A Reusable, Real-Time Spacecraft Dynamics Simulator

Jeffrey J. Biesiadecki **David A. Henriquez** **Abhinandan Jain**
Jet Propulsion Laboratory/California Institute of Technology
4800 oak Grove Drive hi/S 198-235, Pasadena, CA 91109 USA

ABSTRACT

DARTS Shell (Dshell) is a multi-mission spacecraft simulator for development, test, and verification of flight software and hardware. Dshell is portable from desktop workstations to real-time, hardware-in-the-loop simulation environments.

Dshell integrates the DARTS S/C flexible multi-body dynamics computational engine with libraries of hardware models (for actuators, sensors, motors and encoders) into an integrated simulation environment that can be easily configured and interfaced with flight software and hardware for various real-time and 11071 real-time S/C simulation needs.

Dshell is in use by several of NASA's inter-planetary deep space missions including Galileo, Cassini, Mars Pathfinder, and several projects in JPL 's Flight System Testbed.

1 INTRODUCTION

DSHELL is a high fidelity, multi-mission spacecraft dynamics simulation package. The main goals of the DSHELL environment are: to significantly reduce the software development required to interface dynamics simulators, hardware models and hardware-in-the-loop devices; to eliminate the need for separate interface development efforts across the various (analysis, software and real-time) testbeds within a project, and allow easy migration of models between testbeds; to allow the easy support of a variety of S/C configurations and models and simulation environments for all the phases of the mission; allow the easy reuse and customization of hardware models across various missions.

The core of DSHELL is DARTS, a generic computation engine for flexible multi-body dynamics. However, spacecraft dynamics are affected and determined by specific classes of real-time hardware devices. These dynamics-dependent models can be grouped, as they are in DSHELL, into actuator, sensor, motor and encoder models (see Sec. 2.2 for the definition of these classes). Actuators and motors being those devices that affect the dynamics, and sensors and encoders which are affected by the dynamics. Hardware models for devices such as gyroscopes, thrusters, and star-scanners are organized in libraries, which can be created or augmented by the user. Models have standardized interfaces to DARTS, the external simulation environment, and the user. The plug and play simulation can be easily configured and interfaced to flight software for algorithm development, as well as for test and integration. The object-oriented model library includes extensive instrumentation for giving a user the high visibility into the simulation necessary for effective use as a design, development and test tool. The design of DSHELL makes spacecraft simulation a simple task of assembling the desired ensemble of hardware models, with some notion of the spacecraft's inertia and flexibility.

DSHELL and its models are classified as real-time because they complete their execution within a tick of simulation time. This deterministic performance is required for synchronizing the DSHELL models with other simulation models outside of the DSHELL environment. These other simulation models may or may not meet real-time performance criteria. Models are implemented as non-real-time due to the nature of the device they simulate, or to ensure that critical real-time performance requirements of the simulator are met. Non-real-time models respond to events or commands that do not necessarily complete within a tick boundary. Figure 2 shows an example of the kinds of models that can be part of a realistic spacecraft simulation; it also shows how non-real-time models can affect the behavior of real-time DSHELL models. For example with optical navigation, the camera instrument is used by AACS flight software to determine the pointing precision of the spacecraft, which in turn affects how often the DSHELL thrusters are fired fired. The camera image must be synchronized with the spacecraft dynamics, but the simulation should not have to wait to receive the camera image. It is therefore desirable to have event-driven non-real time models run

