# Development of a Next Generation Embedded Simulation Engine for FCS

**Henry Marshall**
**US Army Research, Development and Engineering Command, Simulation and Training Technology Center**
**Orlando, Florida**
Henry.A.Marshall@us.army.mil

**Charlie Ragusa, Stewart Grayson**

**Science Applications International Corporation (SAIC)**

**Orlando, Florida**
leonard.c.ragusa.ii@saic.com william.s.grayson@saic.com

**Gary Green**
**Institute for Simulation and Training (IST), University of Central Florida (UCF)**

**Orlando, Florida**
ggreen@ist.ucf.edu

## ABSTRACT

Embedded training is a Key Performance Parameter of the Future Combat Systems (FCS) program, and thus a requirement for fielding FCS systems. Historically, embedded training capabilities have been appended to existing system architectures as afterthoughts. Consequently, embedded training hardware and software have been distinctly separate from the operational system and consume space and power resources not planned for in the original design. For FCS, embedded training and operational capabilities are tightly coupled and development will proceed concurrently. The Embedded Combined Arms Team Training and Mission Rehearsal (ECATT-MR) Science and Technology Objective (STO) is exploring a new embedded training simulation engine for FCS, based on the OneSAF Objective System (OOS). OOS is being leveraged to provide critical support for modeling and control of both ownship and ancillary virtual vehicles (i.e. ground and air robotic platforms), in addition to the usual computer generated forces support. Early releases of OOS and the FCS System of Systems Common Operating Environment (SoSCOE) are being used to implement the architecture, and an FCS-like Infantry Carrier Vehicle (ICV) crewstation simulator is used as a testbed for its evaluation. This paper explores the issues, advantages and findings of this research and suggests future enhancements to the architecture.

## ABOUT THE AUTHORS

**Henry Marshall** is the Principal Investigator for Mounted Embedded Simulation Technology at the Research, Development and Engineering Command (RDECOM) Simulation and Training Technology Center (STTC). Prior to this assignment he worked at the Simulation, Training and Instrumentation Command (STRICOM) where he spent 11 years as lead for the CGF/SAF, HLA and Linux Porting developments on the Close Combat Tactical Trainer (CCTT) system in addition to being a OneSAF team member. His twenty years with the Government have been mainly in CGF and Software acquisition. He received a BSE in Electrical Engineering and an MS in Systems Simulation from the University of Central Florida.

**Charlie Ragusa** is a software engineer with SAIC. Four of Mr. Ragusa's five years of software development experience have been in the field of modeling and simulation. He has participated in a variety of diverse projects including the Modular Interoperable Synthetic Environment (ModISE) for the U.S. Army STRICOM, Embedded Systems in support of the U.S. Army RDECOM ECATT/MR STO, the Deployable Virtual Training Environment (DVTE) for the U.S. Marine Corps TECOM, and the NASA-funded Psychosocial Performance Factors Testbed. Mr. Ragusa is experienced in DIS protocol, HLA, object-oriented analysis and design, and is a Sun Certified Programmer for the Java 2 Platform. Mr. Ragusa received his BS in Computer Science from the University of Central Florida.

**Gary Green** is a research associate at the Institute for Simulation and Training (IST), University of Central Florida (UCF) with experience in management and research of training and simulation programs. His recent experience includes eight years managing research projects exploring embedded simulation and embedded training issues in support of Army research and development. He is currently the principal investigator for IST's Mounted Embedded Simulation Technology work with US Army RDECOM STTC. His MS in Operations Research is from the US Naval Postgraduate School.

# Development of a Next Generation Embedded Simulation Engine for FCS

**Henry Marshall, Jeff Stahl**
**US Army Research, Development and Engineering Command, Simulation and Training Technology Center**
**Orlando, Florida**
Henry.A.Marshall@us.army.mil

**Charlie Ragusa, Stewart Grayson**
**Science Applications International Corporation (SAIC)**
**Orlando, Florida**
leonard.c.ragusa.ii@saic.com william.s.grayson@saic.com

**Gary Green**
**Institute for Simulation and Training (IST), University of Central Florida (UCF)**
**Orlando, Florida**
ggreen@ist.ucf.edu

## THE NEED FOR A NEW SIMULATION ENGINE APPROACH

The next major training paradigm shift is rapidly underway. This is the move from training based on schoolhouse simulator training to embedded training that will be available on demand in the operational systems wherever they may be deployed. This paradigm shift is made possible by the recent quantum leap in image generator and computational capabilities and the accompanying hardware miniaturization. Despite these improvements significant challenges still exist for embedded training. Embedded systems are typically restricted in size and power consumption as evidenced by the recent Embedded Simulation Request for Proposal for Stryker, which required a single windows- based PC host with power consumption not to exceed 100 Watts with a minimum of 2 X 700 MHz processors. This host must support semi-automated forces (SAF), a terrain database sufficient for precision gunnery, image generation (35Hz or better) and after action review (AAR) capability. This clearly shows the demands placed on restricted computational assets for embedded training. In addition, fielded systems require ruggedized components, giving them computational systems that are typically several generations behind currently available COTS systems. This makes the need for an efficient simulation engine within embedded solutions a major design objective.

