

Distributed Simulation for Formation Flying Applications

Garett A. Sohl* Santi Udomkesmalee[†]

Jennifer L. Kellogg[‡]

Jet Propulsion Laboratory, Pasadena, CA, 91109, USA

High fidelity engineering simulation plays a key role in the rapidly developing field of space-based formation flying. This paper describes the design and implementation of the Formation Algorithms and Simulation Testbed (FAST).¹ This testbed was designed to provide real-time, high-fidelity engineering simulation of multiple spacecraft operating in formation. By distributing the simulation across multiple CPUs, the FAST provides an analyst or software developer with a scalable architecture for examining timing, sensor data-fusion, inter-spacecraft communication and system-wide formation robustness. The testbed is designed to maintain a very flight-like feel, with flight software algorithms running on single-board PowerPC 750 VxWorks computers. The flight computers communicate with the distributed simulation computers over a reflective memory interface. As part of recent development and testing activities, the FAST was used to simulate the Formation Control Testbed (FCT).² FCT is a ground-based, robotic testbed built at JPL. Each robot uses compressed air to float a five degree of freedom attitude platform over a flat floor. The FCT attitude platform has a flight computer, compressed air thrusters, reaction wheels, gyros and a star tracker.

I. Introduction

Formation flying is defined as the maintenance of desired relative position and orientation among multiple spacecraft. The concept of formation flying has applications in a variety of regimes: multiple robots,^{3,4} underwater vehicles,⁵ satellites, aircraft⁶ and spacecraft.^{7,8} Both NASA and the ESA are considering upcoming mission concepts that employ constellations of multiple, low-cost spacecraft. For space applications, formation flying spacecraft provide several benefits over a single, monolithic spacecraft. The inherent redundancy of a multi-spacecraft system yields better fault tolerance compared to the single vehicle approach. Additionally, a formation of spacecraft is highly reconfigurable and upgradeable. New assets can be added to the formation and existing assets can be moved within the formation in order to expand the capabilities of the overall system.

Multi-spacecraft systems do come with their own set of challenges, however. The navigation and control of these coordinated and coupled systems includes difficulties not present in a single vehicle system. A simulation environment that accurately captures the system complexity is needed in order to develop, test and validate control algorithms for formation flying systems. The simulation must be scalable, adaptable and reconfigurable to match the rapidly evolving and changing composition of a formation of spacecraft. The simulation must also meet the various fidelity requirements of a wide range of possible users. This

*Senior Staff Engineer, Jet Propulsion Laboratory, 4800 Oak Grove Drive, M/S 198-219, Pasadena, CA, 91109

[†]Associate Engineer, Jet Propulsion Laboratory, 4800 Oak Grove Drive, M/S 278, Pasadena, CA, 91109

[‡]Staff Engineer, Jet Propulsion Laboratory, 4800 Oak Grove Drive, M/S 192-235, Pasadena, CA, 91109

paper describes the design and implementation of a such a simulation environment, called the Formation Algorithms and Simulation Testbed (FAST).

A. Motivation

Formations of several spacecraft flying in close proximity form the basis for upcoming missions like the Terrestrial Planet Finder Interferometer (TPF-I),⁹ which is designed to detect earth-like planets orbiting distant stars. Other mission proposals involve tens, or even hundreds of spacecraft flying in formation. Despite the large number of spacecraft involved in a formation, there is no decrease in desired simulation fidelity. In fact, the simulation fidelity required for these missions (e.g. simulation of the interferometer optics of TPF-I) can often exceed those of existing single-spacecraft simulations. A scalable software and hardware architecture is needed to meet the needs of these upcoming formation-flying missions.

B. Testbed Design

The Formation Algorithms and Simulation Testbed (FAST)¹ is a distributed testbed designed for TPF-I class missions with formations of up to five spacecraft.¹⁰ It consists of five flight and five simulation computers. Each simulation computer is designed to simulate the dynamics and on-board hardware of one spacecraft in the formation. Dynamics can be either rigid or flexible body. The simulated on-board devices include IMUs, gyros, accelerometers, star-trackers, optics, reaction wheels and thrusters. Since each simulation CPU is dedicated to a single spacecraft, high-fidelity modeling (e.g. high frequency optics) is possible. In order to achieve the maximum benefit of distributing work across multiple CPUs, communication overhead must be kept to a minimum. Spacecraft state information is communicated between computers only when needed for relative sensing. The other form of communication between simulation computers is a simulated inter-spacecraft communication system, which utilizes a dedicated interface and is described in Section III.C.

In the FAST, the simulation computers control execution timing using RTAI Linux. Execution of flight software on a flight computer is controlled by interrupts sent over the reflective memory interface by a simulation computer. The simulation computers can use a global time signal for synchronous operation, or use local timing control. Under local time control, flight software interrupts occur at separate wall-clock times for each spacecraft in the formation. This behavior is similar to what would be experienced in flight as the local clock on each spacecraft may be skewed or drift with respect to another spacecraft in the formation. Flight software must be tolerant to this asynchronous behavior or provide algorithms to periodically synchronize the local clock cycles of each spacecraft. The simulation allows the analyst to examine the effects of these timing issues in a controlled environment.

