Use of the HLA in a Real-Time Multi-Vehicle Simulator

H.C. Chao[1], T.W. Pearce[1], M.J.D. Hayes[2]

[1]*Department of Systems and Computer Engineering, Carleton University*
hcchao@engsoc.org, pearce@sce.carleton.ca

[2]*Department of Mechanical and Aerospace Engineering, Carleton University*
jhayes@mae.carleton.ca

This paper outlines the use of a High Level Architecture (HLA) compliant design for the Carleton University Simulator Project (CUSP). HLA is a modular interoperability standard (IEEE 1516) for combining distributed, networked simulations. Interoperability is achieved through the standardization of the communication interfaces between simulation components. Reduction of large, monolithic simulations into smaller, component based simulation modules allows for the distribution of processor intensive computations across multiple computers, alleviating the need for a powerful single computer. Additional advantages also arise because of the component-based nature of the HLA including reusability, modularity and expandability. Use of an HLA compliant design in CUSP has allowed for independent, concurrent software development allowing for flexibility in project planning and management. Issues with undefined and changing requirements are also now manageable allowing for future changes and expandability as CUSP evolves.

1. INTRODUCTION

The High Level Architecture (HLA) specifications [1] were developed to enable modularity and interoperability to simulation design. Simulators and simulation technologies provide for the controlled reproduction of real life conditions and experiences. Simulator training for commercial and military pilots, along with operator training for other vehicle types is a well-known example. Perhaps less well-known applications of simulator training are in air traffic control, power generation, and health care.

The capability of a simulator or simulation to generate test conditions approximating operational or actual conditions to a high level of fidelity is of paramount importance as a design objective. High fidelity in terms of the environmental stimuli affecting the operator ensures that training in a simulator is positive, and is transferable to a real life situation.

This paper focuses on issues associated with creation of an HLA compliant simulation facility at Carleton University in particular, and design of real-time systems in general. The control and communication software is designed for a 6 degree-of-freedom motion platform within the Carleton University Simulator Project (CUSP)[2], administered within the Department of Mechanical and Aerospace Engineering. The motion platform is designed to be reconfigurable and the control system fully interoperable. This highlights an initiative to offer 4th year students in the Faculty of Engineering and Design a variety of large scale, multidisciplinary, industrially relevant capstone design projects in a virtual enterprise environment.

The results presented in this paper are largely the contribution of the 2002-03, and 2003-04 CUSP Systems teams, lead by faculty and students from the Department of Systems and Computer Engineering. The paper is organized in the following way. Section 2 will describe the evolution of the HLA, and define relevant concepts and terminology. Section 3 gives an overview of CUSP, its scope and technical objectives. This leads to Section 4 wherein the implementation of the HLA in CUSP is detailed. Additionally, an important HLA learning exercise, PoolSim is described. The paper design is described and the two proof-of-concept technology demonstrators from the first two years of CUSP are discussed. Next, lessons

learned from the implementation of the HLA are highlighted. The final section contains conclusions and suggestions for future work.

2. THE HLA

The HLA was developed originally by the US Department of National Defense, with the goal of incorporating interoperability, modularity and reusability into ambitious long-term simulation objectives [3]. The approach taken by the HLA views simulations as components in larger systems, which is a style more akin to product development than that of traditional monolithic simulations. The component-oriented view encourages and focuses attention on interfacing concerns, and how components interoperate to accomplish an objective. Ideally, this approach enables components to be reused more easily, and HLA-compliant commercial-off-the-shelf (COTS) components have been used successfully in the construction of simulations. The standardization of HLA components and processes has been embraced by industry, and the HLA is now established in the public domain as the IEEE Standard 1516-2000. The IEEE has mandated the Simulation Interoperability Standards Organization (SISO) [4] to carry out periodic public reviews and updates to the HLA specification. The review process is currently underway, and some incremental extensions are expected to be approved in 2005.

In the HLA, individual simulation components are called *federates*, and the collection of components that comprise a simulation is called a *federation*. The HLA specification consists of three parts: a set of ten rules that constrain federates and federations, the Object Model documentation Template (OMT) for federates and federations, and the Runtime Infrastructure (RTI) programmer's interface (API). The rules governing federates and federations are simple and straightforward. For example, one of the rules states that federates must only interact using RTI services. The OMT documentation standards ensure that any information exchanged among federates is specified. This interfacing information is essential to achieve interoperation among federates. The documentation can be used for a variety of purposes prior to runtime; however, the information is also used at runtime in support of RTI services. The RTI is a middleware (software) layer that implements HLA services at runtime. The HLA only specifies the API, and therefore a variety of RTI implementations are acceptable.