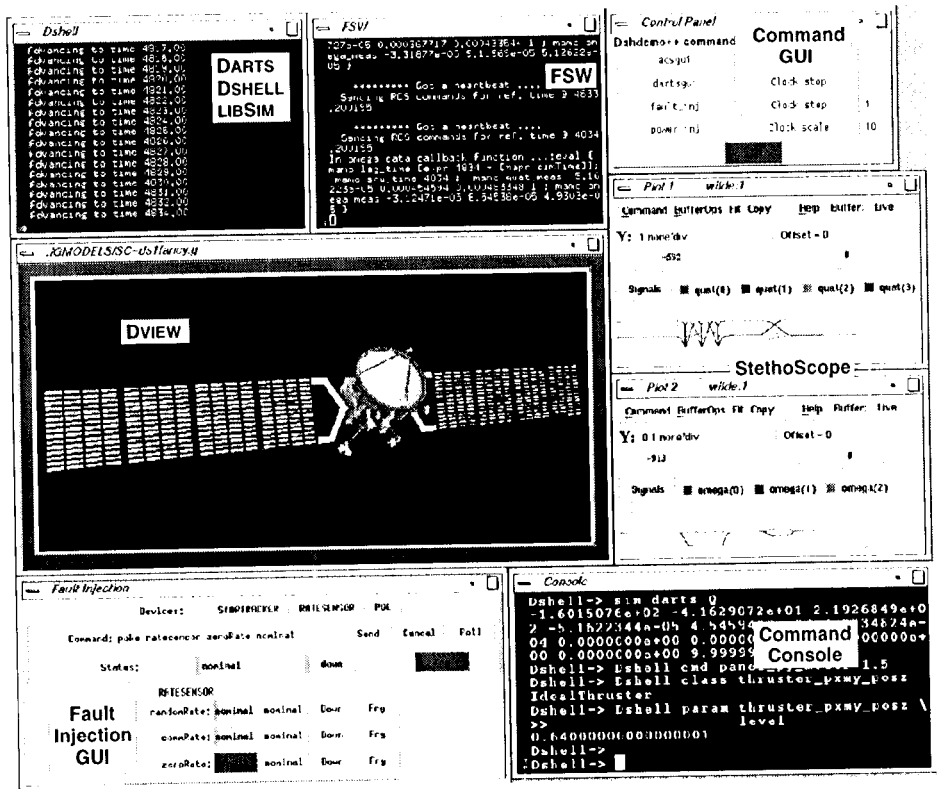

Figure 1: Example spacecraft simulation

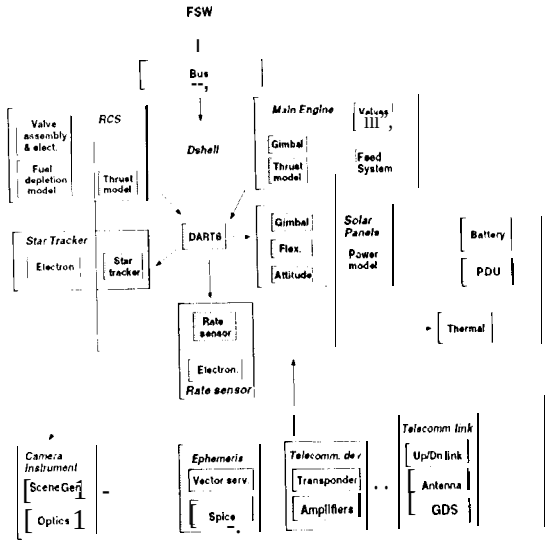

Figure 2: Representative types of models in a spacecraft simulation

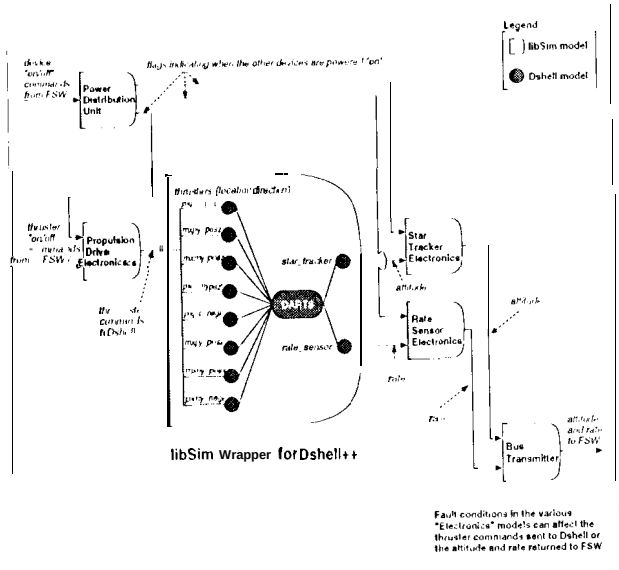

Figure 3: LIBSIM wrap of DSHELL in a spacecraft simulation

communicate with DSHELL real-time models in order to simulate a complete spacecraft. To that end, LIBSIM was developed.

