"System of Systems" Approach for the Development of Next Generation Modular Simulation-Based Training Systems

Cory Kumm
Bohemia Interactive Simulations Inc.
Orlando, FL
Cory.kumm@bisimulations.com

John Burwell
Bohemia Interactive Simulations Inc.
Washington DC
john.burwell@bisimulations.com

ABSTRACT

Within the defense sector of the modeling & simulation industry, most simulation-based training systems are comprised of a mixture of heterogeneous technologies that are highly challenging and expensive to develop, maintain and network. Some systems consist of custom-developed solutions and/or exploit open source technologies. Others rely on commercial, off-the-shelf (COTS) and government off-the-shelf (GOTS) tools. The use of closed, proprietary technologies, data and tools can have great benefit, but adds complexity, cost and risk. Even though a variety of industry standards for communication protocols and datasets exist today for various components of simulation solutions, no formalized intra-organizational "system of systems" approach exists for simulation solution development. As such, simulation software developers are often forced to use ad-hoc development and non-standard methods to interface technologies, especially when working with proprietary components. These ad-hoc methods increase development time and add cost and risk that could be reduced if simulation systems were developed using a common set of industry standards and methods. Where organizations like the Simulation Interoperability Standards Organization (SISO) are helpful, major gaps remain. This paper discusses why the gaps occur and methods that have been used to eliminate them.

ABOUT THE AUTHORS

John Burwell has 30 years of experience developing innovative and disruptive technologies used for simulation and training. With a focus on simulation and virtual environments, John has supported companies developing image generation systems, virtual worlds, serious games, computer graphics, and geospatial imaging. John has a bachelor's degree in Electrical Engineering and Computer Science from the University of Colorado and an MBA in International Business from Thunderbird in Glendale, Arizona.

Cory Kumm's career spans over 20 years and several industries including entertainment, aerospace, defense where he has contributed to a variety of technology awards and patents. Cory established and ran a division at Havok with responsibility for adapting game-based technologies to the needs of the simulation community. Prior to Havok, Cory worked on artificial intelligence projects at MASA Group. From 2002-2009 he held product management roles with Biographic, Engenuity & Presagis for COTs technologies. Cory is an alumnus of Concordia University having studied computer science and is a certified Product Management & Marketing Professional.

INTRODUCTION

Traditionally, simulation system developers relied on fully custom hardware and software to build training systems. As commercial off-the-shelf (COTS), Government off-the-shelf (GOTS) and Open Source technologies became more capable, solutions became more modular and integration and interfaces gained importance. Developers were faced with challenges stemming from a lack of formalized industry standards and a need to more effectively design and develop ever more complex systems. Standards organizations like the Simulation Interoperability Standards Organization (SISO) were set up to help, and have been invaluable in providing a venue to create and manage standards, but major gaps in the standards to support interoperability remain. In fact, gaps are widening as commercial technologies from the video game industry must be leveraged to address defense budget challenges. Today, with billions of dollars being invested by the video game industry in virtual and augmented reality technologies, an opportunity exists for the simulation industry if appropriate to leverage this massive investment however interoperability questions remain.

For many years Bohemia Interactive Simulations (BISim) has operated by taking video game technology and adapting it for simulation use. In [2013], BISim began working to establish a framework for simulation development that supports the integration of a variety of traditional government simulation and emerging commercial technologies to create powerful training solutions. The framework allows for the creation of services or plugins that interoperate through a standard, well-defined application program interface (API). Based on ongoing program requirements, BISim has developed an internal set of standards and applied them to all developments under a new modular simulation development framework called Gears. This framework uses a set of common APIs, which developers can use to create collaborative systems that are highly performant, easy to debug, modular by design, and less costly to develop and maintain as compared with other methods. Over the past few years, the APIs have matured and we now believe the framework is ready to be exposed to the greater simulation community in the form of a standards-based, system of systems development approach for the modeling and simulation industry.

BISim wants to share our findings and collaborate with industry to foster further development and potentially establish a new formal, system-of-systems development standard. It is our view that for the industry to provide more “just-in-time” production capability of sophisticated simulations, the industry’s system integrators and software vendors will need to work more closely on these paradigms, sharing more openly than has historically been the case. In this paper we will outline the basis of the proposed modular simulation development framework and describe real projects where we have successfully used Gears to develop interoperable technology that links legacy simulation systems with emerging commercial gaming technologies and other third-party applications. We will identify the industry challenges faced, share solution concepts, and discuss our rationale for the system of systems approach. Finally, we’ll provide our view on next steps of how this capability could become an open standard with the simulation industry benefiting and contributing to further development.