Previously, the FAST has been used to simulate a two spacecraft formation flying scenario. More recently, the FAST was used to simulate the Formation Control Testbed (FCT) ground-based robotic system. The FCT is designed as ground-based hardware testbed for testing formation flying algorithms. Each robot uses compressed air to float a five degree of freedom attitude platform over a smooth floor. The FCT attitude platform has a flight computer, compressed air thrusters, reaction wheels, gyros and a star tracker. The FAST is used to simulate the FCT system and examine software robustness and stability before running software on the hardware.

II. Testbed Implementation

This section will describe the computer hardware, interfaces and software used in the FAST testbed.

A. Computer Hardware

As mentioned earlier, the FAST was designed for a TPF-I class mission of up to five spacecraft operating in formation. The testbed consists of five flight and five simulation computers, where each flight computer is coupled with one simulation computer. The flight computers are single-board, Motorola PowerPC 750s

running the VxWorks operating system. The simulation computers are dual-CPU, 1.8GHz AMD machines running Real-Time Application Interface (RTAI) Linux. RTAI is a hard real-time extension to the Linux kernel. Figure 1 shows a picture of the flight and simulation computer racks.



Figure 1. The FAST computer racks

In addition to the five dedicated simulation computers, there are two more computers on the same local network. The first is a head node computer that serves as a firewall and gateway between the simulation computers and the JPL internal backbone. This computer also maintains a RAID array for NFS disk access by the simulation computers. The second computer is a dual-head graphics node designed for visualization. It is located in the main lab area and is connected to two 50" plasma screens.

B. Interfaces

There are several network interfaces used in the FAST. These include fiber-optic reflective memory, two independent 100Mbit Ethernet channels and a low-latency, giga-bit Scalable Coherent Interface (SCI) channel. Figure 2 shows how these interfaces connect the various components of the FAST.

The fiber-optic reflective memory interface is used for communication between the simulation computers and the flight computers. This interface allows the simulation and flight computers to share a block of mutually addressable memory. For the flight computer, this memory is accessed identically to the memory of on-board hardware (with a suitable change of base address). The simulation computer can match the memory interface of on-board devices. By forcing the simulation to match the interface of hardware devices over reflective memory, the flight software is identical (with only base-address changes) when running with

FAST Hardware Configuration

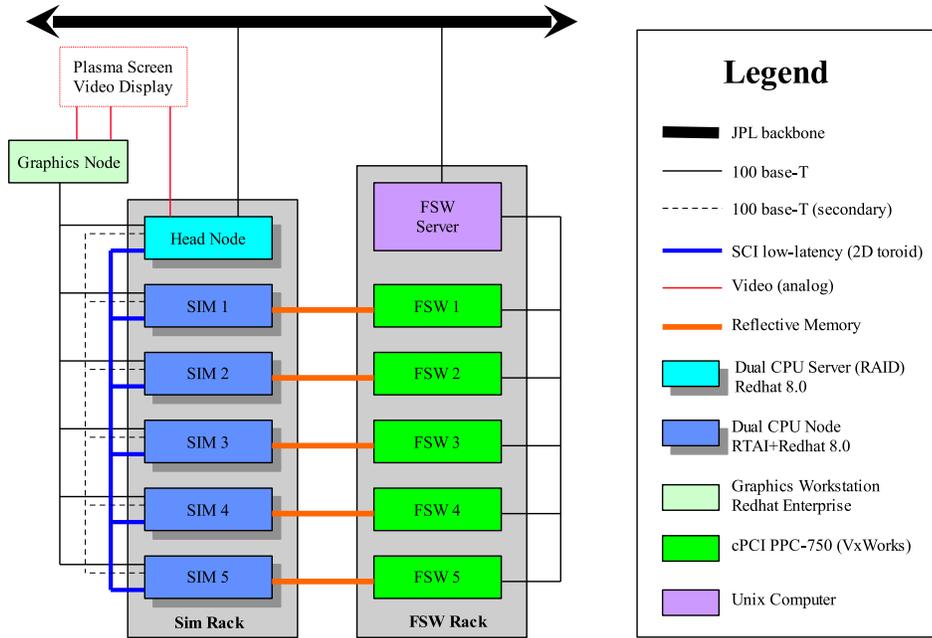


Figure 2. The FAST communication channels

either actual or simulated hardware.

Two independent 100Mbit Ethernet channels allow communication between the simulation computers. The first channels is dedicated to normal traffic (NFS disk access, ssh, etc.) and is used by the simulation computers only for time-insensitive data transmission. The second channel is free of any extraneous network traffic and is dedicated to the transmission of time-sensitive engineering data between simulation computers.