LIBSIM is a library which uses a data flow paradigm for connecting higher-level device and subsystem models, and provides special features for modeling faults. Examples of LIBSIM models include bus interfaces, device electronics, and valves. These properties make 1111S1 M inherently the highest software layer of the simulator. Event-driven hardware device models can be written with L] BSIM, and event-driven processes can be queried via a LIBSI M wrapper model. A LI []-S IM wrapper model would send and receive messages to and from the non-real-tilnc processes, to incorporate the data from these processes into the real-time core. And it is through a LIBSIM wrapper model that D SHELL (figure 3) can have connectivity with a higher level simulator. (See reference [1] for more information on LIBSIM M.)

D SHELL, I) ARTS and LIBSIM are all part of an integrated software package called the Autonomy Testbed Environment (ATBE) toolkit. This toolkit, also crones with a 31) animation tool called D VIEW which renders the motion of a spacecraft, as computed by D SHELL. Figure I illustrates a simple spacecraft simulator which was written using the ATBE toolkit. This simple spacecraft in the spacecraft simulator is being autonomously controlled by A ACS flight software, with the spacecraft state variables being plotted in real-time. A LIBSIM fault injection GUI allows faults to be injected into the spacecraft simulator to test the response of flight software faults. (See reference [2] and [1] for a more detailed overview of ATBE.) But the core package in the ATBE toolkit is still DSHELL, without which realistic spacecraft simulation could not be built.

2 DSHELL DYNAMICS SIMULATOR

1 DSHELL is a multi-mission spacecraft simulator for development, test and verification of flight software and dynamics-dep endent hardware. l) S1[llll, is portable from desktop workstations to real-time, hardware-in-the-loop simulation environments. I DSHELL integrates the I DARTS flexible multi-body dynamics computational engine and libraries of hardware models (for actuators, sensors and motors) into a simulation environment that can be easily configured and interfaced with flight, software and hardware for various real-time and non real-tilnc spacecraft, simulation needs.

The main goals of the D SHELL environment are: to Significantly reduce the software development required to interface dynamics simulators, hardware models and hard ware-i n-the-loop devices; to eliminate the need for separate interface development efforts across the various testbeds (analysis, software and real-time) within a project, and allow easy migration Of models between testbeds; to allow the easy support of a variety of S/C configurat ions and models and simulation environments for all the phases of the mission; and to permit the easy reuse and customization of hardware models across various missions.

D SHELL is a library implemented in C+ + may be embedded in another simulator as described in section 1. Or, a small "main()" rout ine can be written to send data between flight software and D SHELL models, and advance simulat ion time. For model development, a generic "open-loop" version of main() is available in which the user controls tire! and data to and from models. This is invaluable for writing batch scripts to do regression testing.

Simulation time is tracked by DSHELL from the start of simulation. Each tick of simulation time is an 1/0 **step for** DSHELL. Inputs and outputs to and from D SHELL models are expected to occur within that tick. Each 1/0 **step** consists of an integer number of **integration steps.** And for each integration step, D ARTS computes the multi-body dynamics.

2.1 DARTS Dynamics Algorithms for Real-Time Simulation

The D ARTS dynamics compute engine [3] implements a fast and efficient spatial algebra recursive algorithm [4, 5] for solving the dynamics of flexible, multi-body, t ree-topology systems. It is very general, and is also in use for non-spacecraft applications such as molecular dynamics [6]. DARTS is a library implemented in ANSI C available for Unix and VxWorks platforms.

