates the application from details of the actual commmunication network such as processor architecture, operating system, and communication protocol. In addition, Tramel implements a publish/subscribe communication model that further shields application code from having to understand the configuration or state of the distributed application at any time.

All, and only, those zones within a given Tramel universe can use Tramel to exchange messages, and no zone can be in two universes al the same time. To guarantee that no dependencies on virtual subsystems and environment models (which would compromise fidelity) are built into core software, and vice versa, we partition the FST into two application universes. The core software elements inhabit a "flight software universe" and use Tramel only to exchange data among themselves. The victual subsystems and environment models inhabit a separate "support equipment universe". That is, we build a firewall between the core and the outer layers of the FST architectural bull's eye by mapping them into different Tramel universes and committing 10 use Tramel for interprocess communication. All communication between the core and the outer layers uses non-Tramel techniques implemented in the interface drivers layer of the architecture.

Organization of the Paper

This paper is organized as follows: the next section describes the architecture of the integration and test environment. Within that section are subsections on the primary subsystems, The penultimate section is a discussion of the software support for the architecture previously described. The paper concludes with a summary of lessons learned and topics for future work.

System Simulation Architecture

The spacecraft avionics arc surrounded by simulation support equipment, ground data system software, and consoles. Standard network interfaces and bases, commercial real-time operating systems and widely-used languages (C, C++, and 1 abView) mean that tbc FST environment consists primarily of ofl-the-shelf commercial products or JPI ,-developed software using industry - standard techniques.

To illustrate the plug-and-play nature o{ tbc architecture, the initial communication medium was ethernet. This was replaced with a MIl ,- S'1'1)- 1553 bus without affecting spacecraft or simulation subsystems.

The emphasis on modularity and the low cost of nc[work-ready embedded processors bas resulted in a distributed, mLllti-processor FST. The avionics and support simulation processes themselves typically use multiple processors.

The FST builds on JPL's efforts since 1992 to build prototype spacecraft. A specific example is the Asteroid Comet Moon 1 Explorer (ACME) spacecraft studies performed in 1993. ACME is a small, rigid body spacecraft stabilized in three axis using six reaction cont ml system (RCS) thrusters. ACME differs from more complex spacecraft in that hardware redundancy is minimized, for example, attitude control, command processing, and data handing reside on a single processor. This is representative of the smaller spacecraft in JPI ,'s future.

The FST simulation environment provides software and test-equipment support for attitude control, command and data communication, power, telecommunication, instrument, and data recording subsystems. Realistic factional interfaces have been defined and implemented to allow subsystems to be replaced, This results in a testbed with low inter-subsystem coupling and high intra-subsystem cohesion. Subsystem simulat ions are replaced by breadboards, engineering models, and flight hardware as they become available.

At t it ude Cont rol Subsystem (ACS).

'1 he design of the at tit ude control subsystem was based on the Cassin i spacecraft's software object architecture. The resulting architecture is a collection of disjoint objects communicating via standard data paths [AB95]. This results in an extensible system in which changes resulting from the incorporation of new technology or growth in capabilities are localized in a few objects. For example, switching from one type of gyroscope to another impacts only the Gyro Manager module.

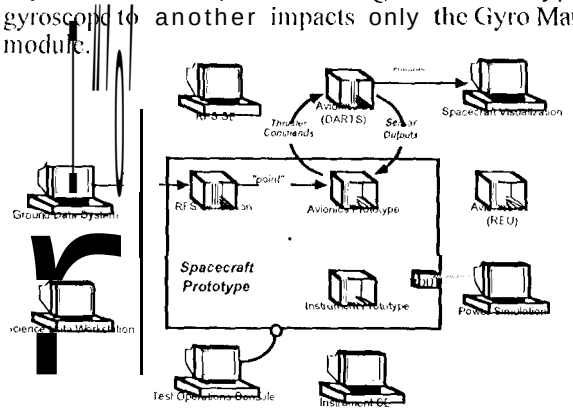

Figure 2. Attitude Control Subsystem Test

Figure 2 is an instantiation of the onion-layer architecture of the previous figure. In the center is the spacecraft prototype. The enclosing box represents the interface drivers. Simulation subsystems and test equipment are the outer layer of the diagram, The emboldened arcs represent comm an i cat ion triggered by a "point" command i nit iated from the ground data system.