INDUSTRY CHALLENGES

The defense industry continues to express a desire to work with common open and modular architectures that are not limited by proprietary software to modernize and keep pace with new technologies. The United States Air Force, for example, has 46 complex simulation systems that average 11 configurations, which amounts to 506 unique systems used in multirole player training. [1] These systems must connect to form a common or joint exercise and many systems have been created using proprietary technology developed and updated by many integrators and subcontractors over the span of the last 20 years. These systems are all custom developments with architectures that require support from the original vendors – even for small enhancements. [1] With major advancements in hardware and operating systems over a 10+ year span, small enhancements can end up requiring a complete overhaul and re-write of the system. Enhancements may turn into full maintenance programs that require a complex series of requests for funding, proposal and program award competitions, and, finally, years of actual development and deployment of the capability. The duration of the entire process is roughly 3 to 5 years from start to finish which is completely untenable for the sort of operations tempo faced by the DoD today. Understandably, the Air Force is asking industry for a strategy to facilitate accelerated development and updates of simulator systems including, but not limited to, upgrades of the 46 disparate simulation systems. [1] The desired modular development capability being requested by the Air Force has been named SCARS or Simulator Common Architecture Requirements and Standards. SCARS describes a desire for a common open architecture to facilitate rapid development and avoid pitfalls of being ‘locked’ into the use of proprietary technology. SCARS today is a concept, and the Air Force has asked industry for a response on how industry can help modernize and accelerate simulation-based Air Force training enhancements.

The U.S. Navy is also hard at work trying to solve a similar problem with its Future Airborne Capability Environment Technical Standard Edition or FACE™. FACE is an effort coming out of the Naval Air Systems Command (NAVAIR) naval aviation systems command (PMA-205) to define a new set of open architecture standards for the development of new training systems. With the cost of software development for these systems rising exponentially and under pressure to reduce costs, reuse of software across training systems is essential. An open architecture, utilizing open standards at defined software interfaces is the desired future for training systems.

Standards that allow disparate systems to simply communicate with one another is only part of the challenge. Another challenge of common interoperable solutions relates to content and context. For example, there is the need for various forces to have a common “fair fight” where all entities operating as a joint force experience the same synthetic environment. Fair fight refers to having correlated, synthetic environment representations within a collective simulation. The U.S. Army is currently asking industry to develop concepts for a common, one-world synthetic terrain simulation capability comprised of open standards that various disparate simulation systems can leverage for training or analysis needs. The U.S. Army’s Synthetic Training Environment (STE) is designed to provide collective multi-echelon training and mission rehearsal capability for all training domains across the Army. [2] The common synthetic environment has the mandate to provide rapid training capability with minimal manpower required to develop geo-specific or real-world virtual geographic updates to the STE anywhere in the world. The intent is to have one common, validated source of information that can be used by all virtual, constructive and gaming training systems. The complex derived requirement is that all the various simulation systems such as those described earlier for the Air Force would need to support a common synthetic representation of the world.

Other considerations are reduced defense budgets, which lead to a reduction of live training and an increasing focus on simulation-based training, a need to modernize legacy training systems, and, most importantly, significant advances in computing and hardware technologies and simulation software, meaning simulation training is becoming more and more realistic and capable. Leveraging commercially developed technologies, particularly from the video game industry, is of increasing interest as billions of dollars in R&D are spent each year on videogame software advancements and some core elements of the technology can be re-used for training purposes. For example, emerging virtual reality technologies currently in development for entertainment applications are well suited for many part task training programs. Since commercial video game technologies do not follow standards as they are often very bespoke developments incorporating these technologies into training programs can be particularly challenging.

As modernization efforts begin to consider the latest commercial hardware and software solutions, an ever-growing plethora of new technologies and devices coming from the commercial sector are being considered. These include Virtual Reality (VR) devices such as the helmet mounted displays (HMD)s from Oculus Rift and HTC Vive. These devices, coupled with modern, game-based rendering technology, should be able to support various human-in-the-loop training systems, replacing some of the larger more traditional (and far more expensive) dome- and collimated display-based training devices. The conundrum, however, is that the defense acquisition cycle is far too slow to keep pace with the commercial sector’s advancements. One Defense industry program refresh cycle requiring 3 to 5 years can represent several new generations of commercial technologies. Consumer VR headsets, for example, will have released 3 to 5 new versions and a myriad of driver updates during a 3- to 5-year period, and within the same time frame early versions of these devices will have become obsolete and unsupportable. For example, Oculus Rift’s evolution is illustrated below:

- 2013: the developer version of Oculus Rift called DK1 was released with a combined resolution of 1280×800 and initial PC hardware requirements.
- 2014: the developer version DK2 was released with increased HD (1920x1080) resolution support and increased hardware requirements. DK1 was made obsolete.
- 2016: the commercial version of Oculus Rift started shipping, with more stringent hardware and driver requirements than the DK2 version.
- 2017: Oculus Touch new hand controllers designed to work with the Rift are introduced, again new drivers and development integration requirements for the new hardware.

Developers' challenges are many: modernizing and keeping systems up to date; integration of a variety of heterogeneous simulations often from a variety of sources; development and maintenance of a common world representation; and, reducing costs via leveraging new capabilities such as virtual reality. To provide the best training capability possible for the warfighter, we need to constantly evolve capabilities and increase overall system fidelity including taking advantage of new technologies as they become available. We must also find ways of iteratively enhancing training and simulation capability faster and with more agility than before, which is why we need a common open standards-based, system-of-systems approach.