Figure 1 shows a high-level breakdown of likely FCS training system components. This array of components and their interactions within FCS present a number of issues.

At the highest level the FCS simulation engine must support fully embedded training across its family of systems. In addition, it must also support full and part task schoolhouse trainers. Ideally, the same simulation engine should support all of these training venues.

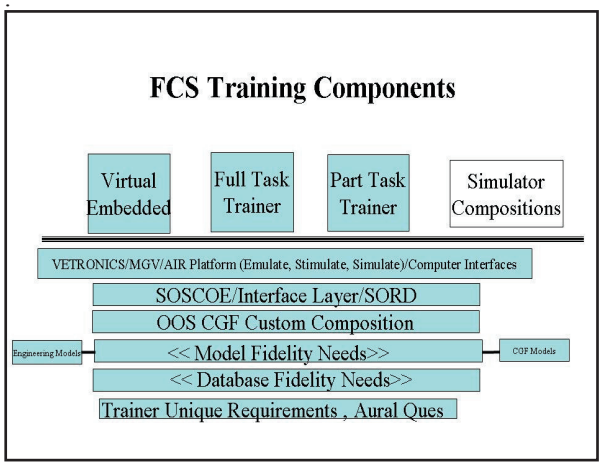


**Figure 1.** FCS Training Components

At the next architectural level, the embedded simulation system must communicate directly with the Vehicle Electronics (VETRONICS). The way in which the simulation interacts with each of the various vehicle subsystems is a major design issue. Broadly speaking the interactions can be grouped into three categories, as follows:

- Emulation – vehicle subsystem supports a special training mode that allows it to receive input from the simulation via a channel/interface that is dedicated to simulation. The subsystem processes the input and returns output to the simulation, all the while fully "aware" that it is running in a simulation mode, thus *emulating* its own operational capability
- Stimulation – the main computer provides *stimulation (input)* to an operational system in the same way that the system is designed to receive it. The operational system processes the stimulus in the usual way, totally unaware that it is participating in a simulation.

- Simulation – during simulation the operational system is disabled or otherwise disengaged and the embedded simulation takes over its function (in the simulated sense) for the duration of the simulated exercise. Thus the computer *simulates* the functions and output of the operational system.

For FCS, the communication layer connecting all the various subsystems will be the System of Systems Common Operating Environment (SoSCOE). SoSCOE will provide the input/output functions as well as numerous other capabilities. Below the communications layer is the Computer Generated Forces (CGF) (OneSAF Objective System (OOS)) layer, which has the ability to provide custom compositions to meet unique simulation needs. To support the exchange of information between SAF simulation "back ends", OOS has created a Simulation Object Runtime Database that will likely be accessible through SoSCOE to support information exchange between the SAF and FCS system.

Below the SAF are the models and databases needed within the simulation. Models of different resolutions will be required. SAF models typically have low fidelity while engineering models provide higher resolution at the expensive of much greater computational demand. A major design issue has been identifying the level of model and terrain database fidelity required for training. Ideally an architecture should maximize reuse among the various simulator compositions required.

After evaluating the issues among the FCS training components, the need for a streamlined simulation engine became apparent. This streamlined simulation would operate under the restricted computational resources allocated for embedded systems to support the emerging demands of the FCS and other simulator developers such as PM CATT. The ECATT-MR STO has undertaken the development of a streamlined simulation engine to meet these needs. The focus of this engine is a merger of simulation "ownship" capabilities with the CGF.

## PAST EFFORTS

Several related efforts were reviewed in preparation for this research. SAIC's internal research effort called Multi-Modal (M2) SAF, was based on the Close Combat Tactical Trainer SAF. M2 SAF interfaced directly with CCTT SAF entities such that the physical models in the SAF could be controlled through an external interface, such as a joystick, as opposed to the driver, gunner and commander behavioral models residing in the SAF. This allows the operator to override the normal SAF entity behavior and directly control the entity.

Another related effort was the Advanced Concepts Reconfigurable Trainer which was developed by the UAMBL at Fort Knox. The design of this simulator was based on the use of modified ModSAF and later OneSAF Testbed Baseline (OTB) library files as the simulation engine. Additional libraries were added to provide simulator specific functions such as reading input from controls and handles, IG interfaces, and aural cues. The developers found they had to modify the weapon slew rates, fire control, weapon firing and vehicle dynamics since the SAF implementations of these models were all inappropriate for manned simulators.

Other related findings include the design of the Close Combat Tactical Trainer in which many of the common components, such as the Environmental and System Services are used by both the SAF and the manned simulators. This module reuse resulted in lowered code development costs while providing better configuration management and interoperability between CCTT networked simulations. In addition, the utilization rates of Manned Simulator hosts are typically a fraction of the SAF host loading during any given exercise.

Based on the success of these related efforts we determined that a SAF-based simulation engine to support embedded training was feasible.