The final interface channel between the simulation computers is a Scalable Coherent Interface (SCI). Each simulation computer has a SCI card from Dolpin Interconnect Solutions, Inc. The simulation computer SCI network is arranged in a 2D-torus configuration, and provides micro-second level latency for transmitting time critical messages between simulation computers. In the FAST, the SCI is typically used for the inter-spacecraft communication model - passing small messages between simulation computers with extremely low latency.

C. Software Architecture

The FAST uses the Hierarchical Distributed Reconfigurable Architecture (HYDRA).^{11,12} HYDRA is a C++ based, lightweight client-server architecture that uses a publish-subscribe paradigm for inter-client communications. HYDRA shares many features with other architectures^{13,14} designed for distributed applications, but strives to provide more flexibility in connectivity services and behavior overriding.

HYDRA makes extensive use of C++ classes to provide a highly configurable system. Connectivity between clients is described by the **Services** class. Each client publishes available **Services** with the server at run-time. Once a matching provider and consumer for a given service is detected by the server,

the two applications are commanded to create a point-to-point direct connection for communication. The `HYDRAConnector` class is used to abstract the transport interface (SCI or 100Mbit Ethernet) between the simulation computers. Each `Service` can specify a desired transport mechanism or leave the decision up to the server application. Data is exchanged between clients using the `Packet` class, which provides automated serialization, error-checking and data marshaling. Figure 3 shows the various class-level components of HYDRA.

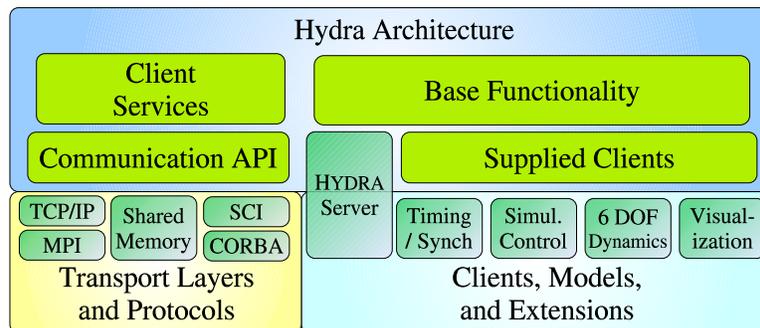


Figure 3. HYDRA Architecture

By allowing HYDRA to forge and manage the connections between client applications, we reduce the burden on the simulation developer. The developer can concentrate on simulation design issues (e.g. performance, fidelity, etc.) and not worry about the various distributed communication issues that are handled by HYDRA. HYDRA is not meant to eliminate the often difficult job of creating a simulation suitable for a distributed environment. The simulation developer must still strive to maintain a clean interface between distributed simulation components. A well designed simulation should be a modular and self-contained collection of applications. A well designed simulation client allows the user to choose the number of spacecraft in formation merely by changing the number of simulation computers. A formation flying simulation of n spacecraft would involve running a simulation module on n separate computers.

In the FAST, the head node is used to run a HYDRA server application. Each simulation computer runs one or more HYDRA client applications that encapsulate the simulation of a single spacecraft in the formation. HYDRA automates the connection of the client applications described in Section III that comprise each spacecraft simulation.

III. Spacecraft Simulation

A fundamental component of a distributed formation flying testbed is an independent simulation of each individual spacecraft. The continuous states (linear and angular position and velocity) of each spacecraft are affected by on-board actuator devices (e.g. thrusters) and, in turn, affect the output of on-board sensor models. Simulating the independent motions of the formation of spacecraft is the basic step in creating a formation flying simulation. However, the spacecraft in a formation are not isolated and independent systems. They communicate data through inter-spacecraft communication systems. This data is used in both control and sensing for the overall formation.

A. Dynamics

Simulating the dynamics properties of a spacecraft is a task whose complexity can greatly vary. A useful dynamics model might be an extremely simplistic point mass body or an elaborate model including numerous flexible body modes. This point is especially validated by formation flying simulations, where simulation requirements can vary from a simple rigid body model used for high-speed ACS simulation to a high-fidelity,

multi-mode, flexible body model used for high-frequency simulation for simulating the optical system of pointing mirrors, delay lines, sensors, etc.

These challenges require the use of a modeling architecture for dynamics simulation that spans a wide range of model fidelity. DARTS,¹⁵ Dynamics Algorithm for Real-Time Simulation, is a high-fidelity, flexible multi-body dynamics simulator used for real-time simulation. DARTS facilitates creating a dynamical system comprised of an open chain of bodies, which are defined by mass, center of mass location and inertia. DARTS supports both rigid and flexible bodies, allowing the user to supply stiffness and damping matrices for each body. Bodies are connected together in a chain using joints. DARTS supports a variety of joint types, including pins, sliders, planar, gimbals, 3-DOF translations, 3-DOF spherical, ball (quaternion), and full 6-DOF (3-DOF translation + ball). DARTS computes joint accelerations given the forces applied at any point on the multi-body chain.