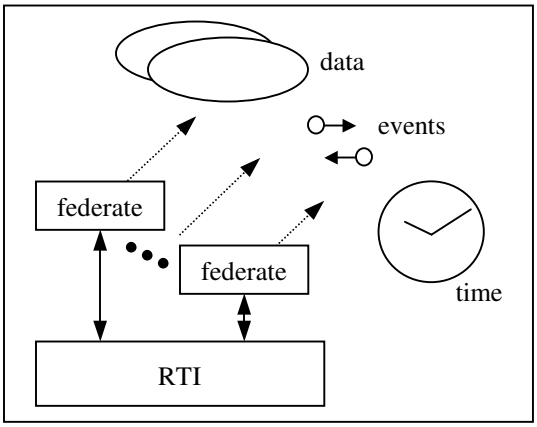

Figure 1 Abstract view of a federation

Figure 1 shows an abstract view of a federation at runtime. The federates use RTI services to accomplish all interactions. The RTI allows the federates to share persistent data, instantaneous events and simulation time. Note that the abstract view does not imply an underlying computing architecture. The federates might all be executing on the same computer, they might each have their own dedicated computer, or they may be mapped to the computing architecture in some other fashion. Abstracting federates away from the underlying computing architecture simplifies scaling the processing power to meet the needs of a federation.

The interoperation of federates requires federates to share information. The sharing of data and events is accomplished using an object-oriented publish/subscribe mechanism. The concept of a *class* is used to define types, and then objects are created as instances of the classes at runtime. Federates make information available by publishing, and obtain information by subscribing. The RTI manages communications between the publisher of specific information and the subscribers to that information. While the sharing of data and events is typical of distributed application components, the sharing of simulation time is a unique characteristic of simulation components, and a critical aspect of HLA interoperability.

The RTI services are organized into categories: **Federation Management**: allows federates to create, join, leave and destroy a federation. **Declaration Management**: allows federates to declare the classes of the objects that they will publish, and the classes of the objects they will

subscribe to. **Object Management**: allows federates to create, modify and delete shared objects (instances). **Ownership Management**: only the owner of an object may modify the object, and these services allows federates to exchange the ownership of shared objects. **Distributed Data Management**: allows federates to define abstract, simulation-specific regions that can help to reduce runtime communication overheads. **Time Management**: allows federates to share a global notion of time, and to synchronize local activities in global time. The simulators described in this paper use services from all categories except Distributed Data Management.

The RTI services have been designed to accommodate many different simulation styles. As a result, most federates will use only the subset of services that are appropriate to the needs of the federate. While this may seem superficially obvious, patterns in the use of RTI services have a significant impact in the interoperability and reuse of components.

Individual federates communicate with one another through *ambassadors*, as shown in Figure 2. The RTI Ambassador allows the federate to invoke RTI services, and thereby interoperate with other federates. The Federate Ambassador allows the RTI to callback to the federate to inform the federate of interoperations originated by other federates. When developing federates, the federate-specific code must be programmed, including the behaviour to be performed during Federate Ambassador callbacks. To simplify concerns over thread-safe code, the federate must call the RTI tick service periodically. During the tick call, the RTI will perform any callbacks that may be pending.

3. CUSP

The *Carleton University Simulator Project (CUSP)* is a 4th year capstone, multiyear design project (in one-year phases) with a short project cycle time (8 months per phase). The 2003/04 academic year is the second phase. The complexities of CUSP are typical of any large project. There are 35 participants, including faculty and students from the Departments of Mechanical and Aerospace Engineering, and Systems and Computer Engineering, as well as consultants from industry. The project requires that current participants learn and build upon the work from previous years. As a result, participants are faced with a large learning curve that needs to be overcome quickly. The project participants are divided into teams and groups, depending on each individual's specific areas of expertise and interests. This allows for individuals to participate in a manner conducive of their unique skills and abilities.

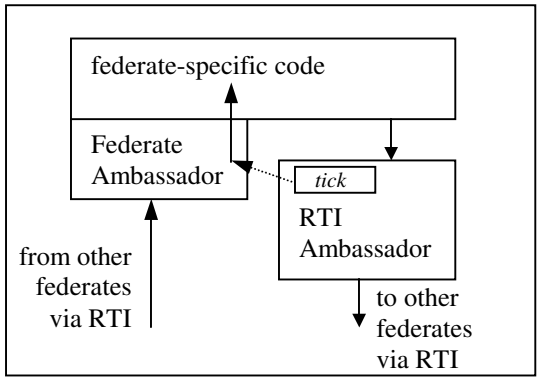

Figure 2 Federate ambassadors

The long-term goal of the project is to design and implement a full scale, six degree of freedom motion platform with a novel architecture. The platform will be reconfigurable to support multiple vehicle simulations for operator training. The simulation facility will be used to support a variety of simulation needs of industry, academia and government, and ideally, the facility will be self-supporting. The facility will meet industry standard quality, and be compliant with all government and university safety regulations. The project will provide a valuable learning and training environment for engineering students as well as providing a marketable facility to enhance simulation needs at Carleton University and the surrounding region.