## INITIAL DEVELOPMENT

During the first year of the ECATT-MR STO, a proof of concept demonstration of a SAF-based simulation engine was undertaken. This effort was called the Single Host Embedded Simulation System Architecture Proof of Concept. In addition to investigating the possibility of an embedded architecture with the CGF as executive of the system, this work also explored issues associated with running the entire system and associated processes on a single host. The footprint of the single host system was in line with limitations listed in the Stryker RFP described above. OTB was used in this proof of concept. The basic architecture can be seen in Figure 2. The system design followed the "A-Kit" - "B-Kit" concept used by the Tank Automotive Research, Development and Engineering Center (TARDEC) for their Crew integration and Automation Advanced Technology Demonstration embedded simulation software. In this design, the A-Kit typically includes the controls that are unique to the vehicle and the B-Kit includes the common simulation items such as the SAF.

This logical division arises naturally as a result of the appended nature of most embedded training systems. One advantage of this architecture is a high degree of decoupling between the A-Kit and the B-Kit. The two "kits" communicate using a well defined interface, which consists of a set of A-Kit/B-Kit messages. Thus, any set of controls (be they COTS joysticks or onboard VETRONICS) that can support the message set could control the simulation.

Another design feature borrowed from TARDEC's prior work was the use of a Process Interface Unit (PIU). The PIU is a software abstraction of the shared memory and message queue facilities present in the Linux operating system. Although the system could very well have been designed without a PIU, the decision was made to include it to minimize coupling between the A-Kit Interface and OTB. There were other design issues as well but they are beyond the scope of this document.

Finally, a major feature of the architecture was its use of VMWare Workstation to support two operating systems running concurrently on a single physical computing platform. This enabled the use state-of-the-art graphics hardware and drivers to support the IG Runtime on the Windows host OS, while at the same time allowing us to run OTB in its preferred Linux environment. The Linux OS was run as the guest OS within the VMWare Workstation. The embedded simulation was initialized from the SAF application. To shift an entity from constructive control (SAF behaviors) to external simulation control (tele-operations), an *external control* command was added to the task matrix of a selected SAF entity. Upon execution of the external control command, any behaviors currently executing for the entity were preempted and the entity began responding to inputs from the joystick, steering wheel, and foot pedal input devices. This effectively transforms the SAF (at least from the perspective of the selected entity) into a virtual simulator. To avoid contention for the input devices, the behavior was designed to enforce the constraint that no more than one entity could execute the external control behavior at a given time.

If the external control behavior was subsequently terminated the execution reverted back to constructive operation and the previously interrupted activity would resume. This ability to seamlessly transition back and forth between constructive and virtual operation was very well received.

A separate process, the A-Kit software, was dedicated to the acceptance of inputs from the external devices. The A-Kit communicated the inputs to the A-Kit

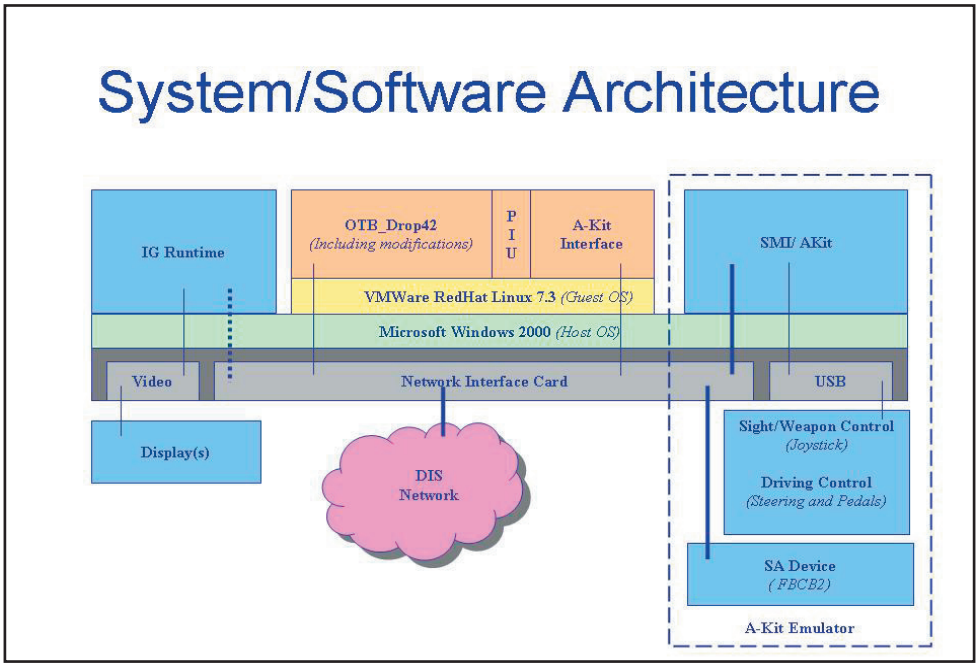


**Figure 2.** Single Host Architecture

interface via TCP messages with application-level semantics. The A-Kit interface communicated these to the SAF via the PIU, whereupon the SAF scaled the inputs based on the physical parameters of the controlled vehicle. During virtual control, dead reckoning is turned off so that entity state information can be sent more frequently, providing a significantly smoother visualization. In addition, the Probability of Hit (PH) function is turned off since this activity is now deterministic based on the operator's control and aim. When the weapon is fired a request is sent to the image generator to determine whether or not a hit would occur. As previously implied, the library restores both dead reckoning and gunnery PH calculations upon termination of the external control behavior.