DEFINING COMPLEXITY AND ASSOCIATED INTERFACES

Bounding the approach to a system-of-systems architectural design effort requires an identification of the technologies, interfaces and various subsystems that must work together via a schema required to achieve the targeted solution sets. At the top level, solutions require core hardware and software elements that serve as the fundamental platform for a training device or simulator that potentially connects with other simulators and live assets to support collective training. Key hardware elements to consider are listed below:

Hardware Components	Elements
Personal Computer (PC)	CPU, GPU, Graphics card, system bus, I/O, Memory, Storage systems
Networks (LAN, WAN, Internet)	Physical layer, protocols, switches, I/O Devices
Display Systems	Monitors, Projectors, Domes, Collimated Displays, VR and AR HMDs
Headsets and Speakers	
Motion Systems	Cueing devices, 3DOF, 6DOF
Interface Devices	HOTAS, Joysticks, Steering wheels, pedals, instrumentation
Crew Stations	Cockpits, Shoot Houses, Classrooms

<u>Hardware Challenges</u>

Hardware components, especially CPUs, graphics cards, and displays, change frequently. Developing with this evolution in mind requires the use of middleware concepts for software design and for hardware suppliers to minimize any changes to the exposed interfaces for developers. A relevant example here would be OpenGL or DirectX. These interfaces evolve relatively slowly to ensure that software developed can function on legacy, current and future hardware for a commercially reasonable lifecycle, which is typically 5 years. Where 5 years is still not adequate for the typical lifecycle of a simulation platform, these interfaces tend to be very stable and enduring in the commercial sector. Key software elements to consider are listed below:

Software Components	Elements
Operating Systems	Windows, Linux, VxWorks
Application Code	As needed to support subsystems
Network Communication Protocols	DDS, CIGI, DIS/HLA, TENA, Link 16
Physics Model Representation	Physx, Bespoke
Industry Standards	CIGI, OSVR, CDB, DIS/HLA, NPSI, OpenStreetMaps, OpenFlight
Databases	GIS data, DEMs, Shape Files, Imagery, 3D Model Data
Graphics APIs	OpenGL, DirectX

Software Challenges

Software-based technologies are often less stable, mostly custom and vary substantially from vendor to vendor in terms of openness, longevity and compatibility as a component of a larger solution. Here, over the years, development practices have evolved from monolithic applications (stove pipe) to more modern, componentized middleware approaches. For the purposes of this paper, we will focus on the interfaces that are intended to provide an ability for integrators to build "systems of systems".

APPLICATIONS

Running on the hardware platform are a series of applications that must work together to provide functionality which replicates man-made and natural systems. Some of these applications are based on 20-year-old validated programs that are ideally reused. Others applications are based on cutting-edge technologies coming out of the commercial video game industry. Ideally, solutions can be developed using the best-of-breed applications, regardless of origin. Creating training solutions for military application requires a the right mixture of realistic virtual environments, realistic weapon system simulations, intuitive interfaces and assessment capabilities to monitor human performance. The exact feature and performance requirements vary by use case. Where the list is far from comprehensive the following is a list of example functionalities required:

- User interface
- Virtual environment
- Avionics and weapon system simulation
- Immersive display
- Crew station
- Vehicle simulation
- Radio communication and sound capabilities
- Assessment and scoring capabilities
- Networked, multi-user operation
- Computer generated forces

As you can see from the above list, many hardware and software components are required to develop these systems. To add further complexity, even the defense industry's own “standards” continually evolve and often have been developed with specific implementations in mind. Each category of device or software component has unique challenges. The goal is to define an optimal approach where this sort of disparate applications can work together on a designated platform.

POTENTIAL APPROACHES

An approach promoted today around the DoD to address the development challenges is to avoid the use of any proprietary technologies and in some cases to only use Open Source solutions. One risk in this approach is that there is a limited market for most of the technologies needed to develop high-performance simulations so without leveraging proprietary software that is continuously updated and maintained, solutions will be sub-optimized or require significant and costly custom development. Open Source solutions often lack a custodian meaning that no one organization is motivated to maintain or update these solutions. Hardware and operating system evolutions then become the responsibility of the DoD or organizations using these technologies in end applications which means the long-term lifecycle costs must be considered. These costs are often far more than the cost to use COTs technologies and may deliver less capability. Open Source technologies are ideal for some applications, but with complex simulations, a mix of proprietary and nonproprietary solutions will be much more effective. Another major risk is simply commercial - if companies are not allowed to develop their own IP, they cannot be profitable and, in turn, can not hire top quality developers and therefore the best talent will go elsewhere to other industries.

