S11-61

N96-12924

P-9

A DYNAMIC SATELLITE SIMULATION TESTBED BASED ON CLIPS AND CLIPS-DERIVED TOOLS

Thomas P. Gathmann
Rockwell International
Satellite & Space Electronics Division
Seal Beach, CA

Problem

Current day spacecraft are complex machines and those on the drawing boards are increasingly more sophisticated and broader in scope. Gone are the days when a single engineer could fully grasp the intricacies of an entire satellite. Note that the recently launched Galileo spacecraft has several processors on-board the vehicle [1]. This fact, coupled with the increasing power of computing hardware and software tools and techniques, has introduced the possibility of realistic simulations being used for product definition, design, manufacture, and, even, performance analysis. The Strategic Defense Initiative Office (SDIO) is convinced of simulation capabilities since it has funded the National Test Bed (NTB) facility to evaluate the performance of all facets of the "Star Wars" concept.

Due to heated competition for the development and delivery of satellites, there is an increased reliance on simulation of components, subsystems, systems, and entire constellations of spacecraft. Given the wide variety of configurations and purposes of these satellites, flexible and convenient means for generating study and engineering data are necessary [2-4]. Monolithic simulations have become unwieldy and expensive to maintain. Configurable tools that can be quickly and accurately constructed are required. Rapid prototyping techniques have become more accepted within the aerospace industry for the production of deliverable software and also as a means to manage the software process [5].

We were motivated to define and build a sophisticated satellite simulation capability for the evaluation of a satellite operations automated environment called IntelliSTAR™ [6,7]. This architecture, and associated prototype, addresses the entire spacecraft operations cycle including planning, scheduling, task execution, and analysis. It is aimed at increasing the autonomous capability of current and future spacecraft. It utilizes advanced software techniques to address incomplete and conflicting data for making decisions. It also encompasses critical response time requirements, complex relationships among multiple systems, and dynamically changing objectives. Given the extreme scope of activities that are targeted, a sophisticated, flexible, and dynamic simulation environment was required to drive this prototype. In particular the derived requirements for evaluating the IntelliSTAR™ prototype include:

- provide realistic and dynamic environment
- easily reconfigurable
- multiple levels of fidelity

The overriding need of IntelliSTAR™ was a means for providing a valid evaluation of the concept (see Figure 1). This evaluation was planned to be accomplished through the injection of various scenarios describing mission and behavior types for the spacecraft to be controlled. Given this stimulus, the IntelliSTAR™ prototype provides measures of the plan and its status to satisfy the objectives for the satellite mission.

Approach

The testbed approach to simulation has risen to the top of the list of options due to the following attributes:

- flexibility to easily configure based on unique customer requirements
- modularity of the simulation components to allow the testing of portions of the overall system or varying degrees of fidelity for portions within the same simulation
- interoperability through the use of consistent user and integrator interfaces for reduced training.

Side benefits include the centralized storage and accumulation of metrics and related information of the simulation capabilities and past usage of the testbed.

Our approach to the development, utilization, and maintenance of a sophisticated satellite simulation testbed is the use of rapid prototyping and knowledge-based techniques coordinated with the use of existing simulation and communication resources. An architecture has been defined that provides the following attributes for a spacecraft simulation that addresses autonomy, surveillance, and survivability capabilities (see Figure 2):

- integrating architecture that supports the expansion of capabilities and resources
- high-level user interface for speci-