This system was demonstrated at the Inter-Service Industry Training, Simulation, Education Conference (I/ITSEC) 2003. It was also demonstrated to the FCS training Lead Systems Integrator (LSI) and Program Manager OneSAF as a possible embedded simulation engine. Following this demonstration, a decision was made to redirect the development to use early drops of the OneSAF Objective Systems (OOS) to drive an Infantry Carrier Vehicle (ICV) simulator (see Figure 3) as the proof of concept. This OOS-based system, known as One Simulation (OneSIM), would better align the effort and increase the technology transition possibilities.

## PROOF OF CONCEPT

Development of the OneSIM proof of concept will focus on merging simulation ownship models into the OOS Architecture Framework. Figure 4 shows the OneSAF Product Line Architecture Framework. Of most interest in this effort is the Simulation Core which includes physical and environmental models. Also of interest are the middleware services, since one or more these services will be used as the interface to the vehicle's operational systems.

The initial plans are to drive the basic OOS-provided physical models with the ICV simulator's Soldier Machine Interface (SMI), illustrated in Figure 3. Most of the presently available OOS models are very low fidelity and are not useful for virtual simulation applications such as vehicle mobility. However, they could be used in some limited training situations and the exercise will provide us with a quick look at any issues in driving the ICV simulation with OOS. Once baselining with the low fidelity models is complete the effort will shift to using one or more models of intermediate fidelity. In particular we'll be