In the first year of the project, platform motion requirements to meet the long-term CUSP goals were determined to be:

- +/- 18 " displacement for sway, surge and heave
- +/- 30° rotation for pitch, roll and yaw
- 0.5 g maximum acceleration in all directions
- 500 lb. payload.

Initial research revealed the Stewart platform [5] (or more properly the Stewart-Gough platform [6]) to be a popular design, but the coupling of the six degrees of freedom has led the project to consider alternate kinematic configurations. One implication of the coupling is that at the heave limit of the Stewart-Gough platform no yaw is possible. Additionally, since this is largely an engineering education driven project, innovation for innovation's sake is feasible. The final target platform has been named NASP (Not A Stewart

Platform), and several innovative designs have been proposed. NASP objectives include decoupling the orienting from the positioning degrees of freedom. An immediate benefit is the simplification of mathematical modeling, which allows for incremental expansion of the control of each degree of freedom. To explore the feasibility of the design features and to gain the experience necessary to better understand design issues, a proof-of-concept technology demonstrator platform has been developed. In the first year, a platform with a single translational degree of freedom, named SiDFreD (**Si**ngle **D**egree of **Fre**edom **D**emonstrator), was designed and built. In the second year, the demonstrator has been extended to include two decoupled rotational degrees of freedom, and has been named SIDFreD (**S**everal **I**ntegrated **D**egrees of **Fre**edom **D**emonstrator) to reflect the change.

CUSP participants are organized into teams of approximately five students, with a faculty member serving as lead engineer. The Systems (SYS) Team is responsible for the computing infrastructure, and a motivating goal for the team is the use of the HLA as the underlying infrastructure for CUSP platforms. In the first year, the team focused on a computing architecture suitable for NASP and SiDFreD. Over the past two years, the team's scope has expanded to include various sensors, development environments, and development of a business plan. SYS Team members also participate in broader cross-team activities associated with safety, human factors, washout algorithms, manufacturing, procurement, assembly, system integration and project management.

4. THE HLA IN CUSP

The goals of CUSP impose broad and challenging technical issues for the SYS Team. In addition to the software engineering implications of realizing a framework for a reconfigurable simulation, motion simulators also require real-time performance while incorporating both hardware and humans in the simulation loop. As a result, the SYS Team must view the target platform as both a simulation and an embedded, distributed, real-time system.

The HLA was chosen as the underlying architecture for CUSP platforms because:

- the component-oriented approach of the HLA lends itself to the software engineering principles of information hiding and encapsulation, which in turn encourages concurrent development of components
- the abstraction provided by the RTI allows the underlying computing architecture to be expanded and distributed easily, without requiring further programming
- the use of standardized, third party RTI middleware reduces the amount of supporting software that must be developed and maintained in CUSP
- the reuse goals of the HLA are well-suited to the short cycles of CUSP and the need for cycles to reuse the work of previous cycles
- HLA compliance enables the integration of CUSP platforms with other HLA-compliant simulations
- the HLA represents the state of the art in simulation interoperability standards.

The work of the SYS Team has centred on designing an overarching NASP computing architecture (both hardware and software) that can evolve with future requirements, and the implementation of the architecture for the technology demonstrators. Using the HLA as a guiding infrastructure has simplified and accelerated this process.

The use of the HLA in CUSP is not entirely without drawbacks. The HLA has a comprehensive set of services designed to support a wide variety of simulation styles. The HLA learning curve, and the lack of relevant and readily available examples, are limiting factors for deploying the HLA in an academic project with tight time constraints. To help offset this, the first year SYS Team developed PoolSim, a real-time simulation of a ball rolling on a pool table, as a learning exercise. The PoolSim approach to real-time was reused while developing the SiDFreD simulator, and thereby reduced the number of technical issues encountered. The second year SYS Team familiarized themselves with the HLA by extending PoolSim with additional functionality. Again, the learning experience greatly simplified their subsequent step into the SiDFreD environment.