Evolutionary bus implementations are being evaluated. A MIL-STD-1553 bus has been used as tbc ACS engineering bus and will be replaced by MIL-STD-1773 (fiber) without impacting the ACS software. The Controller Area Network (CAN) developed by the automotive industry, is being planned as an alternative communication medium. Insertion of these ncw media was facilitated by the connection-oriented interface drivers layer chosen to hide the details of tbc underlying network hardware.

The ACS provides core services such as "point and bold" and main engine burn. These services are tbc foundation for higher level maneuver sequences or data acquisition.

Command *and* Data Handling Subsystem (C&DH)

The C&DH subsystem consists of a telemetry management system and a command sequence manager for execut ion of sequences of commands stored on-board. These sequences can be either time or event-driven. Sequences can be parameterized, allowing the implementation of higher level spacecraft functions as macros. 'Ibis is particularly effective in a prototyping environment where mission scenarios can be accomplished by past ing together core services using scripts.

Dynamics Subsystem.

The spacecraft dynamics simulation is based on the Dynamics Algorithms for Real-Time Simulations [DARTS] software developed at JPL. DARTS provides a library of sensor and actuator modules that emulate the physics of a given sensor or actuator (such as thruster rise times and gyroscope tick counts). The DARTS shell (Dshell) accepts thruster commands as input, calls the appropriate actuator modules to produce a resultant force vector, calls DARTS to advance the collations of motion by one time step, then calls tbc appropriate sensor models to produce sensor readings based cm the ncw orientation of the spacecraft.

in addition to physical models of sensors and actuators, a hardware simulation models the electronic interfaces to specific sensors and actuators, e.g. valve drive electronics, inertial reference units or a MIL-STD- 1553 remote terminal. These hardware simulations make it possible to substitute various bus implementat ions with little or no impact on the rest of the system. For example, TCP/IP Sockets over Ethernet can be transparently substituted for bus hardware during development.

instrument Simulation.

It is common that after the power, data, and command interfaces to a scientific instrument are determined, development of the instrument proceeds in parallel with spacecraft development. To decouple tbc development of the instrument and spacecraft, tbc FST provides a two part instrument simulation. Spacecraft simulation and test can proceed without the instrument present, Figure 3 is tbc onion-layer model w itb emphasis on communication between subsystems during test of a simulated or prototype instrument.

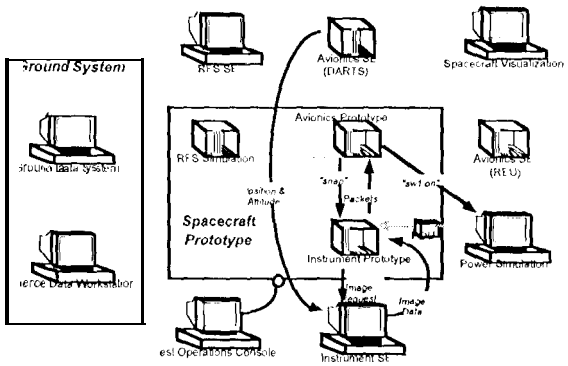

Figure 3. Instrument Simulation

The two parts of the simulation arc an instrument simulation and a simulated environment. These lwo parts are elaborated in Figure 4. The instrument, possibly simulated in software, communicates through the same command-data interface and produces the same power load as tbc physical instrument will. in addition, tbc simulated environment takes observational parameters output from tbc instrument and creates plausible data (images, in some cases) delivered back to tbe inst rument subsystem. The simulated environment can be simple or complex, the latter incorporating position and attitude data from the DARTS simulation, rendered images, background stars derived from star catalogs, typical scene blur provided by an optics model, and simulated detector noise.