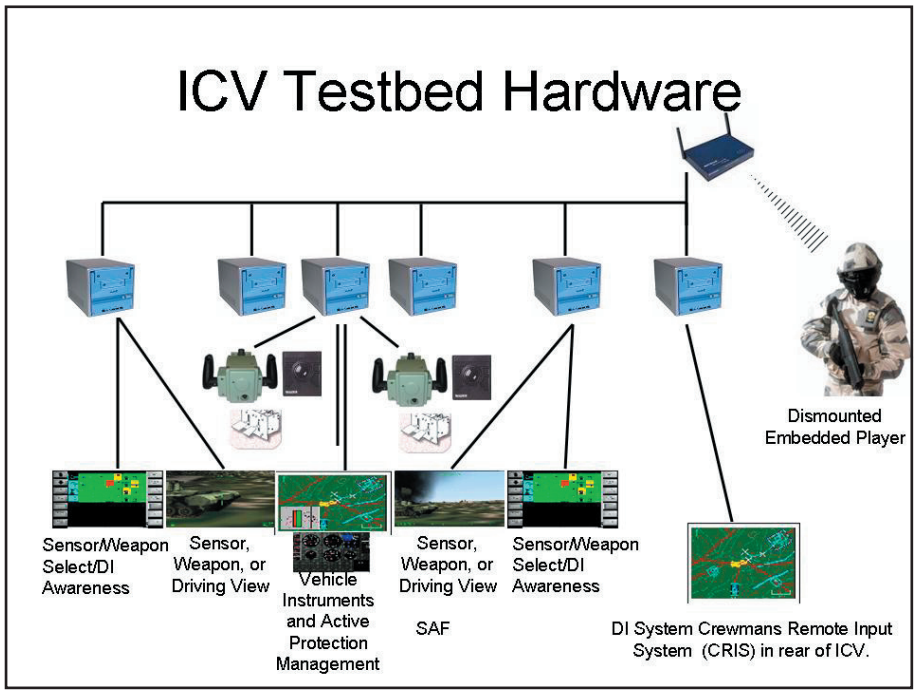


**Figure 3.** ICV Soldier Machine Interface

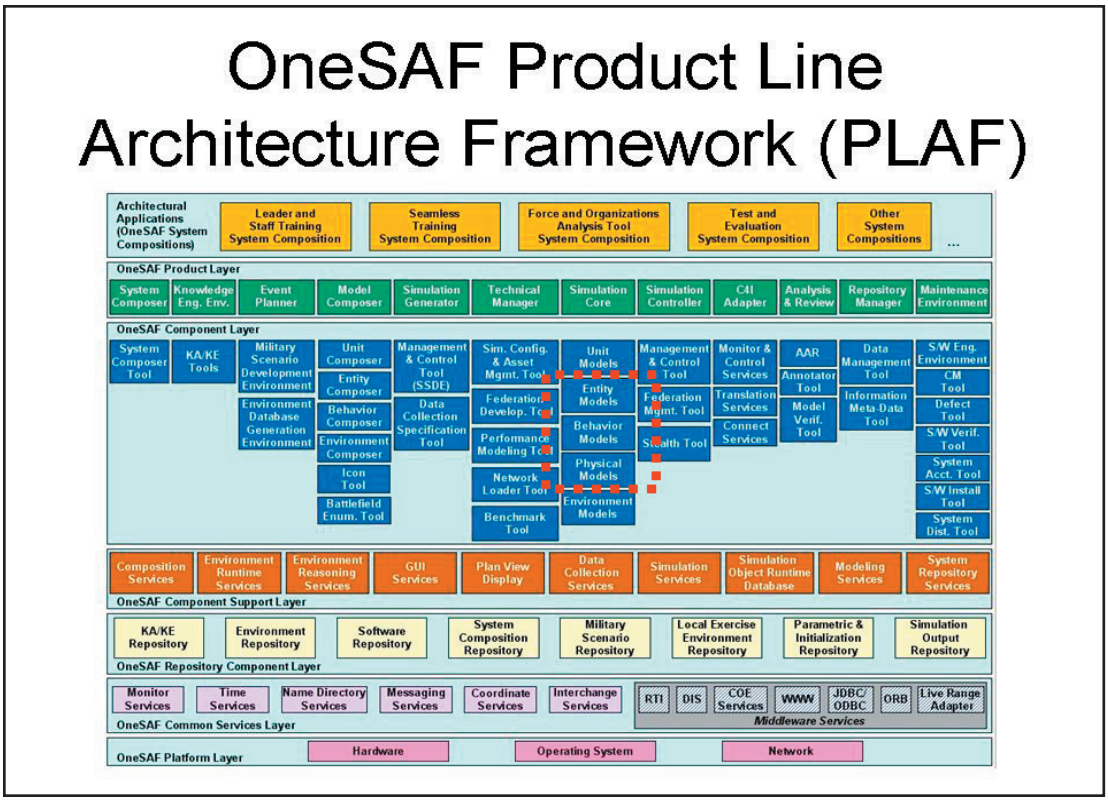


**Figure 4.** OneSAF Product Line Architecture Framework (PLAF)

interfacing to the CCTT SAF mobility model, which is scheduled for porting to OOS at about the time we intend to begin this phase. Finally, we intend to develop and integrate one or more high-fidelity models. The first of these will be the CCTT ballistic flyout model. Time permitting we will also integrate a high-fidelity mobility model such as the TARDEC mobility model in use at IST. The ultimate goal, of course, is to prove that OOS is a viable framework for supporting ownship simulations for embedded training. Once we have a critical mass of externally controllable models (of varying fidelity) in OOS, we can pick and choose from them to create appropriate compositions for a variety of training needs.

## DIFFERENCES BETWEEN OOS AND OTB IN SUPPORTING THE PROTOTYPE

There will be differences between in an OTB- and OOS-based simulation design. The OTB design utilized a 15Hz physical model to receive the PIU messages depicting the controller positioning via polling. The messages were then decoded and were turned into deterministic values based on the physical model limits that are supplied to the hull, gun and turret physical models. This was toggled using an enabling behavior. In OOS the data input will be received through the middleware services, which will be initially prototyped in DIS, but will ultimately be the SoSCOE. The input is then translated into an OOS event and, by way of the SORD, delivered to the appropriate receivers. Upon receipt of the OOS mobility input message, the driver model generates values and passes them to the mobility capability. The mobility capability is an agent for the physical hull model that has been selected. The gunnery input message will behave in a similar fashion, using the turret agent and gun agent.

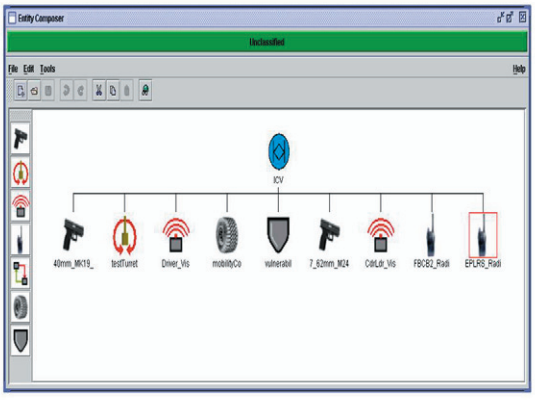


**Figure 5.** OOS Composition

## SOLDIER MACHINE INTERFACE

A virtual simulation of a vehicle must have a means to receive inputs from the user. Minimally, for the vehicle's driver, these inputs consist of steering, braking, throttle, and gear selection. For desktop trainers, the input devices are often simple COTS hardware solutions including: joysticks, yokes, steering wheel/pedal combinations or other facsimiles of driver controls. In the world of embedded simulation/ training the input devices are (will be) the actual input devices used to control the vehicle during real driving operations.

Since the ultimate goal of the OneSIM architecture is to support embedded training in FCS vehicles, and because the operational subsystems of FCS vehicles will communicate largely via the SoSCOE, the ideal solution for our testbed would have been for driving inputs generated from physical input devices to be communicated to OOS via SoSCOE. Using such an approach would greatly facilitate future integration into FCS vehicles. Unfortunately, development of SoSCOE has not progressed sufficiently for this to be practical within the time constraints of the STO. For this reason we have elected to use a SoSCOE surrogate.