An analyst provides an input file that is read at run time and specifies the **bodies** that make up the spacecraft: their masses, inertial and flexibility properties, as well as t he types of **hinges** that bind t hem together. A hinge connects two bodies, and there are many types available (such as pin, [J-joint, gimbal, translational, and others). (When D ARTS is used in conjunct ion wit h D SHELL, all the DARTS information can be placed in the DSHELL input file.) Bodies may be connected in a tree topology, wit h each body having a single parent body, and the root of the tree being referred to as

the **base body.** The locations of named **nodes** where forces may be applied or dynamics properties should 1 ɔe computed arc also specified in a DARTS input file. Because the above data is not hard-coded, dynamics models can be easily constructed for different missions, and models can be changed without, necessitating the recompilation of source code.

2.2 DSHELL Model Classes

I)s11 ELL provides C++ base classes for hardware device models. **Actuators** can impart a force on a node of a body, such as a thruster. Sensors are attached to a node of a body and make use of dynamics calculations produced by DARTS for that node. Examples of sensor models include star trackers and gyroscopes. **Motors** are attached to hinges and are used to articulate! the bodies that the hinge connects. **Encoders** are also attached to hinges, and are to motors what sensors are to actuators. DSH ELL device models are massless, and other than applying a force or articulating a body, do not affect the dynamics of the spacecraft. All four of these classes are derived from a common base class **(Model),** which defines data and methods associated with each model.

Dat a for DSHELL models consists of parameters, discrete states, continuous states, commands, and outputs. **Parameters** are **values** that are **set** while reading the DSHELL input file upon startup, but are not changeable by the model itself. **Discrete states** are initialized at startup, and may be modified by both the model and the user during run time. **Continuous states** are updated by the numerical integrator in I DARTS, and require the model builder to provide a method for computing the derivatives of these states. **Commands** are time tagged data structures sent by flight software, and **outputs** are time tagged data structures sent to flight software. Parameters, discrete states, commands and outputs may be of any basic C data type (such as *int* or *double*), C enumeration, structure, or fixed-size array. Structures may be nested, may contain arrays, arrays of structures are permitted, and so on. Continuous states arc either *dou ble* or arrays of *double.*

~'here are various methods available for a D SHELL model to define its behavior. **Pre-** and **post- I/O step** methods ate called at the beginning and end of an I/O step, and are typically used for models to retrieve commands from and send data to flight soft ware, respectively. **Pre-** and **post-integration step** methods are called at the beginning and end of an integrat ion step, and are typically used to compute discrete states. Each integration step, an integrator calls a function to compute the time! derivative of the DARTS state vector. This function also calls **pre-** and **post-derivative** methods for each D SHELL model immediately before and after computation of D ARTS derivatives. The pre-derivative method is typically used for actuators to apply forces to the nodes they are attached to. The post-derivative method is typically used to comput e the time derivative of any continuous stat es t he model may have. The number of times these derivative methods are actually called per integration step depends on the numerical integration algorithm selected. Note that unlike LIBSIM, DSHELL models do not interact with each other directly, so the relative order in which their methods are executed docs not matter (figure 4).

The base classes provide several methods useful to a model, including methods to get the simulation time, step sires, and I) AlVI's information. These would be called from the model's pre/post I/O step and other methods described in the previous paragraph.