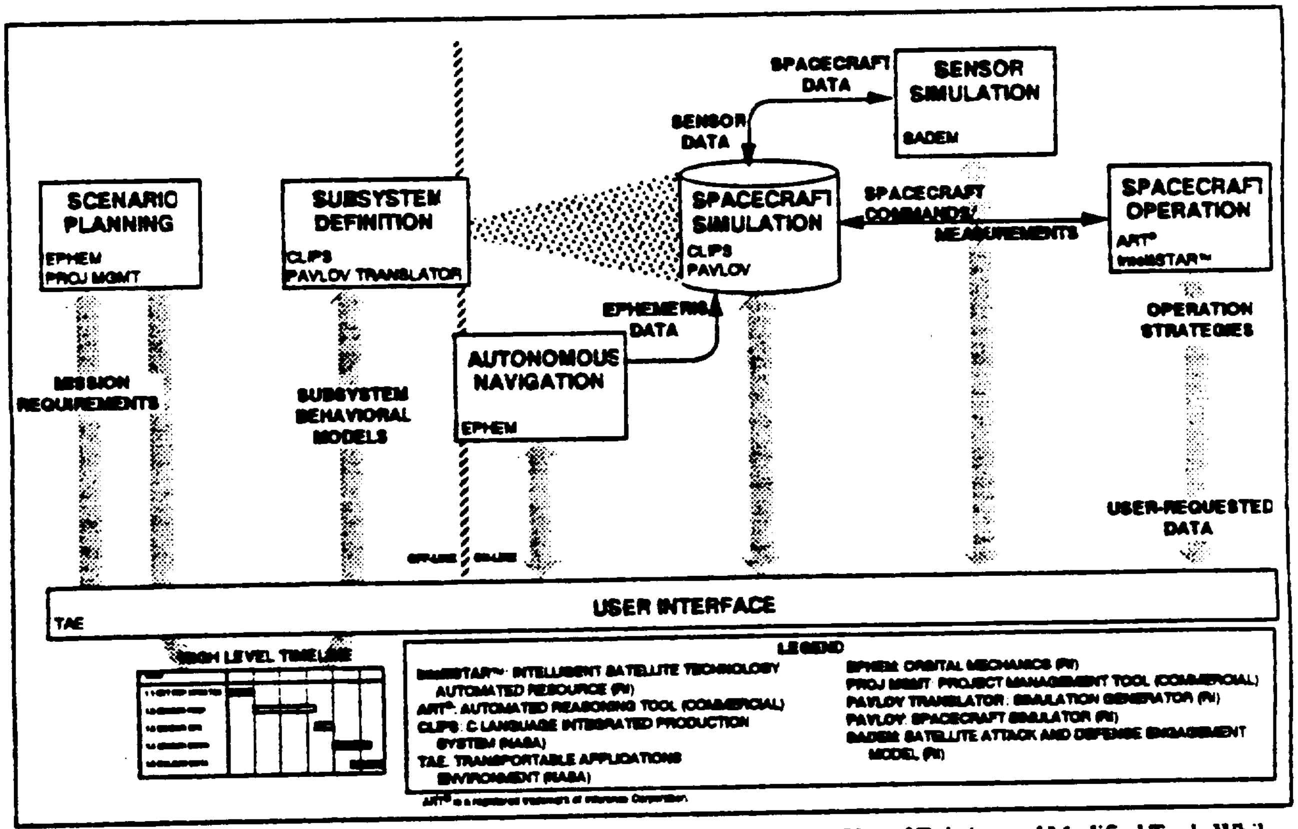

SPACECRAFT DATA
SENSOR SIMULATION
SADEM
SENSOR DATA
SCENARIO PLANNING
EPHEM
PROJ MGMT
SUBSYSTEM DEFINITION
CLIPS
PAVLOV TRANSLATOR
SPACECRAFT SIMULATION
CLIPS
PAVLOV
SPACECRAFT COMMANDS/ MEASUREMENTS
SPACECRAFT OPERATION
ART®
IntelliSTAR™
EPHEMERIS DATA
AUTONOMOUS NAVIGATION
EPHEM
MISSION REQUIREMENTS
SUBSYSTEM BEHAVIORAL MODELS
OPERATION STRATEGIES
USER-REQUESTED DATA
USER INTERFACE
TAE
HIGH LEVEL TIMELINE
LEGEND
IntelliSTAR™: INTELLIGENT SATELLITE TECHNOLOGY AUTOMATED RESOURCE (RI)
ART®: AUTOMATED REASONING TOOL (COMMERCIAL)
CLIPS: C LANGUAGE INTEGRATED PRODUCTION SYSTEM (NASA)
TAE: TRANSPORTABLE APPLICATIONS ENVIRONMENT (NASA)
EPHEM: ORBITAL MECHANICS (RI)
PROJ MGMT: PROJECT MANAGEMENT TOOL (COMMERCIAL)
PAVLOV TRANSLATOR: SIMULATION GENERATOR (RI)
PAVLOV: SPACECRAFT SIMULATOR (RI)
SADEM: SATELLITE ATTACK AND DEFENSE ENGAGEMENT MODEL (RI)
ART® is a registered trademark of Inference Corporation.