Supporting the needs expressed by industry for openness, modularity, ease of integration and interoperability of heterogeneous simulation systems developed by various companies is very challenging - extremely long platform lifecycles; current software “open standards” in use overlapping in functionality with other similar components (having grown from initial uses into other areas); and standards overloaded with features for convenience. As these software components grew over the last couple of decades, most were built with specific applications in mind and preconceived notions of use heavily embedded into these components. The real challenge now is how to adapt 20+ years of software development to the desired open and modular future the Government and industry is asking for. We believe the following factors need consideration:

- Extensibility and futureproofing
- Ease of use, development & integration
- High performance
- Reusability of components
- Iterative development
- Backward and forwards compatibility
- Support for multiple programming languages, with no assumptions on how components will be developed
- Integration with external systems or 3rd party technology
- Demand for non-proprietary systems

Certainly, one of the most important elements in an industry that requires 20+ years of capability sustainment is futureproofing. Here we should consider how the commercial sector has succeeded in providing some level of stability. DirectX & OpenGL both maintain ~5 years of stable APIs that allow developers to support legacy, current or forthcoming hardware **without** customization of software developed upon these APIs. Other examples include eBay, Google, Netflix and Amazon all of which evolved from monolithic applications that were difficult to develop and maintain into systems called “Ecosystems of Microservices”. [8][9][10] The evolution from monolithic to microservice is actually very similar to the current transition the US DoD is asking for. In the commercial sector, industry giants needed to be able to vastly expand their capabilities supporting both internal development and external independent development all while maintaining backwards and forwards compatibility knowing that their own platforms would evolve and improve over time.

Ease of use, development and integration for these companies was facilitated by a new approach of breaking down complexity into two layers. The first layer is the common, stable and as much as possible unchanging external facing macro level APIs (OpenGL, DirectX instructions, for example), and secondly a micro or component service API that is developed where constant evolution and improvements can take place all while maintaining compatibility with existing components.

Macro- or external-facing service-based APIs allow for a common description of all the interfaces or services that various components will leverage. The micro or component APIs are relatively small and easy to understand and develop. A component approach makes building solutions simpler and more effective as only the new portions of the solution would require development and existing functionality remains intact. The philosophy is that each API, either service or component, can operate independently or be joined together to form larger functionality. If component APIs are developed to support the stable service APIs, development, deployment and interoperability can happen organically and keep legacy systems working while new capabilities are introduced.

While this approach has a variety of benefits, it also has a few drawbacks:

- Publish/Subscribe bidirectional communication needs to be built into components
- Service APIs need to be meticulously designed and developed for futureproofing
- Additional overhead exists with component-based systems
    - Additional memory required
    - Development time increased to develop robust interfaces

PROPOSED APPROACH

Considering how the commercial sector is approaching the complexity for system of systems development, and with an understanding of how the defense simulation community operates, BISim chose to develop a microservice hybrid known as Gears. Gears is an open standards-based, modular simulation development framework designed to meet requirements for ease of use, modularity, support of multiple programming languages and the ability to maintain backwards compatibility with legacy technologies. Gears uses an ‘API first’ design to facilitate the development of components that are modular, re-usable, and safe to refactor.

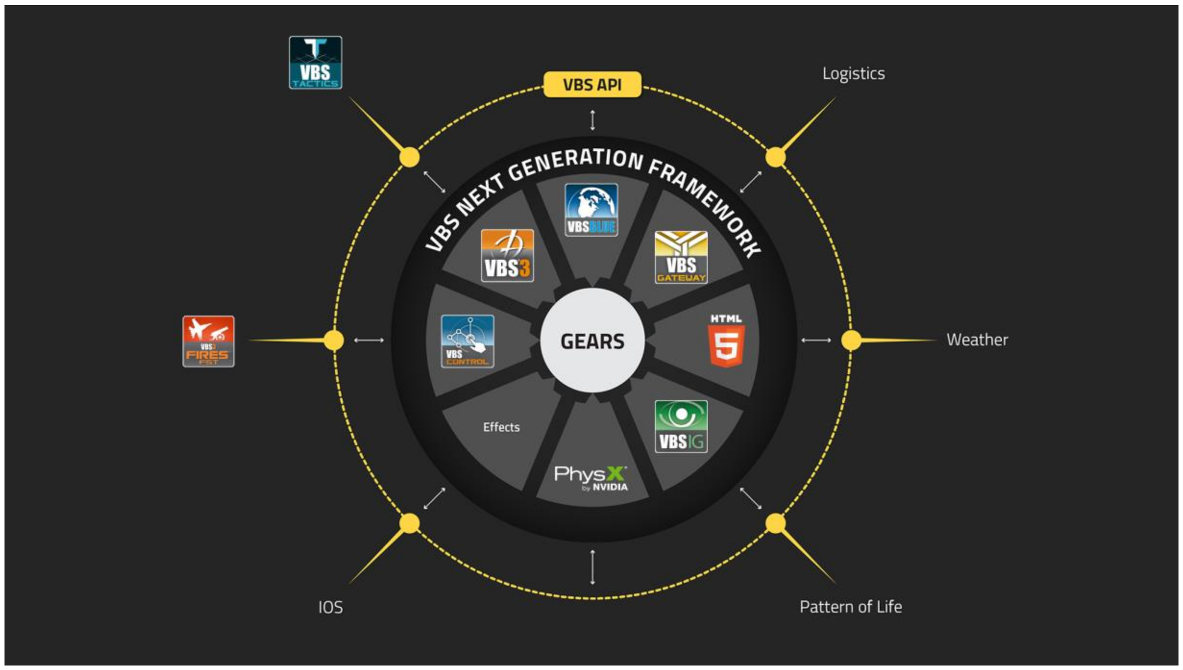

Figure 1: Gears Framework.