4.1 PoolSim

PoolSim was developed as a learning exercise to become familiar with HLA issues, and in particular, to explore a technique for obtaining real-time performance. Although sharing simulation time among federates is a central element of the HLA, a limitation of the HLA is that it does not specify support for real-time behaviour. As a result, real-time synchronization must be introduced by simulation components. The functional objective of PoolSim is to simulate a ball rolling on a pool table in real-time. This application provides a gentle introduction to many of the aspects of a typical simulator, including: a graphics display showing the ball on the table, physics calculations to update the position of the ball periodically, user control to start, stop and monitor the simulation, and real-time synchronization.

The PoolSim federation was designed to consist of the Timer, User, Physics and Display federates. The federates are shown in Figure 3, and briefly described below.

The Timer federate introduces real-time synchronization by injecting a time event into the federation every 1/60 of a second (i.e. at a 60 Hz frequency), the minimum graphics refresh rate required in training simulators. PoolSim is implemented for a Windows platform, and the implementation of the Timer federate required some low-level Windows programming to reduce jitter in the time events.

The User federate provides an operator's interface to control the federation. Operator input controls the start/stop of the simulation, and allows the size of the table to be sized dynamically. The User federate publishes table data and control information.

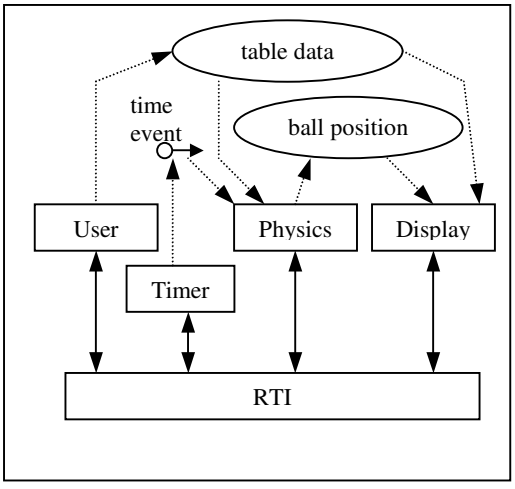

Figure 3 PoolSim Federation

The Physics Federate calculates and publishes the ball's position and maintains a local value of the ball's velocity. To accomplish this at a regular interval, the federate subscribes to the 60 Hz time events injected by the Timer federate and to the table data provided by the user federate. When a time event is received, the federate calculates a new position and velocity for the ball based on the current values and the table parameters (size, rolling friction, and bumper dynamics). The new position value is published once it is calculated.

The Display federate displays the ball on the table. The federate subscribes to the table size and ball position data and refreshes the display whenever any subscribed value changes. Changes to the ball position occur at 60 Hz creating the illusion that the ball is rolling on the table.

The federation is initialized with default table data and the Timer federate in *idle* mode, where it is not generating any time events. The control information published by the User federate is used to put the Timer in *running* mode, where it generates periodic time events. Since the simulation calculation is time triggered, the simulation can be paused and resumed easily by toggling the Timer federate mode between *idle* and *running*.

4.2 SIDFreD (Year 1)

SIDFreD, shown in Figure 4, is the translational motion platform demonstrator designed and manufactured in the first year of CUSP. The purpose of the platform was to demonstrate the feasibility of implementing and manufacturing an HLA compliant motion. In keeping with the vehicle simulator theme of CUSP, SiDFreD was configured for a ground vehicle driving simulation with the single degree of freedom corresponding to vehicle sway.

The driver's platform was constructed with aluminum, and mounted on pillow blocks and rails. A chain connected to a 2 hp vector motor moves the platform. The motor is equipped with a controller card that supports a serial connection to a computer. The platform supports the driver's cockpit, which includes a seat with a five-point restraining harness, a force-feedback PC gaming steering wheel with matching pedal and brake set, and a projector to display the simulated road being driven.

Figure 4 SiDFreD driven by 4th year student Nicholas Spooner.

A block diagram showing SiDFreD's physical subsystems and their interconnections is shown in Figure 5. Two networked PC workstations running Windows were used as computer subsystems. The sensor subsystem consists of an optical mouse mounted on the movable driver's platform. The sensor provides an accurate and inexpensive way to track the current position of the platform. The position is used to verify that the platform motion is consistent with the CarWorld output, and to support a software controlled kill switch should the platform's motion encroach on safety limits. The software-independent cut-off switches associated with the safety subsystem provide further safety.

The software subsystems running on the computers handle all processing requirements for the simulator. The software is organized as a set of HLA-compliant federates, and the RTI middleware hides all aspects of distributing the federates between the computers.

To simplify the software development task, an open source car driving simulation, called CarWorld [7] was modified to meet SiDFreD's requirements. CarWorld was chosen because it provided a complete driving simulation including a vehicle dynamics model, a graphical display of a simple road (using OpenGL [8]), and force-feedback steering wheel and pedal controls (via DirectX [9]). Some modifications were made to improve the visual image of the road, but a more significant effort was required to convert CarWorld from a standalone program into an interoperable HLA compliant federate.