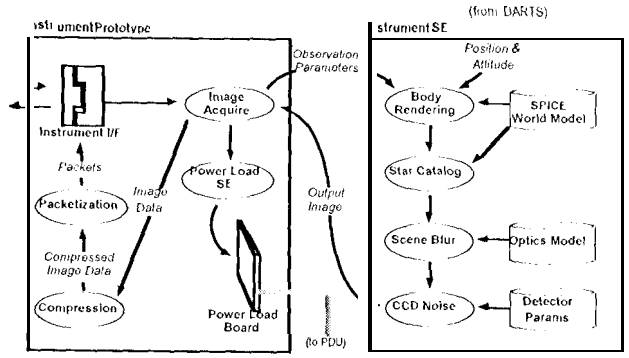

Figure 4. Instrument simulation and environment.

```
on 1 )(). TURN_0
  {sacs point 0.28 0.28 0.69 -0.59 -announce
DO_SNAP_ 0}
on 1)0. SNAP_ 0
  {camera snap -mosaicid 3 3 0 0 announce
DO_TURN_1}
on 1) C). TURN. 1
  (sacs point 0.20 0.19 0.72 -0.62 -announce
DO_SNAP_1}
01, DO_SNAP_1
  {camerasnap-mosaicid 3 3 0 1 announce
DO_TURN_2}

on DO_TURN_9 "announce MOSAIC COMPLETE"
announce DO_TURN_0
```

Figure 5. Command sequence for 3x3 Image Mosaic

Software Support for the FST Environment

The FST software development environment provides several mechanisms that support incremental development, smooth integration, and extensibility, The common foundation of these mechanisms is Tramel, the media-independent message passing system described briefly above. Tramel enables subsystem modules to produce output by publishing messages without knowing what other modules will receive them, and to consume input by subscribing to messages without knowing what other modules will produce them. Each message has a subject, which is an application-selected integer that functions somewhat like a method selector in an object-oriented programming language such as Smalltalk, and may also optionally have content, an arbitrarily long array of bytes. A task joins an application universe by registering (basically, declaring some ASCII string to be its name) and after having registered may subscribe to any number of message subjects; different message handlers (callback functions) may be declared for each subject. A task publishes a message by specifying to Tramel its subject, content, and content length; Tramel handles delivery of the message to every subscriber, whether on the same processor or on other processors, using sockets, messages queues, pipes (FIFOs), or whatever other communication channels are available; the publishing task is never aware of the location of the recipient(s) or the transport mechanism(s) used. In this way, the implementation of one module is wholly decoupled from that of any other. The Appendix contains a sample C program that uses Tramel to publish an alarm message every sixty seconds.

One helpful extension of Tramel is Tcl_tramel, a Tcl [Ous94] application programming interface to Tramel functionality. This library provides Tcl commands that record subscriptions and unsubscriptions and publish Tramel messages. Subscribing to a given subject from within a Tcl script automatically links that subject to a callback function that passes the content of each message to a Tcl interpreter. This enables applications written in Tcl can participate fully in a Tramel application universe.

"fstshell", which is built on Tcl_tramel, is another useful mechanism for encapsulating modules. By linking With fstshell and invoking its fststart() function, FST application code automatically acquires the ability to interact with other modules in the same application universe and also to be commandable via Tcl. Figure 5 contains an example of a high-level fstshell command sequence. Subsystems can exchange commands by publishing messages containing Tcl commands and subscribing to the commands published by other subsystems. This makes integration of new, higher level functionality simple.

For example, an optical pointing module can be added simply by having it subscribe to image messages from the camera and publish "point" commands. Encapsulating subsystems behind fstshell facilitates distribution of functionality over processors -- an instrument pointing module can be transparently moved to another processor.

[conclusion

The FST employs several mechanisms which facilitate Spacecraft subsystem integration and test and also provides an environment for demonstration of new technology in an end-to-cad system context. The FST provides core services under a layer of higher level functions that enable integration of new technologies.

These technologies might include new hardware as well as more abstract functionality implemented in software. Hardware such as instrument interfaces, data compression engines, or ACS devices can be quickly integrated using commercial hardware while breadboards are being implemented in programmable gate arrays. Complex software entities, such as rok-based logic engines, can be layered quickly behind the reusable subsystem interfaces and demonstrated interacting with other software components.

Future work includes formal specification of the Tramel messaging functions. Reverse engineering of formal specifications from existing spacecraft code has been demonstrated [CA93], and distributed asynchronous messaging schemes similar to Tramel have been formally specified [AK86].