Figure 1. The Satellite Autonomy Application Evaluation Testbed Maximizes the Use of Existing and Modified Tools While Providing Flexibility and Realism.

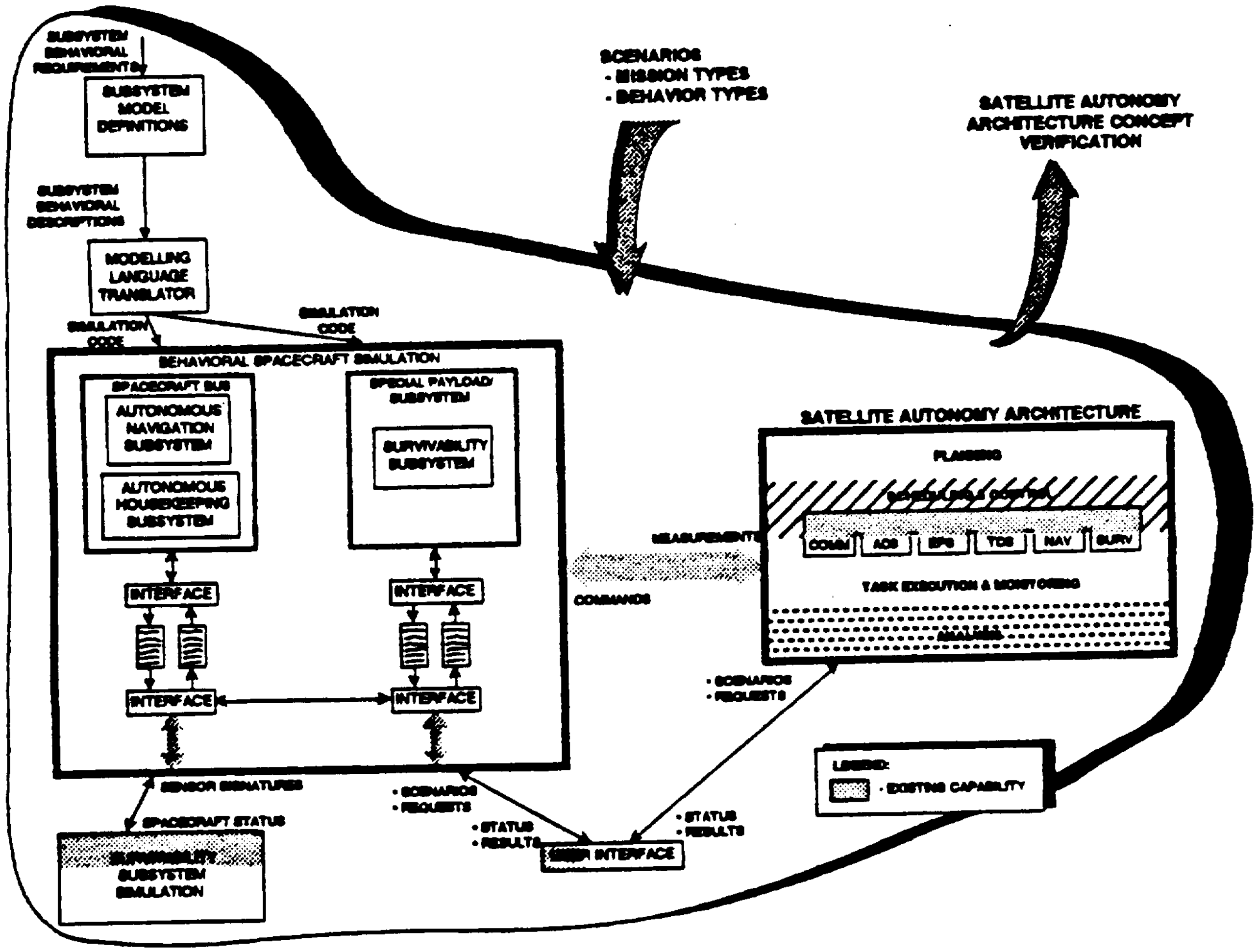

SUBSYSTEM BEHAVIORAL REQUIREMENTS
SUBSYSTEM MODEL DEFINITIONS
SUBSYSTEM BEHAVIORAL DESCRIPTIONS
MODELLING LANGUAGE TRANSLATOR
SIMULATION CODE
SIMULATION CODE
BEHAVIORAL SPACECRAFT SIMULATION
SPACECRAFT BUS
AUTONOMOUS NAVIGATION SUBSYSTEM
AUTONOMOUS HOUSEKEEPING SUBSYSTEM
SPECIAL PAYLOAD/ SUBSYSTEM
SURVIVABILITY SUBSYSTEM
INTERFACE
INTERFACE
INTERFACE
INTERFACE
SENSOR SIGNATURES
SPACECRAFT STATUS
SURVIVABILITY SUBSYSTEM SIMULATION
• SCENARIOS
• REQUESTS
• STATUS
• RESULTS
USER INTERFACE
• STATUS
• RESULTS
• SCENARIOS
• REQUESTS
SCENARIOS
- MISSION TYPES
- BEHAVIOR TYPES
SATELLITE AUTONOMY ARCHITECTURE CONCEPT VERIFICATION
SATELLITE AUTONOMY ARCHITECTURE
PLANNING
COMM
ACS
EPS
TCS
NAV
SURV
TASK EXECUTION & MONITORING
ANALYSIS
MEASUREMENTS
COMMANDS
LEGEND:
▨ - EXISTING CAPABILITY

Figure 2. An Integrating Architecture Has Been Defined and Developed to Provide Validation for a Satellite Operations Prototype.

fying simulation requirements and features in the form of a modelling language

- automated translator from the modelling language to CLIPS code which can be executed
- separability of generic spacecraft features from specialized components, subsystems, and payloads
- interface to an existing survivability simulation
- interface to an existing intelligent satellite operations framework
- interface to a graphical user interface

Architecture Description

We are using CLIPS as our basic programming language to create the modelling language, language translator, and simulation itself. The modelling language allows an engineer to specify the behavior of a system or subsystem in high-level terms that could be directly derived from specifications. The translator takes the modelling language constructs, verifies their consistency, and creates CLIPS knowledge bases which can be executed. The simulation uses the CLIPS forward-chaining mechanism as the driving force behind a system that is scalable to real-time events. Time can be specified directly or used in relative terms to compress or expand time to meet user requirements.

Satellite Modelling Language (SML)

The modelling language was created to provide a higher level interface to the identified end-user, a spacecraft design engineer. This interface allows the engineer to input requirements and features in a format which is familiar. This promotes a more rapid acceptance and utilization of the testbed resource resulting in increased productivity and the exploration of a larger number of engineering options.