The OneSAF program office is integrating OOS into the SoSCOE. This research will concentrate on using enhancing the SORD interface to communicate between the vehicle controls and OOS. Once this is accomplished and OOS has been integrated onto the SoSCOE, there should be little effort required for OneSIM to communicate with the OOS SORD through SoSCOE

Examination of the OOS PLAF revealed that interfaces to DIS, HLA, and the Common Operating Environment (COE) are (or will be) contained in the middleware services subset of the OneSAF Common Services Layer. Messages received from external sources (DIS PDUs, HLA Objects and Interactions, SoSCOE Messages, etc.) are processed by the appropriate middleware service and routed to the SORD as OOS Objects or Events. Consumers (in our case, Entity Models and Controllers) of the various OOS Objects or Events are concerned only with the Events or Object updates themselves -- not which middleware service was responsible for generating them. In this way the OOS internals are shielded by the abstraction layer provided by the SORD. Communication with OOS, therefore, can be done through HLA, DIS, or directly with the SORD with no modification to OOS. To take advantage of the abstraction provided by the SORD, we have elected to use the DIS middleware layer as the connection point for our input devices.
A dedicated (separate from OOS) process reads the USB input devices and transmits driver and gunner control information onto the DIS network using SetData PDUs (Figure 6). The DIS Interoperability Middleware Services decode the SetData PDU and create appropriate OOS Events.

In theory, when SoSCOE matures sufficiently to be useable, we can adapt the testbed to support it by equipping the COE middleware service with translators that will convert the raw SoSCOE driver and gunner inputs into similar (if not identical) OOS Events.

## RUNTIME ISSUES

For a virtual simulator the simulation runtime is typically optimized for a single entity and near real-time performance is expected. As more and more demands are made upon the system, every effort is made to ensure that time critical code is executed at the proper time. This ensures realism in the simulation visuals and helps to prevent simulation sickness. Furthermore, certain high-fidelity models require a high tick rate with a tightly constrained tick interval, to function properly

For constructive simulations, such as a SAF system, the runtime is optimized for large numbers of entities. The models themselves are often simplistic (relative to those used in virtual simulations) and rarely have critical time constraints. Although scheduling is still very important in these systems, if the runtime is over-taxed, the simulation will not catastrophically fail, but will fall behind in terms of simulation time. Should the load on the CPU decrease the constructive simulation has the possibility of recovering and "catching up".

The OneSAF Objective System is by definition a "SAF" system. As such its runtime is (will be) optimized, *as it should be*, for constructive simulations. To support virtual simulation the OneSAF runtime will almost certainly need to be extended to support a special mode that will give priority to those threads and events that are time critical.

Beyond the design of the simulation runtime infrastructure, OneSAF Objective System has the added challenge of being written in the Java programming language and therefore running inside of a Java Virtual Machine. One of Java's biggest strengths, automatic memory management, is also often regarded as one of its biggest weaknesses. In traditional programming

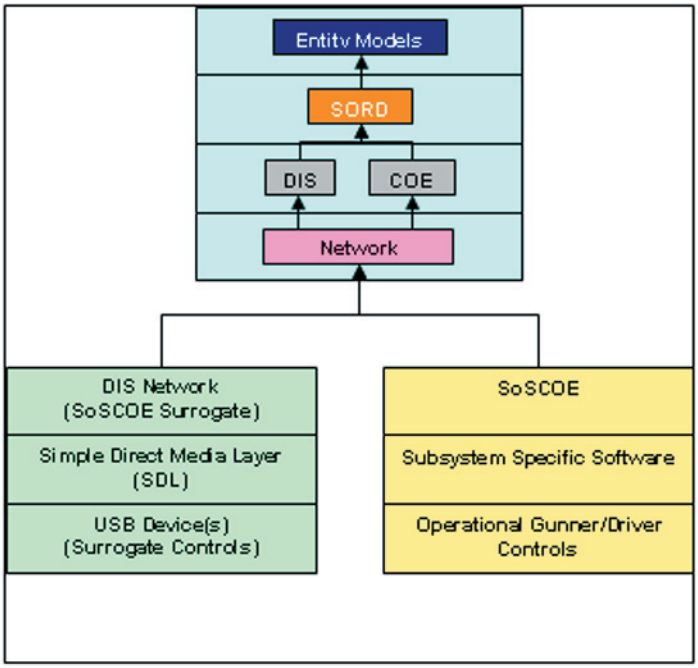


**Figure 6.** OOS Ownship Control Device Interface Architecture

languages, memory is explicitly managed by the programmer. If memory is allocated from the heap by the programmer, then it is the programmer's responsibility to de-allocate that memory when it is no longer needed. In Java, programmers are free to allocate memory whenever and wherever they need with very little concern for explicitly reclaiming it. Unfortunately, this luxury is not free. To support this, the Java virtual machine must keep track of unreachable objects present in the heap and periodically dispose of them, thereby reclaiming the memory they occupied. This is called garbage collection. When the garbage collector runs, normal program execution is interrupted and time-sensitive programs, especially long running ones, are often impacted noticeably. Until recently programmers have used various tricks, such as object pooling, to minimize creation of objects on the heap and thereby reducing the frequency of the garbage collection.

In recent releases of Java, more advanced garbage collection algorithms have been introduced to the extent that the experts in this field now recommend against taking such measures. One promising approach is the so-called "concurrent garbage collection" mode, in which garbage collection proceeds concurrently with the program's normal execution. In this approach, overall system performance is reduced by a small fraction, but relatively long pauses resulting from garbage collection are essentially eliminated. Not surprisingly, this approach is most effective on a multiprocessor machine, where one of the processors can be dedicated to the garbage collection thread. Despite the advances in garbage collection technology, standard Java will probably continue to come up short when it comes to supporting virtual simulation. Fortunately the Real-Time for Java Expert Group, under the Java Community Process, has developed a specification for additions to the Java Platform that will allow it to be used for real-time applications. Details are available at: http://www.rtj.org/. As part of this research we are testing the performance limits imposed on the standard Java Virtual Machine imposed by its garbage collection. As limitations are defined, we will begin to experiment with the Reference Implementation of the Real-Time Specification for Java.