The techniques described in this paper were developed

by an eight person team in about six months, The resulting architecture is used as demonstration of spacecraft technology in a flight-like environment. Experience with extending the architecture and using it for integration of new technology support the conclusion that these techniques are robust and suited for incremental development of large, complex, distributed real-time systems.

Acknowledgments

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Dr. Auernheimer's work was supported through a NASA/ASEE Summer Faculty Research Fellowship and a sabbatical provided by California State University, Fresno.

Bibliography

[AB95] C. K. Ames and H. C. Briggs. Experimental attitude control simulation architecture for system development and integration. Paper presented at AAS'95 (American Astronautical Society). Keystone Colorado, February 1995.

[AK86] B. Auernheimer and R. A. Kemmerer. RT-ASLAN: a specification language for real-time systems. IEEE Transactions on Software Engineering, September 1986. Also reprinted in the IEEE tutorial Hard Real-lime Systems, 1988.

[Bur93] S. Burleigh. ROME: Distributed [;-t+ object systems. IEEE Parallel and Distributed Technology Systems and Applications, May 1993. Pp 21-32.

[CA93] B. Cheng and B. Auernheimer. Applying formal methods and object- oriented design to existing flight software. Proceedings of the NASA Goddard Software Engineering Workshop, December 1993.

[CT94] E. K. Casani and N. W. Thomas. The J})], night System Testbed. Proceedings of the Spring 1994 SPACE Conference. Orlando, 1994.

[DARTS] http://robotics.jpl.nasa.gov/tasks/dshell/

[FST] http://fst.jpl.nasa.gov/

[Ous94] J. K. Ousterhout. Tcl and the Tk Toolkit. Acidisoa-Wesley, 1994.

[Tan87] A. S. Tanenbaum. Operating systems: Design and implementation. Prentice-Hall, 1987.

[Tramel] file://tsunami.jpl.nasa.gov/home/scott/tramel/man/tramel.3

Dear Author,

Congratulations on the acceptance of your paper for publication in the proceedings of the Workshop on Parallel and Distributed Real-Time Systems, being held April 15-16, 1996, in Honolulu, Hawaii. PLEASE READ THE FOLLOWING COMPLETELY BEFORE FORMATTING YOUR PAPER - THIS LETTER CONTAINS IMPORTANT INFORMATION!

Attached to this message is a set of electronic instructions for formatting your paper. If you have any formatting/publication questions, please feel free to contact me (numbers below). For registration, program, or other questions, please contact either Lonnie Welch at Ph: + Intl. 540-653-2193 or e-mail: welch@homer.njit.edu; or David Andrcws at Ph: + Intl. 501-575-5090 or e-mail: dla@engr.uark.edu.

o GUIDELINES/FORMATTING - These instructions have been updated. Please see the attachment: INSTRUCT.TXT. You should be able to format your paper successfully if you use the measurements in the attachment.

0 PAGE LIMIT - Short papers: **4** pages. Extra pages are allowed, at a cost of $100,00 US EACH. Please make your check payable to: WPDRTS 96 and include it with your paper. [Do NOT make it out to any other name. The conference "does not accept credit card payments and extra pages not paid for will not be published.] If you need a receipt, complete the receipt request form and include it with your **check and the original manuscript that** you send to me. See attachment: RECEIPT.TXT

o COPYRIGHT release - print out the form, fill it out COMPLETELY -- BE SURE TO INCLUDE THE TITL E OF YOUR PAPER, THE AUTHORS' NAMES, etc. -- Just signing it is NOT **enough! Send it in with** your paper -- we cannot publish your paper without it. See attachment: COPYRGHT.TXT

o MANUSCRIPT - I'[ease submit an **ORIGINAL** of your paper. We must have the best quality possible. We will accept PostScript copies of your paper with the provision that they are not encapsulated, that you send them to me with the subject referenced as WPDRTS paper, and that you send them in promptly. If you send a PostScript paper, you still **MUST*' send in your copyright form (fax it to me first, then send the original to my attention), plus your check for any extra pages. If you use LaTeX formatting macros, we have an updated set available and ask that you use it and not any older versions you may have. Please request it by sending "me an e-mail.

o ELECTRONIC ABSTRACT of your paper - Only E-mailed abstracts will be accepted. Send your abstract to: WPD96-abs@computer. org.