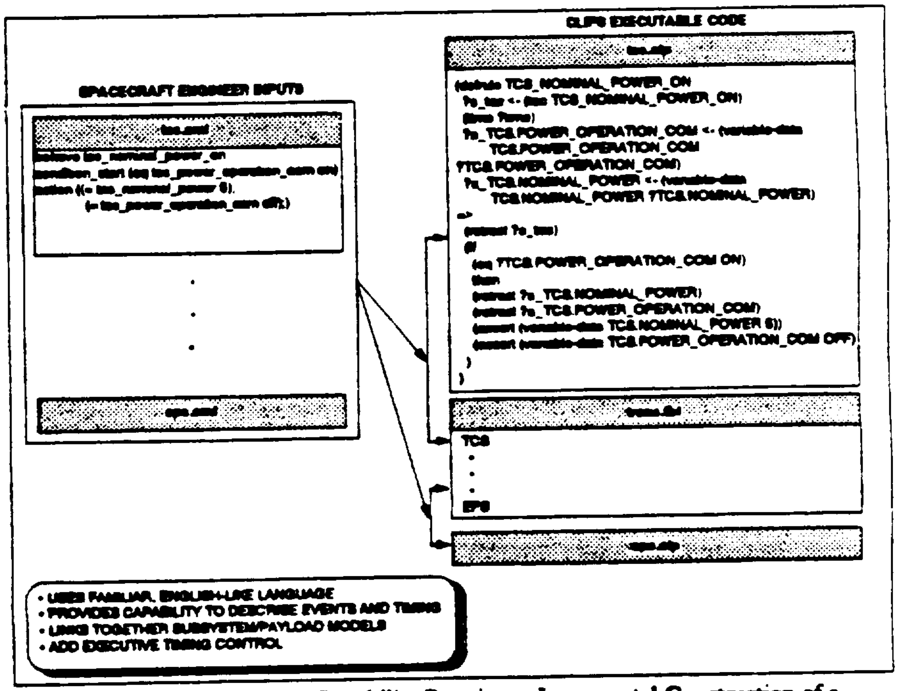
SML provides context-relevant and English-like language constructs to the spacecraft engineer. Through these constructs, the capability to describe events and timing is provided. This is accomplished through the use of three main structure types: templates, objects, and rules. Templates define conglomerations of objects, objects relate to physical or functional entities, and rules describe the behavior of the objects for various conditions.

The simulation itself uses CLIPS' forward-chaining technique to create a reactive and dynamic model of a spacecraft in its orbiting environment. Since spacecraft typically operate in a data- and situation-driven environment, CLIPS is a perfect match. Processes on a satellite are usually invoked on either a time or event basis. The stimuli cascade through many devices and components to achieve the necessary and required states. Side effects of component actions are relied upon heavily on spacecraft. These factors closely match the advantages of a system built with CLIPS.

Modelling language translator

The modelling language translator accepts the simulation specification from the engineer and converts it into CLIPS knowledge bases which can be executed (refer to Figure 3). This circumvents the need for the spacecraft engineer to become familiar with a new, and probably very different, software language. Also, since the CLIPS simulation code is automatically generated, the proper syntax and semantics are maintained within the knowledge bases. CLIPS is being applied in a manner much like an information compiler.

The translator accepts the SML constructs and converts them into CLIPS-acceptable syntax. Templates and objects are converted to facts while behavior rules translate into CLIPS rules. The CLIPS rules handle all the


SPACECRAFT ENGINEER INPUTS
CLIPS EXECUTABLE CODE
TCS
EPS
• USES FAMILIAR, ENGLISH-LIKE LANGUAGE
• PROVIDES CAPABILITY TO DESCRIBE EVENTS AND TIMING
• LINKS TOGETHER SUBSYSTEM/PAYLOAD MODELS
• ADD EXECUTIVE TIMING CONTROL

```
behave tcs_nominal_power_on
condition_start (eq tcs_power_operation_com on)
action ((= tcs_nominal_power 0),
        (= tcs_power_operation_com off);)
```

```
(defrule TCS_NOMINAL_POWER_ON
 ?e_ter <- (ter TCS_NOMINAL_POWER_ON)
 (time ?time)
 ?e_TCS.POWER_OPERATION_COM <- (variable-data
     TCS.POWER_OPERATION_COM
?TCS.POWER_OPERATION_COM)
 ?e_TCS.NOMINAL_POWER <- (variable-data
     TCS.NOMINAL_POWER ?TCS.NOMINAL_POWER)
=>
 (retract ?e_ter)
 (if
  (eq ?TCS.POWER_OPERATION_COM ON)
  then
  (retract ?e_TCS.NOMINAL_POWER)
  (retract ?e_TCS.POWER_OPERATION_COM)
  (assert (variable-data TCS.NOMINAL_POWER 0))
  (assert (variable-data TCS.POWER_OPERATION_COM OFF))
 )
)
```

Figure 3. The Translator Capability Permits an Incremental Construction of a Spacecraft Simulation.

bookkeeping involved with the behavior such as retracting facts after they are no longer required and asserting the pertinent facts.