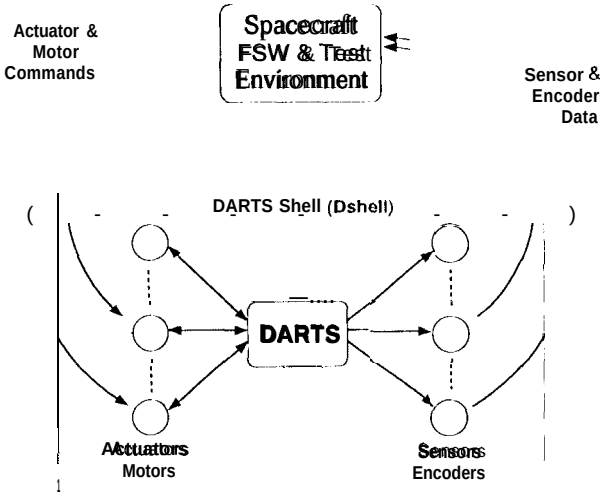

Figure **4:** Typical data flow for a D SHELL simulation

2.3 DSHELL Model Libraries

Classes for actual device models ate derived ft om any of the four base classes described in section **2.2.** The code for model classes may be grouped into reusable libi aries, organized perhaps by mission, by vendor, or by type of device. There are several models available for tilt'ustcls, gyroscopes, star scanners, accelerometers, and other devices used on JPL spacecraft. They can be used as-is for quick prototype simulations, or as a starting point for similar models on a new spacecraft.

Because all DSHELL models use the same set of predefined methods, an automatic code generator is available to simplify model development. The model developer writes a text file that describes the model, listing the types, names, and descriptions of the parameters, states, commands, and outputs associated with the model. A prototype graphical user interface is available for generating this file. The code generator takes this file as input, and generates a C++ header file and stub source file for the model class. The developer then fills in methods (pre/post I/O step and the rest) as needed to define the model's behavior. Very little knowledge of C++ is needed, but it is useful to be familiar with C.

The automatic code generator also makes an **interface class,** specific **to** the model class the developer is defining (figure 6). The developer never changes this code and does not need to even look at it. This class provides model-specific functions to issue commands and retrieve outputs from a model, code commonly needed to define a text interface to the model's data, and other methods needed by DSHELL. The command and output functions would typically be called from the simulator or main() routine that calls other DSHELL routines. They are model-specific to keep them tylm-safe (avoiding the use of **void *** pointers reduces the occurrence of some programming errors). This also allows a simpler interface for commands and outputs of basic types, and is faster than performing any kind of marshalling or conversion of structures. The code generated for the interface class is meant to eliminate tedious coding by a developer that is typically needed for a model. It is generated in a class separate from the **actual** model class to clearly delineate code the developer should modify. This helps keep the code for the stub model class small.

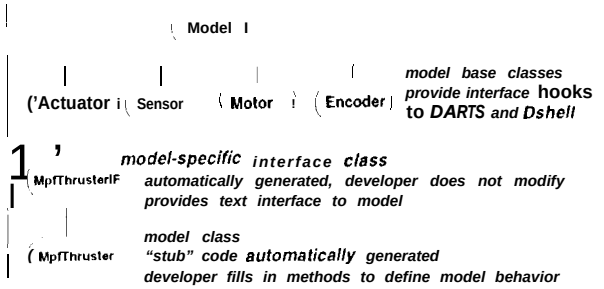

Figure 6: DSHELL class hierarchy

2.4 DSHELL Run Time Environment

The input file containing DARTS information may also contain statements to instantiate models, specifying the model class and instance name. States and parameters for a model may be initialized here as well. Again, not had-coding this information makes it easier to change configurations without recompiling code.

Like LIBSIM, DSHELL also has an extensive set of *Tcl* commands which can be used to get information about the simulation and models therein. In particular, the values of model states and parameters can be peeked and poked from the command line, commands to models can be issued as if they came from flight software, and outputs from models can be examined. There are enough commands available to query which models are instantiated and the data types and descriptions of model states that a graphical user interface to display state data can dynamically create itself, so a programmer does not need to change GUI code if the simulation configuration changes or new models are added. A prototype of such a GUI has been implemented using *Tk*.

All DARTS and DSHELL model state variables can be **checkpointed** to a text file containing "poke" commands. This file can be edited by the user if necessary without needing to know any syntax other than the already familiar *Tcl* commands. On a subsequent run, this file can be used to initialize states and **resume** a previous run.