Converting CarWorld into an HLA federate involved designing how it would interoperate

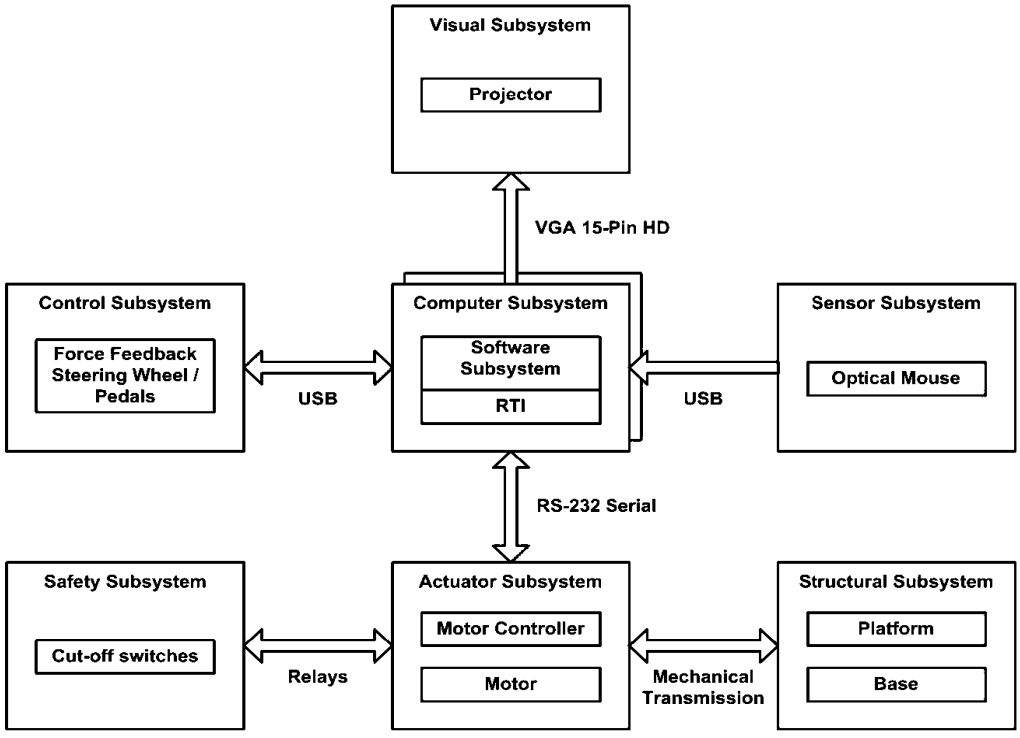

Figure 5 SiDFreD physical subsystems

with other federates, and the deciding what information should be published and subscribed. The team decided that all of CarWorld's existing capabilities would be utilized and that the lateral acceleration, velocity and position of the vehicle should be published. The published vehicle state data could then be used by other federates to drive the platform actuator to create corresponding lateral vehicle motions. To provide data updates, the CarWorld federate was designed to subscribe to a periodic time event (similar to that used in PoolSim). The arrival of each time event triggers the generation and publication of new vehicle state data. The necessary vehicle state variables were already present in the CarWorld vehicle dynamics model, and therefore extending the code to created the CarWorld federate involved:

- using HLA Federation Management services to join the SiDFreD federation
- using HLA Declaration Management services to declare the intention to publish the vehicle state data and subscribe to time events
- using HLA Object Management services: to the register instances of the vehicle state variables, to receive callback notifications when time events occur, and to update the vehicle state values when new values are calculated.

The SiDFreD federation is shown in Figure 6. For simplicity, only the federates and the shared information are shown. The **Timer** federate is similar to that described for PoolSim, and generates time events at a 60 Hz frequency. The **CarWorld** federate (described above) manages the driver interface and the driving simulation. The federate subscribes to time events, and triggers subsequent federation activity by publishing updates to the vehicle state data. The **Mouse** federate also subscribes to time events and publishes the current platform state (position, velocity and acceleration) when each time event occurs. The **Dynamics** federate accounts for the physical limitations of the platform's motion envelope and actuator. The federate converts the vehicle state data into (published) platform dynamics to be realized by the platform. In addition the federate monitors the current platform state to verify that the platform is moving as expected. If the platform does not move as expected, it shuts down the platform motion, thereby implementing a software safety switch. The **Actuator** federate converts the desired platform dynamics into motor control commands and sends the commands to the motor controller. The federate also polls the controller and publishes the actuator state. The **Monitor** federate provides the passive logging of all shared information, and proves a simulator operator's interface to control the federation.