Figure 1 illustrates an approach that has external service APIs and internal component APIs. This approach uses the components to define larger features that then expose a service API layer. The components themselves can be volatile, may change at any time, and, more importantly, will keep pace with new capabilities added to a system. The “spokes” of the wheel are sets of functionality that can be used alone or combined to produce a complete application. Each spoke can be replaced without compromising the overall system or interface via the service API.

The Gears approach accomplishes the following:

- Organizes technology into components
- Exposes functionality through an API
- Follows standards for interoperability internally and with 3rd party products
- Combines groups of components to create solutions
- Allows components to access the functionality of other components directly

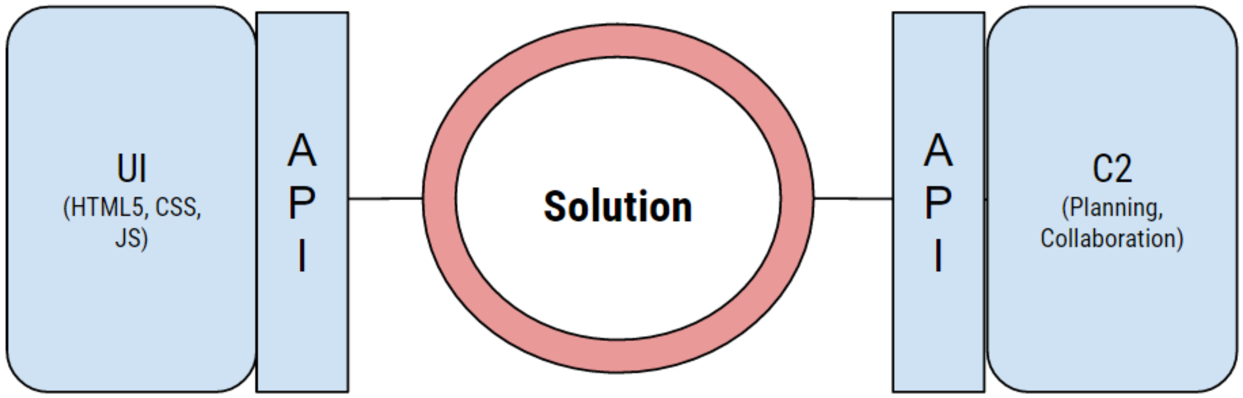

Figure 2: Example Implementation Using Gears Approach.

The best practices and guidelines of this approach include the following:

- Creating production-quality APIs
- Building components to implement those APIs
- Building products using those APIs and components
- Gears runtime that loads and executes components
- Gears Studio that accelerates building products using components by enforcing Gears rules
- Continuous build and integration infrastructure for build automation and distribution (including test automation)
- Central location to store components and aggregate components

Sample of Current and Planned APIs

Environment	
Terrain	Provides query information relating to the terrain such as height, normals, and surface type
Entity	Provides an efficient way for developers to create and control entities within a simulation
Scenario	Provides functionality for setting scenario attributes such as time of day or weather and allows hooking into scenario events such as mission start or mission end
Intersect	Performs intersection queries for objects and terrain within the simulation
Munition	Provides controls to create and display weapon fire effects and detonations.

Camera	Provides an efficient way for developers to create and control the cameras needed to provide custom display solutions with VBS IG.
Interoperability	
CIGI	Used to create a CIGI session between a source and destination host. Allows user to override behavior of all supported CIGI packets.
DIS	Communicating via DIS in a distributed simulation environment. Allows user to override behavior of all supported DIS packets.
HLA	Communicating via HLA in a distributed simulation environment. Allows user to override behavior of all supported HLA packets.
Systems	
Sensor	Provides mechanism for custom sensor implementations and is built upon the Render API
Laser	Provides mechanism to create and modify lasers in the simulation including visualization
UI	Provides ability to create, modify, and interact with UI elements. Used to set callbacks for UI actions.
Desktop Input	Provides easy access to input devices on the local machine such as mouse, keyboard, joystick, and head trackers
Symbology	Provides high level helper functions for drawing symbology in 2D and 3D. Built upon the 2D and 3D Render API
Utilities	
Transform	Provides access to object matrices and functionality for attachments.
Render	Low level API that allows hooking into rendering passes and events, and provides access to the graphics device. Allows custom rendering effects and used to implement higher level 2D and 3D APIs.
2D/3D Render	Provides high level helper functions for easier drawing and rendering in 2D and 3D. This is built on top of the Render API
Spatial	Provides mechanism for manipulating the spatial coordinates of objects in Geodetic or ECEF coordinates

Entity Manager	Provides functionality that can be used to modify entity properties. This can be used along with Spatial API to manipulate position and orientation of entities
View Manager	Provides a way to create and manage views and can be used with Spatial API so that a view can be positioned and altered at runtime

CASE STUDIES

The utility of the Gears approach is best illustrated by a series of case studies that helped define the fundamental approach to solving the challenges. Using Gears, BISim has successfully developed a variety of projects with measurable successes in terms of reuse, development cost reduction and improved reliability. The following describe real-world developments faced in development of the Gears concept.