A numerical integrator is used to integrate the accelerations computed by DARTS and advance the system state forward in time. In the FAST, a C++ integrator class was created that allowed the user to select the numerical integration method. A high-speed Euler, a fourth order Runge-Kutta and the adaptive, variable-step CVODE¹⁶ integrators are all supported. The choice of integration method is based on the required problem fidelity. Figure 4 shows how DARTS interfaces to physical actuator and sensor models and supplies derivatives for numerical integration.

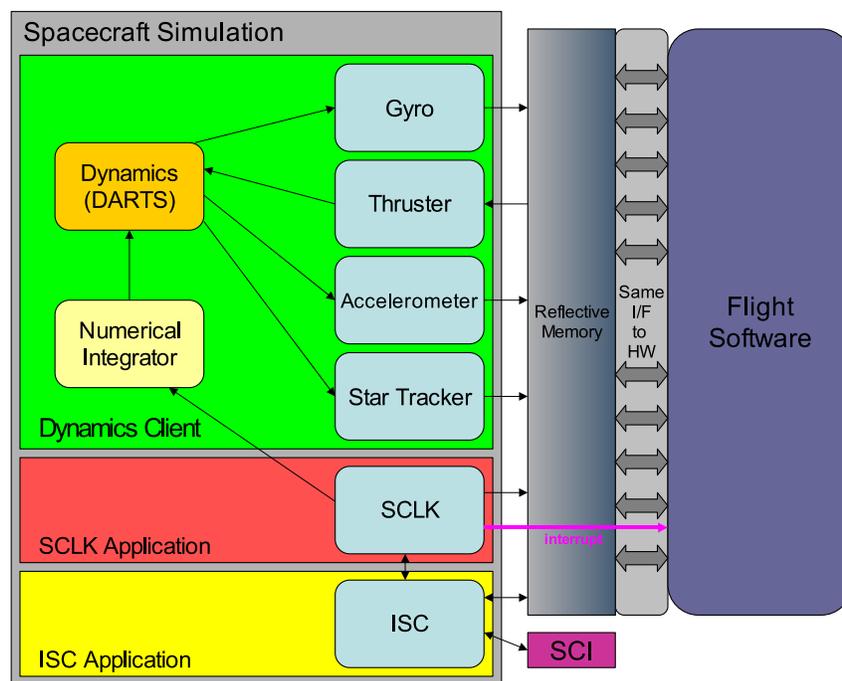


Figure 4. Block diagram of simulation components and interfaces

B. Devices

A spacecraft might be carrying any number of on-board devices such as IMUs, gyros, accelerometers, star-trackers, optics, reaction wheels and thrusters. The capabilities and characteristics of these devices are important components of a complete simulation. Figure 4 shows the typical components of a spacecraft

simulation in the FAST, including the interface with flight software. The simulation and its component device models present flight software with an interface that is identical to the actual hardware.

On-board device models are attached to a DARTS *node*, which represents the location of the physical hardware in a real system. A node can be located at any point on a body and may have its own reference frame. DARTS provides an API for retrieving positions, attitudes and velocities at any node location. Values can be returned in node, body or inertial reference frames. DARTS also provides an API for setting the applied force and torque at a node for use by actuator models such as thrusters. A lightweight C++ model class was created for the FAST. This model class interfaces to the DARTS API and provides the ability to actuate the spacecraft (apply forces/torques at a given node) and sense resulting motions (retrieve linear accelerations and angular velocities of a given node). Additionally, the model class provides an interface to the numerical integrator so that each model can have its own continuous states that are integrated in parallel with the dynamics states. Any model can schedule its own output rate. For example, an IMU model might output data at 10Hz, while a star tracker model output data at 1Hz.

C. Inter-spacecraft Communication

Inter-spacecraft communication (ISC) plays a critical role in formation control. A formation that uses a leader-follower paradigm may rely heavily on ISC in order to send commands from the leader spacecraft to the followers. Even when control is distributed between the spacecraft in formation, ISC is often used to transmit sensor readings or other engineering data. A successful formation flying simulation requires an accurate representation of ISC. The simulation must accurately represent the message timing and failure modes (e.g. packet dropout) in order to test the stability and robustness of the overall formation under closed-loop control.

The ISC model created for the FAST simulates the communication between spacecraft in a distributed spacecraft system. It models transmission delays, communications jitter, timed dropouts and probabilistic dropouts. In addition, it uses the Spacecraft Clock (SCLK) model (described in Section IV.A) to accurately time stamp messages. The model utilizes the hard-real time utilities of the RTAI Linux operating system to periodically poll for and process incoming messages. Messages to and from flight software are sent over the reflective memory interface. Messages between simulation computers are transmitted using the low-latency, SCI communication channel as shown in Figure 4. The SCI channel is dedicated to ISC message traffic in the FAST and provides micro-second level latencies when passing the typically small ISC messages. By utilizing these system capabilities, the ISC model can provide a simulation which aids in testing the control system's robustness to message absence and timing irregularity.