DSHELL can also keep track of multiple S/C dynamics models. Alternate dynamics models of the same spacecraft can be selected from (such as in-cruise versus in-orbit configs with different fuel slosh behavior, or pre- versus post-probe release). Only one such alternate dynamics model may be active at any given time, and DSHELL device models implicitly interface only to the active model. Or, multiple spacecraft can be bookkept, as in the hTcw Millennium Program's Deep Space Flight 3 formation flying mission. Any combination of alternate models for multiple spacecraft is allowed.

As with LIBSIM and DARTS models, DSHELL models can be deactivated from the *Tcl* command line or startup file. This is useful for debugging, or if there are alternate models for the same spacecraft device (perhaps one would interface to actual hardware-in-the-loop).

It is also possible to schedule C functions and *Tcl*

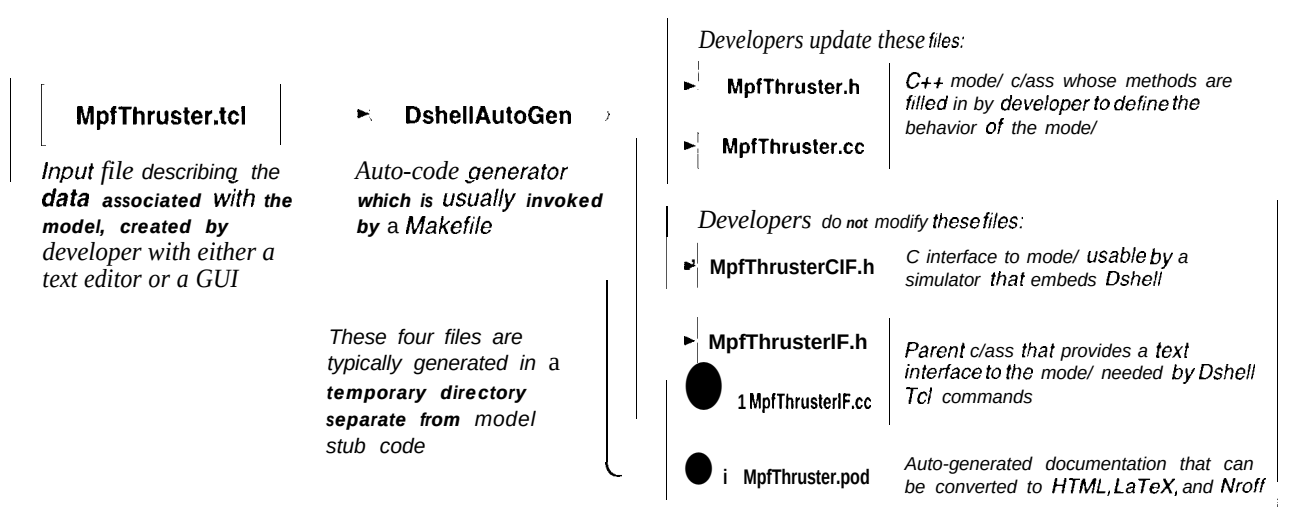

Figure 5: Input and Output Files for the DSHELL Automatic Code Generator

scripts at run time for either one-time or repeated execution. This is very handy for debugging and monitoring variables. It is also useful for interfacing DSHELL to other tools. Such interfaces have been created to Real-Time Innovation, Inc.'s data monitoring tool *StethoScope* and to JPL's 3D viewer *Dview*. Interfaces to other tools can be created in a similar manner, without having to change DSHELL code. Aside from keeping DSHELL code smaller and cleaner, it makes it easy to mix and match interfaces among testbeds which use different monitoring tools.

3 APPLICATIONS

DSHELL was recently used by the JPL's Cassini Project. The project wished to update the dynamics computation core of its **Flight Software Development System (FSDS)**. **FSDS** was developed to simulate the ACS of the Cassini spacecraft in order to provide a testbed for Cassini flight software development. The Cassini Project developed **FSDS** using DARTS to compute mulit-body dynamics, prior to the existence of DSHELL and the ATBE toolkit,. As a result, DARTS was "hardwired" into their simulator without an interface to it.