CCTT Manned Modules

The U.S. Army had a requirement to dramatically increase the realism of scenarios in the Close Combat Tactical Training System (CCTT), one of the U.S. Army's premier simulation-based training systems that was originally developed in the 1990s. CCTT aims to provide armor, mechanized infantry, cavalry and recon crews, units and staffs with a virtual, collective training capability. CCTT comprises three major crew and individual simulators, the CCTT Manned Modules, and Reconfigurable Vehicle Tactical Trainer (RVTT) System specifically designed for vehicle training. For over 20 years, CCTT used traditional image generation systems to create the out-the-window and sensor scenes for the trainees. But with developments in commercial video games, new technologies became available in the form of VBS3, based on the ARMA videogame technology, that could meet the new realism requirements for a fraction of the cost of the traditional image generators. Interfacing the commercial video game technology with legacy simulation systems was achieved by leveraging existing legacy CIGI standards through an interface to VBS through a Gears API.

Using this approach, BISim successfully integrated new IG technology based on proprietary commercial technologies and still support a myriad of joint, collaborative training systems required for this project. Developments were completed within 2 years. Without the Gears approach, BISim would have struggled to meet the stringent requirements of the initial phase of this massive project not to mention the challenges ahead for future planned program enhancements.

VBS Tactics

Another example is the use of Gears for the development of a new software application, which is a 2D web-based interface needed to provide for doctrinal control of artificially intelligent forces in training scenarios. The goal of the solution was to enable commanders to practice tactics in military classrooms where they control AI units in a white force role and craft plans for course-of-action war gaming. The interface was required to leverage improved artificial intelligence that allowed formations to act in accordance with military doctrine. Tactics relies heavily on an HTML5-based UI to provide a significant part of its current functionality. Tactics integrates advanced AI provided by our VBS Control product. This combination provides several interesting problems:

- How to quickly integrate a local UI component that can render an HTML5 UI?
- How to "serve" the HTML5 UI to the UI component?
- How can we prepare now for a complex integration of VBS Control in the future?

Gears allowed re-use of the HTML5 UI view component that was developed as part of a different product allowing rendering of an HTML5 UI in a window just like a browser. Gears then allowed creation of a separate component that serves the actual HTML5 UI just like a normal web server to the HTML5 UI view. Finally, Gears allowed the already defined and desired AI API to have the VBS Control implement the API based on the needs of VBS Tactics. The result is a plug-and-play HTML5 UI viewer, a web server that can serve an HTML5 UI to any web browser, and an AI component that implements the AI API. BISim estimates a 50% saving of developer cost by re-use of our Gears-based UI component and integrating the VBS Control AI behaviors instead of writing a custom AI library.

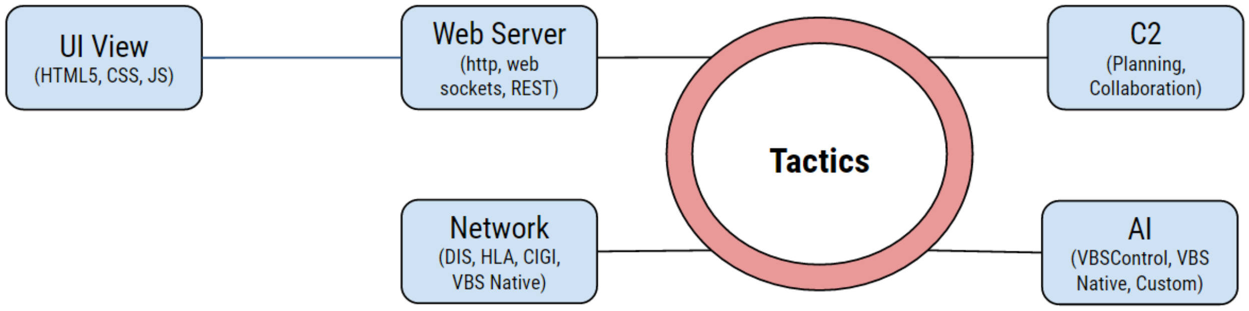
UI View
(HTML5, CSS, JS)
Web Server
(http, web sockets, REST)
C2
(Planning, Collaboration)
Tactics
Network
(DIS, HLA, CIGI, VBS Native)
AI
(VBSControl, VBS Native, Custom)

Figure 3: The VBS Tactics solution components.

The Gears runtime loads all the components for the application and then connects them directly to each other (peer to peer). There is a component that is dedicated to using all the APIs to make product APIs, which are aggregates of the component APIs. We also allow all components to interact in a standard way without necessarily knowing about each other ahead of time. This provides complete flexibility and modularity.