The translator permits the incremental construction of a complete simulation capability. In practice, the modules are aligned with the subsystem designs. For instance, the Thermal Control Subsystem (TCS) templates, objects, and behavior rules are all defined within a single file. The translator maintains a list of all possible constructs and allows the linking of these in any manner specified by the user. The linking procedure also adds the executive timing control to the executable simulation.

Satellite simulation

The satellite simulation generation methodology is represented in Figure 4. Two parallel development paths have been identified for the creation of a realistic and dynamic evaluation environment for the IntelliSTAR™ prototype. One path concentrates on

"Black box testing is not an alternative to white box techniques. It is, rather, a complementary approach that is likely to uncover a different class of errors than white box methods." [8]

Behavioral models permit the description of the inputs and outputs of a function (or process or subsystem or ...). These models permit an empirical or high-level description of an entity. These models can be constructed quickly with readily available information and allow various levels of detail.

Functional models require an extensive evaluation of the theories and principles behind the operation of an entity. These models result from the classical design phase of an engineering process. Functional models have typically been developed in a monolithic mode. Good examples of functional model implementations are the current Computational Fluid Dynamics (CFD) codes being constructed.

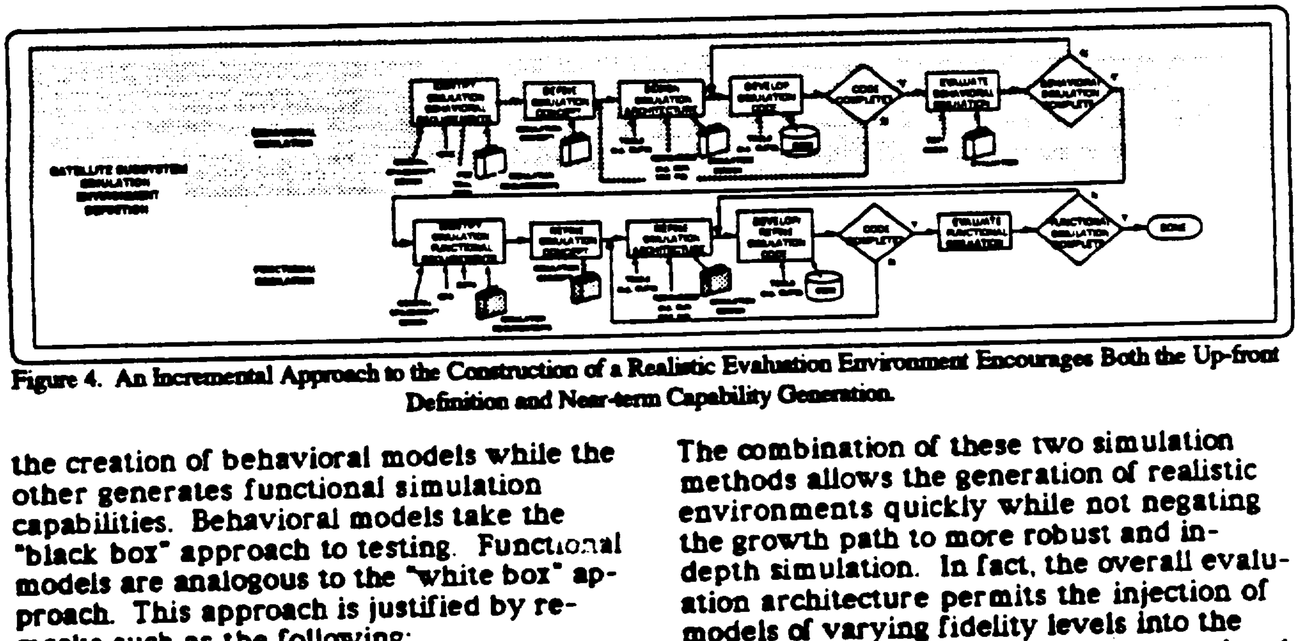


Figure 4. An Incremental Approach to the Construction of a Realistic Evaluation Environment Encourages Both the Up-front Definition and Near-term Capability Generation.

the creation of behavioral models while the other generates functional simulation capabilities. Behavioral models take the "black box" approach to testing. Functional models are analogous to the "white box" approach. This approach is justified by remarks such as the following:

The combination of these two simulation methods allows the generation of realistic environments quickly while not negating the growth path to more robust and in-depth simulation. In fact, the overall evaluation architecture permits the injection of models of varying fidelity levels into the same simulation. Behavioral and functional model can co-exist in the architecture. This provides a flexible medium for testing of the

IntelliSTAR™ prototype. In addition, the evaluation environment is not strictly tailored to that prototype, but also permits the construction of any satellite models.