4.3 SIDFreD (Year 2)

In the second year, SiDFreD (above) has evolved into SIDFreD, with two additional rotational degrees of freedom. The car vehicle simulation theme has been retained; however, the CarWorld application was modified to create a more open and flexible architecture. Second year objectives include introducing new vehicle dynamics models, incorporating washout algorithms, handling safety more prominently, adding decoupled degrees of freedom incrementally, and the reusing the previous physical and software components. The second year SYS Team was able to evolve the first year design to meet all of

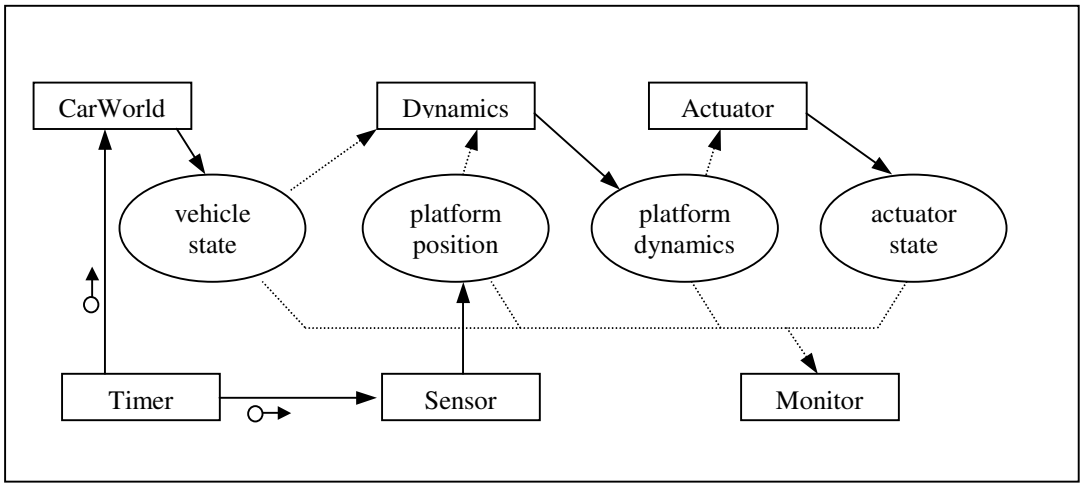

Figure 6 SiDFreD Federation

these objectives.

Figure 7 shows a simplified representation of the resulting SIDFreD federation architecture. When compared to Figure 6, an Application module has replaced the previous CarWorld federate, the Washout federate has replaced the previous Dynamics federate, the Safety federate has been introduced, and Actuator modules have been introduced for each actuator.

The Application module has decoupled vehicle dynamics from CarWorld and introduced a new Vehicle Dynamics federate. The modified CarWorld federate manages user inputs and the display of road as the vehicle is driven. A higher fidelity vehicle dynamics model has been developed to interoperate with the modified CarWorld. From the perspective of other federates, the original CarWorld federate and the new Application module are interchangeable, since both publish and subscribe to the same information.

The SIDFreD Washout federate is a modified and enhanced version of the previous Dynamics federate. Washout is a technique for combining translation and rotation motions together with visual and audio cues to trick an occupant's motion receptors into believing that they are experiencing an application-specific motion (for more details see [10]). The technique allows for the possibility of moving the platform in a manner different from the perceived application-specific motion, and ideally, allows the platform to be returned to the center of it's motion envelope. Returning to the center of the envelope allows the platform to make optimal use of the motion envelope in subsequent (washed-out) motions. The Washout federate subscribes to the vehicle state published by the Application module, and filters the platform dynamics to account for physical limitations and washout. The platform dynamics information published by the Washout federate must be described in terms of motions in each of the directions supported by attached Actuator modules.

Each Actuator module consists of an Actuator federate and a Sensor federate. The Actuator federate converts the appropriate dynamics information into actuator commands and passes these on to the actuator. When possible, the actuator also publishes actuator status information. The Sensor federate feeds back independently measured platform position information related to the actuator.

The Safety federate performs the safety functionality implemented in the previous Dynamics federate. The federate compares the intended platform dynamics specified by the Washout federate with the actual motions observed by the Sensor federates, and monitors the actuators' status. If unexpected motions or status conditions occur, then the federate generates an event to shut down the motions of the actuators.