IV. Timing

The FAST is designed to provide real-time simulation. To accomplish this, the simulation uses the capabilities of the RTAI operating system. The simulation controls the advancement of system time and provides the flight software computers with periodic interrupts over the reflective memory interface. As in a real mission, each simulated spacecraft has its own clock - in the FAST this is the local CPU clock for each simulation computer. Generally, the local CPU clock for a given simulation computer will have drift, offset and jitter with respect to another simulation computer. This is consistent with what flight software running on distributed spacecraft in formation would experience. A simulated Spacecraft Clock (SCLK) model running on each simulation computer allows us to modify the local clock behavior.

A. SCLK model

In a distributed formation of several spacecraft, each spacecraft has its own concept of "time". The Spacecraft Clock (SCLK) model, simulates the physical clock running on each spacecraft. While not typically an important aspect of a single-spacecraft simulation, the SCLK model plays an important role in a formation

flying simulation since spacecraft are operating in close proximity and are often in constant communication. Since no two clocks are exactly alike, variations between the SCLKs on two spacecraft in a formation can cause events (like a flight software RTI) to occur at different times and cause time-tagged messages to be represented in a different time frame.

The FAST implements a SCLK model on each spacecraft simulation computer. The SCLK model provides a flight software readable time value that is periodically incremented (FAST uses a 1ms update) as time progresses. In addition, the SCLK model provides a periodic interrupt over the reflective memory interface that is used to signal and coordinate processes in the flight software. Simulation events are triggered via callback functions that the spacecraft client registers with the SCLK model. In the FAST, the spacecraft dynamics client registers two callback functions with the SCLK model to provide callbacks for advancing the state forward in time as shown in Figure 5.

While the SCLK model uses the RTAI operating system for precise real-time performance, there will still be some small drift between two “real-time” applications running on separate computers. This drift is caused by frequency differences of the clocks on separate computers and can be caused by a variety of factors, including temperature variation. Consequently, the FAST SCLK model utilizes the NTP infrastructure in order to get an estimate on the local CPU clock drift with respect to a high precision source like the Global Positioning System (GPS). NTP synchronizes the simulation computer locals to within 1ms. Using the RTAI operating system, as well as the NTP estimated frequency errors, the SCLK models running on separate computers can be closely synchronized. In order to test the flight software’s sensitivity and robustness to clock variation and synchronization, the user can “dial in” controlled offset and drift values for the SCLK model.

As the spacecraft’s concept of “time”, the SCLK model is used for time tagging information generated by each spacecraft. This information includes sensor data as well as inter-spacecraft communications. Typically, flight software requires synchronization of events occurring on spacecraft in the formation (e.g. TDMA communication windows for ISC). As each spacecraft is controlled by its local clock, the FAST must allow flight software to command a delay (with respect to SCLK time) in order to synchronize events between spacecraft. In order to determine the correct delay, software must first detect the offset between two local clocks. In the FAST, flight software uses ISC to periodically (20 seconds) send a time-tagged, round trip message between two spacecraft computers to determine the offset between the local clocks. This information is then used to synchronize the event timing between spacecraft. For example, if spacecraft A determines that spacecraft B’s SCLK is offset by 10ms, it can command an event (that normally occurs at time t) to occur on spacecraft B at time $t+10\text{ms}$ (according to B’s SCLK) while the same event occurs on spacecraft A at time t (according to A’s SCLK). This delay mechanism is used by the FAST to synchronize the flight software RTI event loops on each spacecraft in the formation.

B. Integrator Control

As described in the previous section, the SCLK model controls the execution of flight software on the flight computers by sending periodic interrupts across the reflective memory interface. The SCLK model also controls the time advancement of the spacecraft dynamics client through periodic function callbacks. Using the SCLK to control the dynamics client allows us to schedule simulated sensor and actuator models to operate at a rate governed by the local spacecraft clock. As in flight, each simulated spacecraft is controlled by its own, independent clock. Unlike a physical system, the simulated system must integrate the equations of motion. Since each system is distributed, each integrator advances independently. If the SCLK advancement rate does not match the wall clock, the integration of the equations of motion for one spacecraft can become desynchronized with another spacecraft in the same formation. This would invalidate any state-based information transmitted between spacecraft simulations since they would be in separate time frames.

This problem is addressed in the FAST by sending a Pulse-per-Second (PPS) signal from a real-time application running on the head node. The PPS signal is received by each spacecraft’s simulated ISC model.

It is used to compute and drift between the local spacecraft SCLK model and the wall clock time (as defined by the PPS signal). If a spacecraft simulation detects a SCLK model running faster than wall clock, it must reduce the amount of time integrated during every second of SCLK time advancement. For example, if a simulated spacecraft SCLK model advances 2.0 seconds between PPS signals, it must integrate the equations of motion for 0.5 seconds for every one second of SCLK time. This synchronizes the dynamic state of the spacecraft with another spacecraft in formation whose SCLK is advancing 1.0 second between PPS signals. Generally, this drift between the local SCLK and the global PPS signal is very small. In the FAST, each spacecraft client compensates for this offset by increasing or shortening the time advanced by the integrator between reading thruster commands and writing sensor data as shown in Figure 5. While this is slightly simplistic, it is sufficient for small amounts of drift.