## TRANSITION FROM CONSTRUCTIVE TO VIRTUAL CONTROL

The original OTB-based proof of concept testbed used a task matrix "external control" command to transition entity from constructive operation to human-in-the-loop control. The command was entered via the graphical user interface of OTB. In the current OOS-based testbed, the transition is made by sending a special "toggle ownship" message from the driver controls. Upon receipt of the toggle ownship message, an

appropriately equipped entity composition will begin responding to "mobility input" events. To effect the change, the Mobility Controller un-registers all behavior triggers and begins responding to Mobility Input events originating from the input devices. For efficiency, the behavior triggers are de-registered, as opposed to being ignored. Upon receipt of the next toggle ownship the process is reversed and the entity resumes execution of any CGF behavior(s) that were previously interrupted. Thus the seamless transition achieved with the OTB implementation is preserved.

## IITSEC 04 DEMOSTRATION

Figure 7 presents a high-level diagram of the planned I/ITSEC 2004 OOS-controlled ICV simulator. A modified version of OOS will provide the Weapon and Mobility Agents to control the dynamics of the ICV. The demonstration will illustrate two ownship framework use cases. One is the mobility agent that will use CGF models being affected by the Human in the Loop (HITL) of the ICV. The other is the weapon agent which will illustrate how a high fidelity vehicle specific model that was re-engineered into OOS can support HITL interactions with the ICV. In this case the model will be based on the Close Combat Tactical Trainer (CCTT) manned simulator weapon fly out. In both cases existing OOS behavioral frameworks which typically mimic human behaviors will be overridden by OneSIM Agents that interface directly to the basic physical models in OOS. The focus of this demonstration is to identify areas of the OneSAF framework that will need to be extended to support ownship modeling as well as to show OOS's utility as a complete simulation modeling tool. The communications in the demonstration will be via DIS and a direct interface to the ICV USB Interfaces. The ICV will send SetData PDUs describing positions of its controls. OOS will send EntityState PDUs describing location, orientation, etc., of the ICV.

## PLANNED FOLLOW ON

The FY04 work focused on getting OneSIM operational on an ICV simulator. The next year's focus will be on extending the system to support dismounted infantry and unmanned simulations (Figure 8). Using OneSIM to support dismounted soldier embedded simulation presents a significant challenge in that the computer resources that can be carried or worn by a soldier are severely limited. The dismounted soldier OneSIM composition will require either networking supporting systems to share the CGF computational load or a scaled down OneSIM version.