Figure 7 shows a simplified representation of the resulting SIDFreD federation architecture. When compared to Figure 6, an Application module

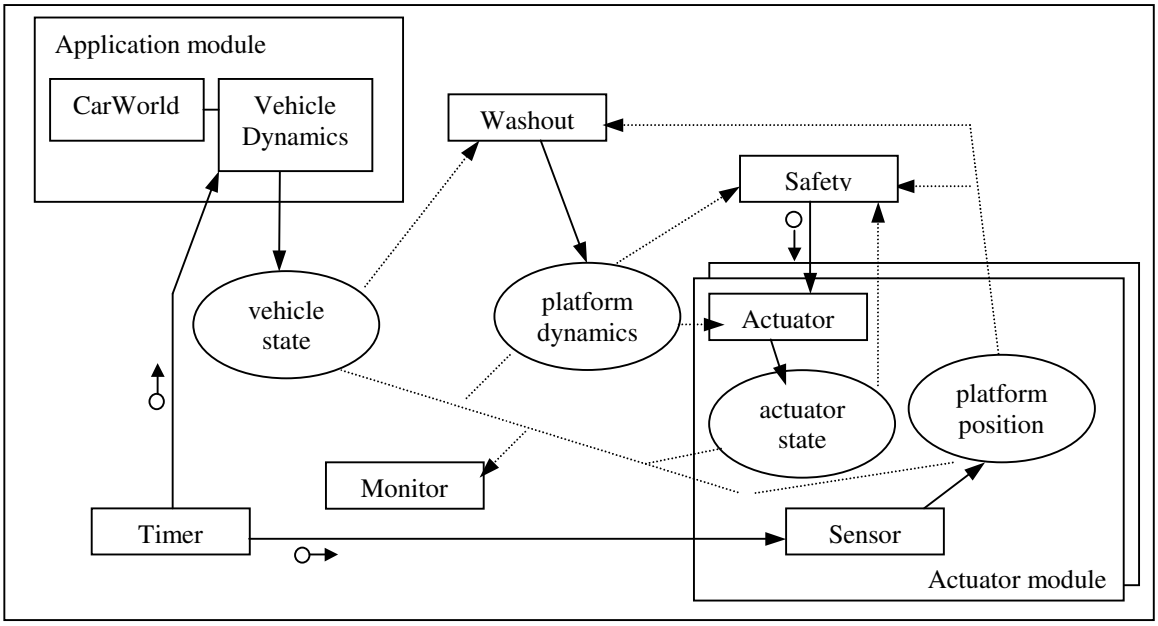

Figure 7 SIDFreD federation architecture

has replaced the previous CarWorld federate, the Washout federate has replaced the previous Dynamics federate, the Safety federate has been introduced, and Actuator modules have been introduced for each actuator.

The Application module has decoupled vehicle dynamics from CarWorld and introduced a new Vehicle Dynamics federate. The modified CarWorld federate manages user inputs and the display of road as the vehicle is driven. A higher fidelity vehicle dynamics model has been developed to interoperate with the modified CarWorld. From the perspective of other federates, the original CarWorld federate and the new Application module are interchangeable, since both publish and subscribe to the same information.

The SIDFreD Washout federate is a modified and enhanced version of the previous Dynamics federate. Washout is a technique for combining translation and rotation motions together with visual and audio cues to trick an occupant's motion receptors into believing that they are experiencing an application-specific motion (for more details see [10]). The technique allows for the possibility of moving the platform in a manner different from the perceived application-specific motion, and ideally, allows the platform to be returned to the center of it's motion envelope. Returning to the center of the envelope allows the platform to make optimal use of the motion envelope in subsequent (washed-out) motions. The Washout federate subscribes to the vehicle state published by the Application module, and filters the platform dynamics to account for physical limitations and washout. The platform dynamics information published by the Washout federate must be described in terms of motions in each of the directions supported by attached Actuator modules.

Each Actuator module consists of an Actuator federate and a Sensor federate. The Actuator federate converts the appropriate dynamics information into actuator commands and passes these on to the actuator. When possible, the actuator also publishes actuator status information. The Sensor federate feeds back independently measured platform position information related to the actuator.

The Safety federate performs the safety functionality implemented in the previous Dynamics federate. The federate compares the intended platform dynamics specified by the Washout federate with the actual motions observed by the Sensor federates, and monitors the actuators' status. If unexpected motions or status conditions occur, then the federate generates an event to shut down the motions of the actuators.

5. LESSONS LEARNED

The use of the HLA has been very beneficial in the CUSP project.

The general use of standards, like the HLA, has established an immediate connection with industry practice. This gives the students realistic direction and focus for their development efforts. An early focus is essential in the limited cycle times of the project, and encourages an attitude of engineering products to comply with standards as well as fulfilling the simulator-oriented requirements.