C. Timing Diagram

There are several applications running on each dual-CPU simulation computer. These applications include:

- **SCLK Client:** This application updates the current SCLK time to reflective memory at 1ms intervals. It controls the advancement of the dynamics client through callback functions. It also controls the execution of the flight software control loops through interrupts sent over reflective memory.
- **Spacecraft dynamics client:** This HYDRA client integrates the dynamical equations of motion for the spacecraft, reads thruster commands from reflective memory and writes on-board device data to reflective memory
- **ISC:** This application sends and receives ISC messages over the SCI interface

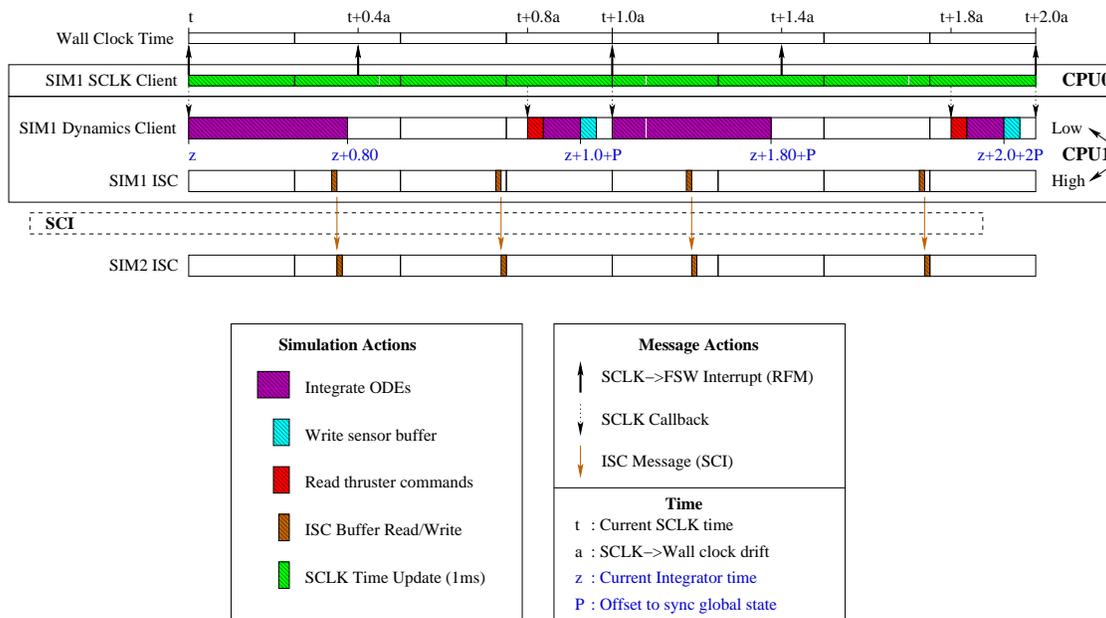


Figure 5. Timing Diagram

Figure 5 shows timing diagram for these applications.

The ISC and spacecraft dynamics client both run on the same CPU. The ISC application is given a higher priority so that it will interrupt the dynamics client when a new ISC message appears in either the SCI buffer or the reflective memory buffer shared by the ISC model and the flight software computer.

There are three different concepts of *time* in Figure 5. The first is *wall clock time*, which advances at a universally constant rate. The second is *SCLK time*. The SCLK model allows the user to select a drift and offset for the SCLK so this time will not necessarily advance at the same rate as wall clock time. The last concept of time is *integrator time*. This is the current integration time of the numerical integrator. Section IV.B describes how we use a periodic signal from a master SCLK to synchronize the global system state of the integrators used by the distributed spacecraft dynamics clients.

V. Example: The Formation Control Testbed

The Formation Control Testbed (FCT) is a ground-based, hardware testbed designed to examine aspects of formation flying for the Terrestrial Planet Finder mission. While this paper generally refers to each object in a formation as a “spacecraft”, for FCT the objects are referred to as “robots”. FCT consists of three autonomous robots navigating in a 40’ diameter, circular room on a flat floor. Each robot consists of an attitude platform on top of a translation platform.

The translation platform floats over a flat floor on three air-bearing feet and carries air tanks for compressed air supply. These on-board air tanks supply air for the feet as well as a spherical air bearing that supports the attitude platform. The spherical air-bear between the translation and attitude platforms provides 3 rotational degrees of freedom. The air bearings provide 5 degree of freedom motion for the attitude platform. A vertical stage, to be fitted to the translation platform and allow vertical motion of the spherical air bearing, will provide a sixth degree of freedom. The FCT robot carries enough on-board air supply for approximately 30 minutes of operation.