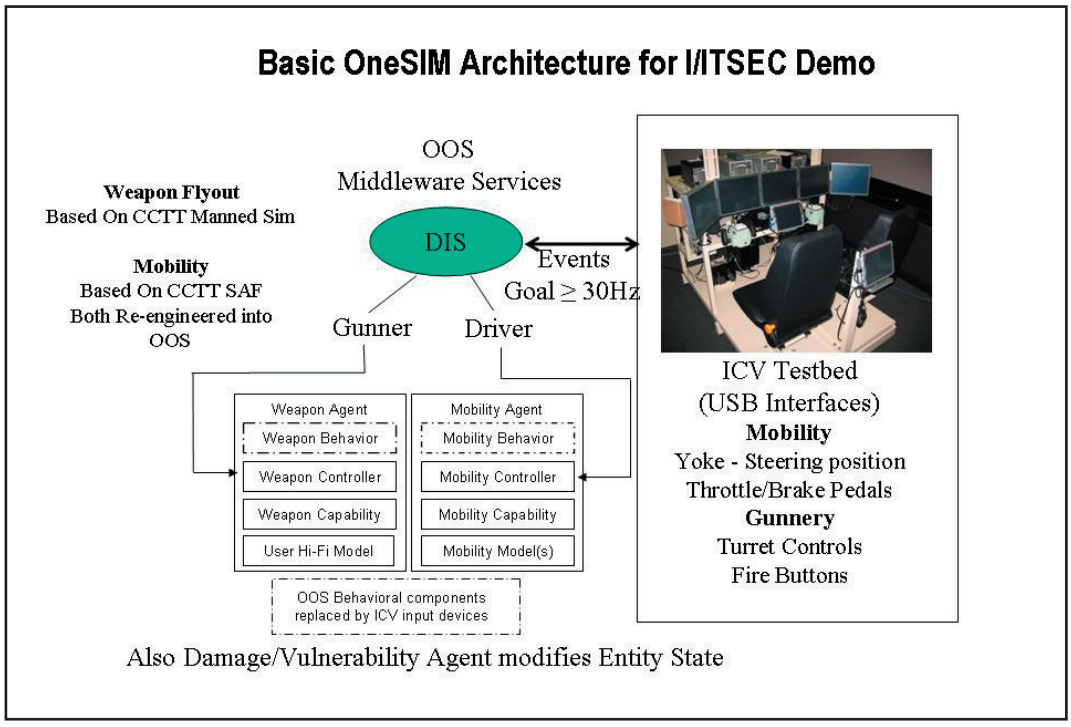


Figure 7: I/ITSEC Demonstration of OOS Controlled Simulator

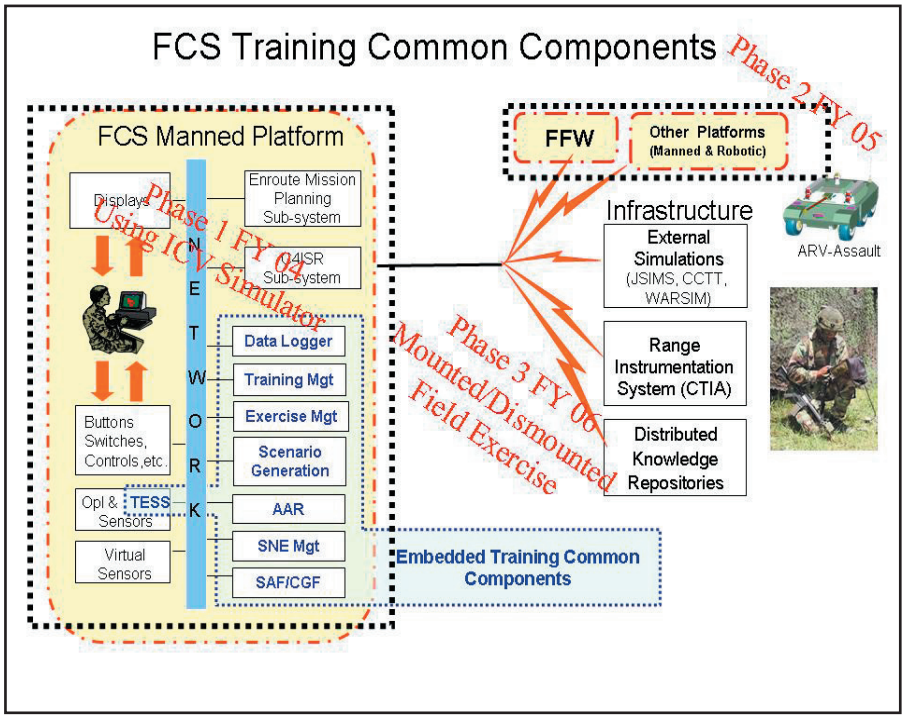


**Figure 8**. Phased Plan for OneSIM Development

Embedded robotic control capability is another promising utilization of the OneSIM engine. Typically, a robotics control trainee has to practice UAV or UGV operations without the equipment. Providing simulation directly from the CGF interfaced with the tele-operation or other controls the operator may use is a logical choice. Several SAF based Robotic Operator control units have been designed and used for unmanned experimentation but none are based on OOS.

In the final year of the STO a culminating event is planned. This event will demonstrate a mounted and dismounted field capable embedded training system that will bring together several STO technologies. In addition to OneSIM, the culminating event will include mission rehearsal, distributed after action review and intelligent tutoring.

Common Gunnery Architecture (CGA) is an ongoing effort of the Program Executive Officer Simulation, Training and Instrumentation (PEO STRI) seeking to create a product line architecture specification that will support all of the various gunnery simulators in use by the US Army. The program goals shown in Figure 9 are complementary to OneSIM. The initial phase of CGA will determine requirements for all the gunnery training simulators. These requirements will be captured in the same PLAS used in the development of OneSAF to track the architectural specification for a product line of systems. Both CGA and OneSIM are approaching the issue from a requirements perspective. OneSIM will consider the issues for using OOS as the core of the CGA architecture.

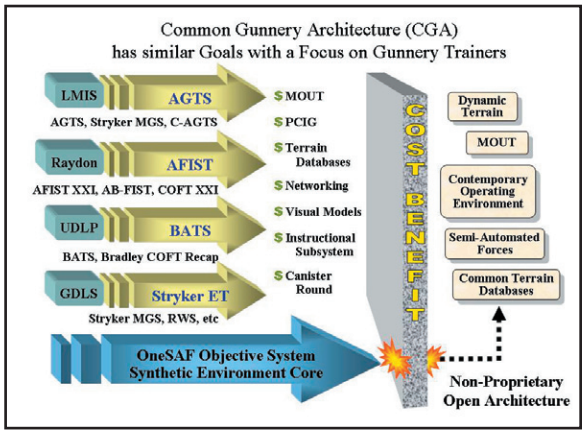


**Figure 9.** Common Gunnery Training Architecture

## CONCLUSION

In conclusion, the development of OneSIM capability in OOS has numerous challenges but potentially great payoff. Given the interest and needs of FCS and PM OneSAF, the ECATT/MR team will continue their work toward a successful demonstration of OneSIM and transition of the technology.

## REFERENCES

SAIC Corporation, *Final Scientific and Technical Report for Embedded Systems, 2003,* CDRL Number A002

Boys, Randy, *Embedded Training via Operational Modes of Vehicle Electronics (VETRONICS),* Presentation at FCS Training IPT #7 3/26/04 FCS Ground Sensor Integrator.