The test architecture encourages a modular generation and management of its constituent parts. A conscious design decision was made to make the generic satellite bus characteristics separable from the specialized subsystems or payloads that comprise a spacecraft. By doing so, a generic capability for simulating spacecraft was created. This model will continue to evolve and the available "library" of models will increase as this effort proceeds. In fact, a major satellite effort at our division is contemplating the use of this capability because of the attractiveness of minimal cost to tailor the system for their purposes. Our research can continue in parallel with this satellite application since models can be interchanged with little effort.

Interfaces

Three types of interfaces currently exist to the simulation environment. These include one to the IntelliSTAR™ prototype, one to an existing survivability simulation, and the last to a user interface capability. The mechanism used for all three interfaces is the same; the results of a generic, distributed process communications project are utilized.

The interface to the IntelliSTAR™ prototype is implemented to allow the evaluation of this satellite operations concept. The interactions between the prototype and the simulation are of two types: continuous and requested. The first type, continuous, contains the telemetry stream content from the spacecraft to the controlling entity (i.e., IntelliSTAR™). The information flow is handshaked between the two portions but the interface is not truly synchronous. IntelliSTAR™ provides an execution time frame to the simulation and the simulation responds for that amount of time or at some smaller increment. The response time is solely determined by the simulation with only the upper bound specified by the prototype.

The second type of interface to IntelliSTAR™ is closer to being of the synchronous variety. A request is made of the simulation for information and the simulator responds with the derived data. The prototype may or may not wait for the results of its query before proceeding with its processing.

An interface with an existing survivability simulation (SADEM - Satellite Attack and Defense Engagement Model) was constructed. SADEM is constructed in an object-oriented and distributed environment. SADEM schedules a communications event to the spacecraft simulation at either a time or based on some condition. Currently, this interface is only one-way due to a limitation in the SADEM development environment.

The last interface is to the user interface module. This interface allows the control and execution monitoring of the simulation. Individual measurements being generated by the simulator may be presented with user-specified limits. Graphical representations of the data are allowed.

Conclusions

The simulation environment allows the integration of several levels of fidelity and the configuration of many diverse components. The modelling language translator assures the consistent generation of syntactically and semantically correct spacecraft simulations. The "garbage in, garbage out" syndrome of many simulations is minimized through the active application of knowledge about spacecraft in general. This approach, and associated testbed development, enables the creation of a sophisticated and consistent satellite simulation environment used for the design, manufacture, and analysis of satellites and their related operations environments.

References

[1] "Galileo Represents Peak in Design Complexity". Aviation Week and Space Technology, Oct. 9, 1989.
[2] Mitchell, Robert R., "Expert Systems and Air Combat Simulation". AI Expert, September 1989.
[3] Rao, Nageswara S.V., "Algorithmic Framework for Learned Robot Navigation in Unknown Terrains". IEEE Computer Magazine, June 1989.
[4] Brown, Marc H., "Exploring Algorithms Using Balsa-II". IEEE Computer Magazine, May 1988.
[5] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement". EEE Computer Magazine, May 1988.
[6] Gathmann, T.P., L. Raslavicius, and J.M. Barry, "A Unified Concept for Spacecraft System- and Subsystem-level Automation". 24th Intersociety Energy Conversion Engineering Conference (IECEC-89), Aug. 6-11, 1989.
[7] Gathmann, T.P., and L. Raslavicius, "Satellite Autonomy Generic Expert Systems (SAGES)". AIAA Computers in Aerospace, Oct. 4-6, 1989.
[8] Pressman, Roger S., "Software Engineering - A Practioner's Approach". McGraw-Hill Company, 1987.