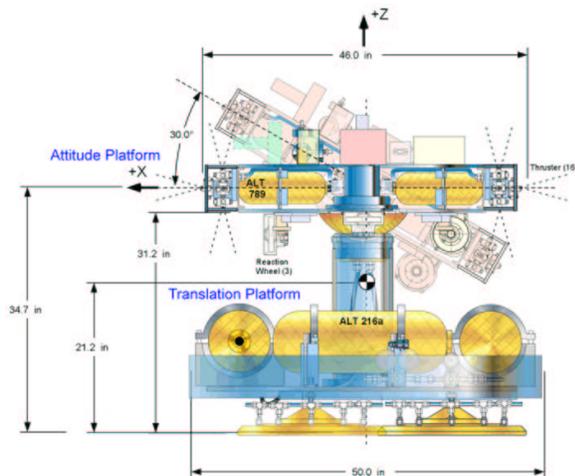


Figure 6. Diagram of FCT robot with dimensions



Figure 7. FCT robot hardware

The attitude platform contains a flight-like avionics suite for the robot. This suite includes a PowerPC 750 computer, sixteen compressed-air thrusters, three reaction wheels, gyroscopes and a celestial sensor.¹⁷ A rechargeable battery supplies power to an on-board PowerPC 750 computer running the VxWorks operating system. A wireless 802.11g channel is used to communicate with the the on-board computer - allowing fully untethered operation. The attitude platform has air tanks for supplying sixteen compressed air thrusters. These thrusters, along with three single-axis reaction wheels provide full attitude control authority. By combining thrusters that produce opposing moments, but cooperative forces, the attitude platform thrusters also provide linear actuation and allow the FCT robot to translate in two degrees of freedom over the flat floor. Figure 6 show a diagram of the FCT robot and figure 7 shows a picture of the actual hardware.

A. FAST for FCT

The FCT system provided the FAST development team with a unique opportunity to validate the simulation architecture against a physical system. The FCT robot's on-board computer uses the same processor as the flight computers in the FAST (Figure 1). The FCT simulation in the FAST is designed to the same Interface Definition Document (IDD) as the actual hardware. This allowed the FAST team to test and validate the simulations ability to mimic actual hardware devices over the reflective memory interface. Figure 8 shows a screen capture of the FCT robot simulation in the FAST.

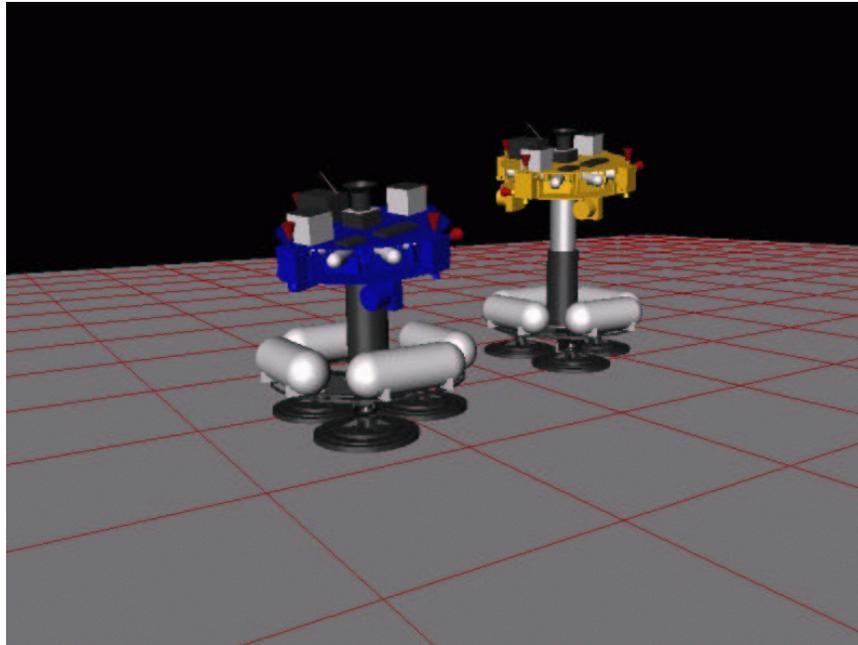


Figure 8. Screen capture from 2-robot FCT simulation in the FAST

The FAST simulation provides a valuable tool to the FCT flight software development team. New flight software builds are first tested using the simulated robots before loading on the physical system. This procedure allows debugging to occur in the simulation testbed, where variables such as thruster performance, sensor noise, external disturbances, ISC dropouts and clock drift are user controllable. The configurable nature of the FAST simulation allows the algorithm developer to test the overall system robustness. As the FCT system is still being procured (only one robot and a small portion of the flat floor is currently installed at JPL as of August, 2005), the FAST simulation also provides the algorithm developer with the ability to design software for hardware that does not yet physically exist. The flight software designed for formation control of multiple FCT robots is currently undergoing performance testing in the FAST distributed simulation even though the second FCT robot will not be delivered until winter of 2005. The value of this developmental testing is directly related to the ability of the FAST simulation to accurately represent the various components of the real system.