The depth of the HLA requires each cycle to invest a non-trivial learning effort; however, the return on the investment is well worth it. The PoolSim learning exercise is an excellent stepping-stone for the students to familiarize themselves with the HLA using a simple and related example. The transition to SIDFreD is a natural progression, and the PoolSim exposure to the real-time synchronization technique used in SIDFreD is a valuable asset.

The component-oriented nature of the HLA has greatly simplified the mapping of design components into implementation components, and has forced the interfacing and interoperation of components to be addressed before implementation. The goals of interoperability and reuse have been brought to the surface early in the design, rather than allowing them to lurk as issues to be discovered later in development, where they would be more problematic to address. Once the federate interfaces and information sharing within the federation had been designed, federates were developed concurrently, and the subsequent integration into a federation was not an awkward and time-consuming process.

The RTI middleware creates an extra software management load and additional runtime processing requirements; however, the savings in development time have been very positive. The federation management services greatly simplify the task of initializing and controlling a distributed application. The publish and subscribe mechanism allows distributed

components to share information, without having to develop a further sharing infrastructure. Perhaps the only limitation in the RTI services is the lack of a real-time synchronization mechanism; however, this has been solved easily using the Timer federate.

6. FUTURE WORK

As CUSP progresses into Year 3 and beyond, NASP and SIDFred will evolve towards the goal of implementing a simulator with 6 degrees of freedom. The current SIDFreD design has been developed with the explicit objective of simplifying the addition of additional degrees of freedom and should provide a solid basis for the next cycle. With the architecture in a stable state to accommodate additional actuators, attention can be turned towards more application-oriented areas including: display technology, washout, fidelity, and application configuration.

The current display technology consists of a simple projector mounted on the SIDFreD platform. This approach will likely not be feasible as additional degrees of freedom are added and the platform's motion becomes more comprehensive. Several alternatives are being considered, ranging from a virtual reality helmet with head tracker and goggles, through to projection in an enclosure around the driver. The current NASP design is leaning towards the enclosure approach, and SIDFreD will likely incorporate modifications to explore this option in the next development cycle.

An effective washout approach will combine motion with visual and audio cues. The presence of more than one degree of freedom in SIDFreD will permit greater use of washout in platform motion, and the Washout federate in the SIDFred architecture will be a center of attention. The coupling of the washout algorithm with visual and audio cues has not yet been addressed, and this is expected to require research and development to implement corresponding changes to the CarWorld application.

From the SYS Team's perspective, coupling washout with visual cues will require some ability to modify the display to respond not only to application purposes (e.g. the display as determined in response to the steering and pedal controls in CarWorld), but to respond to washout inputs as well. This concept should not be difficult to implement since the Washout federate can publish visual "dynamics" information, which can be subscribed to by CarWorld. Incorporating this into CarWorld will require further adaptation to subscribe to the Washout visual dynamics and adjust the display accordingly.

A major concern in the future will be improving the fidelity of simulators. The HLA's modular, component-oriented approach should support this by allowing federates to be replaced seamlessly by new federates which better simulate the environment. The ease of replacing and modifying components promises faster integration and testing, thus reducing the amount of time and effort involved. The continued benefits of the HLA will be a contributing factor in the future success of CUSP.

7. REFERENCES

[1] "*IEEE Standard for Modeling and Simulation High Level Architecture (HLA)",* IEEE Std 1516-2000

[2] *"Carleton University Simulator Project (CUSP) Final Report – Year 1"*, Department of Mechanical and Aerospace Engineering, Carleton University, 2003

[3] US Department of Defense, 5000.59-P, *"Modeling and Simulation Master Plan"*, October, 1995

[4] Simulation Interoperability Standards Organization, *World Wide Web, http://www.sisostds.org/*

[5] Stewart, D., "A Platform with Six Degrees of Freedom", *Proc. Institute of Mechanical Engineering*, Vol. 180, part 1, No. 5, pp. 371-386, 1965-1966

[6] Gough, V.E.. 1956, "Discussion in London: Automobile Stability, Control, and Tyre Performance", *Proc. Automobile Division, Institution of Mech. Engrs*., pp. 392-394.

[7] M. Hewat, CarWorld, v0.219, 2000, *World Wide Web, http://carworld.sourceforge.net*

[8] OpenGL, version 1.5, 2003, *World Wide Web, http://www.opengl.org/*

[9] Microsoft Corporation, DirectX, Version 9.0b, *World Wide Web, http://www.microsoft.com/windows/directx/*

[10] J.S. Freeman, G. Watson, Y.E. Papelis, T.C. Lin, A. Tayyab, R.A. Romano, and J.G. Kuhl. "The Iowa Driving Simulator: an Implementation and Application Overview", *1995 SAE World Congress*, 1995