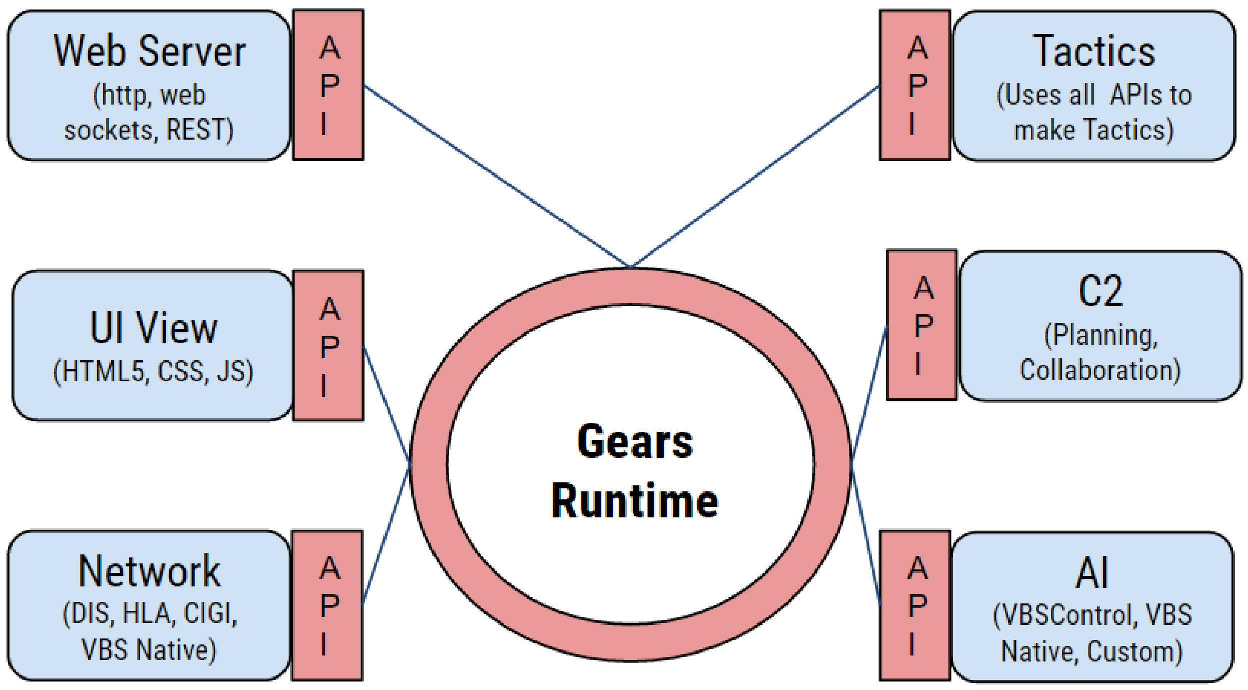
Web Server
(http, web sockets, REST)
API
Tactics
(Uses all APIs to make Tactics)
API
UI View
(HTML5, CSS, JS)
API
C2
(Planning, Collaboration)
API
Gears Runtime
Network
(DIS, HLA, CIGI, VBS Native)
API
AI
(VBSControl, VBS Native, Custom)
API

Figure 4: Gears Peer-to-peer connectivity.

We further found that using our approach, UI can be loaded in consumer browsers (e.g., Chrome) over the Internet. Microservices components connects with external services and is available to all cloud applications. Our approach even allows reuse of this integration with these services.

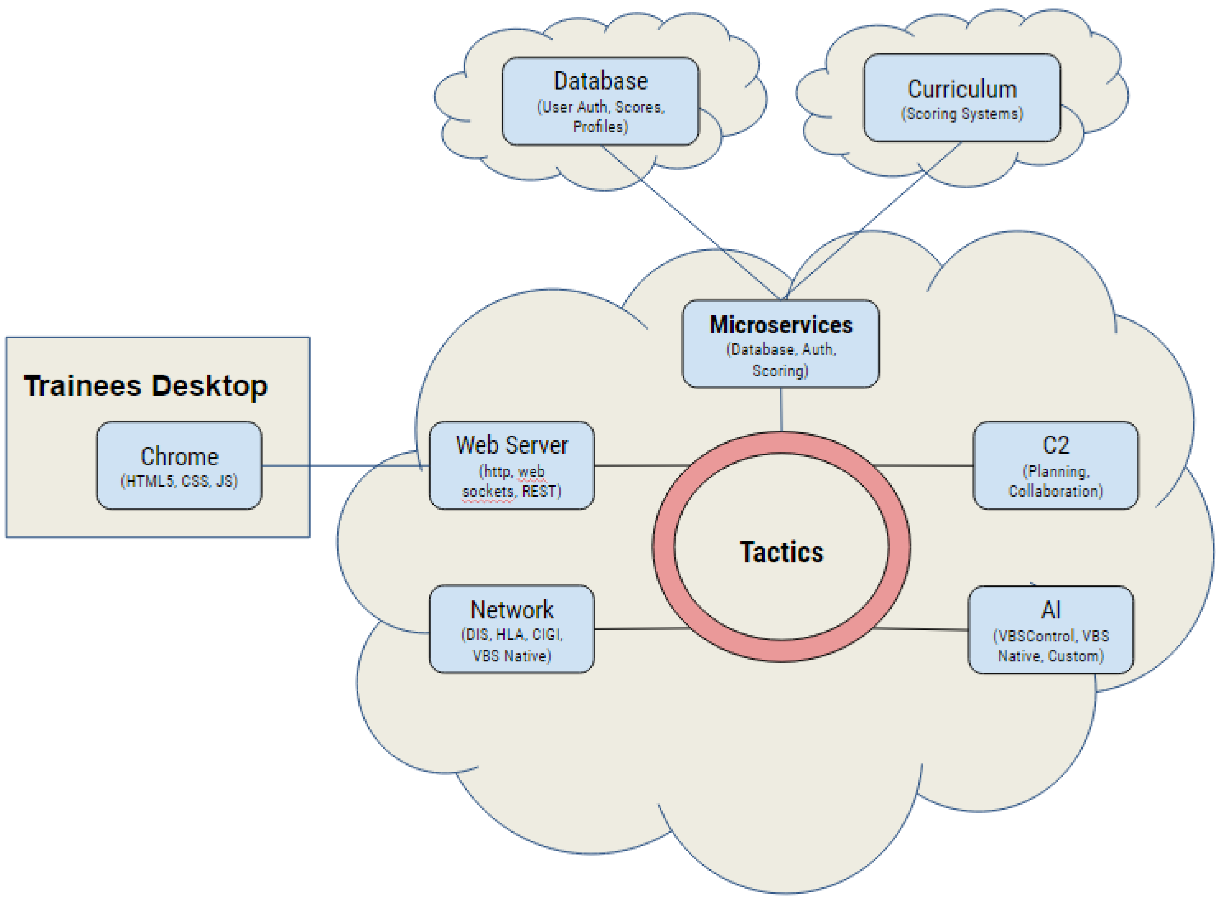
Database
(User Auth, Scores, Profiles)
Curriculum
(Scoring Systems)
Trainees Desktop
Chrome
(HTML5, CSS, JS)
Microservices
(Database, Auth, Scoring)
Web Server
(http, web sockets, REST)
C2
(Planning, Collaboration)
Tactics
Network
(DIS, HLA, CIGI, VBS Native)
AI
(VBSControl, VBS Native, Custom)