VI. Conclusion

The Formation Algorithms and Simulation Testbed (FAST) provides a distributed simulation environment suitable for a range of formation flying applications. FAST is comprised of a set of simulation computers

and a set of flight computers. A reflective memory interface is used for communication between the simulation and flight computers. Although the FAST was designed for a five spacecraft, TPF-I class mission, the simulation environment is scalable to encompass formations of any number of spacecraft using the HYDRA architecture. HYDRA automates the inter-connection of any number of distributed simulation components at run-time using a publish-subscribe paradigm. The simulation components include spacecraft dynamics client, inter-spacecraft communication and spacecraft clock models. The distributed nature of the simulation components closely mirrors the distribution found in real spacecraft formations - with individual local clocks and dedicated communication protocols. Recently, a simulation of the FCT robotic system was developed for testing and validating the FAST simulation tool. FAST has proved to be a useful and timely tool in software development for the FCT system.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract with the National Aeronautics and Space Administration. We would also like to acknowledge the Terrestrial Planet Finder project's support of development of the FAST.

References

- ¹Wette, M., Sohl, G., Scharf, D., and Benowitz, E., "The Formation Algorithms and Simulation Testbed," *2nd International Formation Flying Symposium*, Washington DC, September 2004.
- ²Regehr, M., Acikmese, A., Ahmed, A., Aung, M., Bailey, R., Bushnell, C., Clark, K., Hicke, A., Lytle, B., MacNeal, P., Rasmussen, R., Shields, J., and Singh, C., "The Formation Control Testbed," *IEEE Aerospace Conference*, Montana, 2004.
- ³Wang, P. K. C., "Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation," *Journal of Robotic Systems*, Vol. 8, No. 2, 1991.
- ⁴Fierro, R., Das, A., Kumar, V., and Ostrowski, J., "Hybrid Control of Formations of Robots," *IEEE Conference on Robotics and Automation*, Seoul, Korea, May 2001.
- ⁵Stilwell, D. and Bishop, B., "Platoons of Underwater Vehicles," *IEEE Control Systems Magazine*, Vol. 20, No. 6, December 2000.
- ⁶Giulietti, F., Pollini, L., and Innocenti, M., "Autonomous Formation Flight," *IEEE Control Systems Magazine*, Vol. 20, No. 6, December 2000.
- ⁷Beard, R., Lawton, J., and Hadaegh, F., "A Coordination Architecture for Spacecraft Formation Control," *IEEE Control Systems Technology*, 1999.
- ⁸Ren, W. and Beard, R., "A Decentralized Scheme for Spacecraft Formation Flying via the Virtual Structure Approach," *Journal of Guidance, Control and Dynamics*, Vol. 27, No. 1, January 2004.
- ⁹Aung, M., Ahmed, A., Wette, M., Scharf, D., Tien, J., Purcell, G., and Regehr, M., "An Overview of Formation Flying Technology Development for the Terrestrial Planet Finder Mission," *IEEE Aerospace Conference*, Montana, 2004.
- ¹⁰Aung, M., Ahmed, A., Wette, M., Scharf, D., Tien, J., and Purcell, G., "An Overview of Formation Flying Technology Development for the Terrestrial Planet Finder Mission," *IEEE Aerospace Conference*, Montana, March 2004.
- ¹¹Martin, B. and Sohl, G., "HYDRA : High-Speed Simulation Architecture for Precision Spacecraft Formation Simulation," *AIAA Modeling and Simulation Technologies Conference*, Aug. 2003.
- ¹²Martin, B., "HYDRA: Advances in the High-Speed Distributed Simulation Framework," *AIAA Modeling and Simulation Technologies Conference*, San Francisco, CA, August 2005.
- ¹³Dahmann, J. S., Fujimoto, R., and Weatherly, R. M., "The Department of Defense High Level Architecture," *Winter Simulation Conference*, 1997, pp. 142-149.
- ¹⁴de Sousa, J. B. and G. A., "A Simulation Environment for the Coordinated Operation of Multiple Autonomous Underwater Vehicles," *Winter Simulation Conference*, 1997, pp. 1169-1175.
- ¹⁵Jain, A. and Man, G., "Real-Time Simulation of the Cassini Spacecraft Using DARTS: Functional Capabilities and the Spatial Algebra Algorithm," *5th Annual Conference of Aerospace Computational Control*, Aug. 1992.
- ¹⁶Cohen, S. and Hindmarsh, A., "CVODE, a Stiff/Nonstiff ODE Solver in C." *Computers in Physics*, Vol. 10, No. 2, March-April 1996.
- ¹⁷Shields, J., "The Formation Control Testbed Celestial Sensor: Overview, Modelling, and Calibrated Performance," *IEEE Aerospace Conference*, Montana, March 2005.