With minimal intrusion into the source code, the DSHELL was successfully integrated into FSDS. The addition of DSHELL provided a seamless Tcl interface to peek and poke its DARTS state variables and parameters, and created a seamless interface to the four alternate spacecraft dynamics models for **FSDS**. The DSHELL interface also provided visibility into the DSHELL slid DARTS data for external monitoring tools. One such monitoring tool is *StethoScope*, developed by Real-Time Innovations, Inc. With *Stethoscope*, the DSHELL and DARTS variables can be monitored in real-time. And also in real-time, DVIEW could display the dynamic behavior of the spacecraft. The mal-time graphical displays of the dynamic state of the spacecraft allow an analyst to analyze and debug during the simulation, and not just post-simulation. The ability to peek and poke the DSHELL variables allows the analyst to change the dynamics state to test the response of the flight software to different dynamic states without restarting the simulation.

After the DSHELL toolkit was successfully integrated into FSDS, FSDS was redubbed **DSHELL High Speed Simulator** (DHSS). Currently, the various device models in DHSS are lining converted into DSHELL and LIBSIM models to also provide pcx!k/poke, monitoring and checkpointing capabilities to DHSS.

This work demonstrates the versatility of DSHELL. It can be used not only to build new simulators, but also to add more capabilities to existing simulators and testbeds which currently usc! DARTS for dynamics computation.

4 CONCLUSION

A reuseable, real-time spacecraft simulator is essential for the design, development, testing and integration of autonomy flight software and hardware. DSHELL was made for just such a need. Since unique spacecrafts

are defined in an input file for DSHELL, there is no architectural limitation to the reuse of DSHELL. And its real-time performance and design allows real hardware to be swapped with any DSHELL hardware model, and vice versa. This characteristic removes domain boundaries for the types of testbeds with which DSHELL can be used. DSHELL can migrate from a poor fidelity, pure software simulation, to a high fidelity, highbred hardware and software simulation.

Currently, DSHELL and the ATBE toolkit are being used in the development of Cassini High Speed Simulator, which is nearing completion. The High Speed Simulator will be used during Cassini mission operations to test command sequences prior to uplink.

For more information on DSHELL, visit the DSHELL web site: http://dshell.jpl.nasa.gov

5 ACKNOWLEDGEMENTS

The authors would like to express their thanks for the work performed by the other ATBE team members: Sally Chou, James Fu, Gani Ganapathi, Chester Joe, Patti Koenig and Ling Su.

The mead described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

REFERENCES

[1] J. Biesiadecki, A. Jain, and M. James, "A Reconfigurable Testbed Environment for Spacecraft Autonomy," in *i-SAIRAS'97*, (Tokyo, Japan), July 1997.

[2] J. Biesiadecki and A. Jain, "A Reconfigurable Testbed Environment for Spacecraft Autonomy," in *Simulators for European Space Programmes, 4th Workshop*, (Noordwijk, The Netherlands), ESTEC, Oct. 1996.

[3] A. Jain and G. Man, "Real Time Simulation of the Cassini Spacecraft Using DARTS: Functional Capabilities and the Spatial Algebra Algorithm," in *5th Annual Conference on Aerospace Computational Control*, Aug. 1992.

[4] G. Rodriguez, K. Kreutz-Delgado, and A. Jain, "A Spatial Operator Algebra for Manipulator Modeling and Control," *The International Journal of Robotics Research, vol. 10,* pp. 371381, Aug. 1991.

[5] A. Jain, "Unified Formulation of Dynamics for Serial Rigid Multibody Systems," *Journal of Guidance, Control and Dynamics*, vol. 14, pp. 531-542, May-June 1991.

[6] A. Jain, N. Vaidehi, and G. Rodriguez, "A Fast Recursive Algorithm for Molecular Dynamics Simulations," *Journal of Computational Physics*, vol. 10G, pp. 258 268, June 1993.