Figure 5: Gears used in the Cloud.

NEXT STEPS

The proposed approach is an open standards-based, modular simulation development framework designed to meet requirements for ease of use, modularity, extensibility and the ability to maintain compatibility with legacy technologies. With proper development and adoption, it has the potential to provide a system of system approach to integrating emerging and legacy technologies supporting reuse. With an ‘API first’ design to facilitate the development of components that are modular, re-usable, and safe to refactor we believe that once validated our approach could be a revolutionary step forward for the simulation industry, drastically improving capability for the defense industry by optimizing development, reuse and the overall stability of training and simulation capabilities delivered.

The next steps will involve work to formalize a service API layer that can be shared openly with industry as a common interface method, helping the industry more easily connect various disparate technologies and software. As these efforts mature, we plan to look for ways to work with standards organizations to evolve the technology. For example, Gears has already been proposed as a standard to NATO organizations currently using VBS and we look forward to continued collaboration with industry to further our work. Gears, as we are calling this development paradigm, could help define a new industry standard and help the industry collaborate more effective, keep pace with commercial innovation and, most importantly, provide relevant capability to the military.

References

[1] "USAF Looks to SCARS for Cost and Schedule Savings." (2016, November 28). *Show Daily: I/ITSEC 2016 Official Day New Digest*. National Training and Simulation Association. Retrieved from http://www.iitsec.org/Documents/IITSEC_2016_show_daily/IITSEC_ShowDaily_Day%201.pdf

[2] Synthetic Training Environment (STE). (n.d.).United States Army Acquisition Support Center. U.S. Army. Retrieved from http://asc.army.mil/web/portfolio-item/synthetic-training-environment-ste/

[3] High Level Architecture https://www.dmso.mil/public/transition/hla/

[4] Common Image Generator Interface Product Development Group. (2014). *SISO-STD-013-2014 Standard for Common Image Generator Interface (CIGI) Version 4.0.* Retrieved from https://www.sisostds.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=42031&PortalId=0&TabId=105

[5] "Distributed Interactive Simulation." (n.d.). Retrieved from http://open-dis.sourceforge.net/DIS.html

[6] Real-Time Innovations. (2014). *Data Centric Middleware*. 6-8. Retrieved from https://info.rti.com/hubfs/docs/RTI_Data_Centric_Middleware.pdf

[7] Real-Time Innovations.(2014). A Comparison and Mapping of Data Distribution Service and High-Level Architecture." Retrieved from https://info.rti.com/hubfs/whitepapers/Comparison_and_Mapping_of_DDS_and_HLA.pdf

[8] Richardson, C. (2014). "Pattern: Microservices architecture." *Microservice architecture*. Retrieved from http://microservices.io/patterns/microservices.html

[9] Hoff, T. (2015, December 4). “Deep Lessons From Google And EBay On Building Ecosystems Of Microservices.” *High Scalability*. Retrieved from http://highscalability.com/blog/2015/12/1/deep-lessons-from-google-and-ebay-on-building-ecosystems-of.html

[10] Christensen, B. (2013, January 15). “Optimizing the Netflix API.” *The Netflix Tech Blog*. Retrieved from http://techblog.netflix.com/2013/01/optimizing-netflix-api.html

[11] The United States Office of the Secretary of Defense Test Resource Management Center. (n.d). “Test and Training Enabling Architecture (TENA).” TENA. Retrieved from https://www.tena-sda.org/display/TENAintro/About+TENA