A COMMON COMPONENT-BASED SOFTWARE ARCHITECTURE FOR MILITARY AND COMMERCIAL PC-BASED VIRTUAL SIMULATION

by

JOSHUA LEWIS
B.S.A.S. LeTourneau University, 2001
M.S.E. Embry-Riddle Aeronautical University, 2002

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Modeling and Simulation
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2006

Major Professor: Michael D. Proctor

ABSTRACT

Commercially available military-themed virtual simulations have been developed and sold for entertainment since the beginning of the personal computing era. There exists an intense interest by various branches of the military to leverage the technological advances of the personal computing and video game industries to provide low cost military training. By nature of the content of the commercial military-themed virtual simulations, a large overlap has grown between the interests, resources, standards, and technology of the computer entertainment industry and military training branches. This research attempts to identify these commonalities with the purpose of systematically designing and evaluating a common component-based software architecture that could be used to implement a framework for developing content for both commercial and military virtual simulation software applications.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAR	After Action Review
AI	Artificial Intelligence
ATC	Air Traffic Control
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Components Off The Shelf
DEVS	Discrete Event Specification
DOD	Department of Defense
DSTA	Defense Science & Technology Agency
FSC	Full Spectrum Command
FSW	Full Spectrum Warrior
IAF	Israel Air Force
ICT	Institute for Creative Technologies
IFR	Instrument Flight Regulations
J2EE	Java 2 platform, Enterprise Edition
LAN	Local Area Network
MCMSMO	Marine Corps Modeling and Simulation Management Office
MEU	Marine Expeditionary Unit
MFS	Microsoft Flight Simulator
MOVES	MOdeling, Virtual Environments, and Simulation
PC	Personal Computer
SAAM	Software Architecture Analysis Method

SAF	Semi-Automated Forces
SAF	Singapore Armed Forces
USAF	United States Air Force
USC	University of Southern California
VFR	Visual Flight Regulations

1. INTRODUCTION

This chapter provides an explanation of the concepts and background information on the technologies used in this research project. It defines what is meant by component-based software architecture and identifies some advantages to using one. Background information is presented on personal computer (PC)-based virtual simulation including common uses and implementations. Finally, a description of the research area is outlined and an initial argument is made for a component-based software architecture that could be used to develop PC-based virtual simulations for both the military and the entertainment industry.

Component-Based Software Architecture

Software architecture represents the fundamental, encompassing design intelligence behind the implementation of a software system. Similar to blueprints made for a building, software architecture provides an embodiment of decisions made to meet a software project's objectives. It is a structural plan that specifies how the elements of a software system cooperate to meet a set of requirements (Hofmeister, Nord, Soni 2000).

Software architecture is also an abstraction of the design of a complex software system. It deals with the high-level structure of a software solution (Kruchten 1995). Software architecture is not a comprehensive decomposition of a system; it is a construct that helps manage the complexity of design through simplification and encapsulation (Hofmeister, Nord, Soni 2000).

Another role of the software architecture is that of a communication tool (Clements, Kazman, Klein 2002). To management, requirements analysts, and software

developers the architecture provides a rationale for the set of decisions made which, based on the prioritization of project objectives, detail a framework and direction for implementation. As such it is a traceable link between a project's software requirements specification and its design.

Hofmeister, Nord, and Soni (2000) identify the relation of the software architecture task to other development tasks:

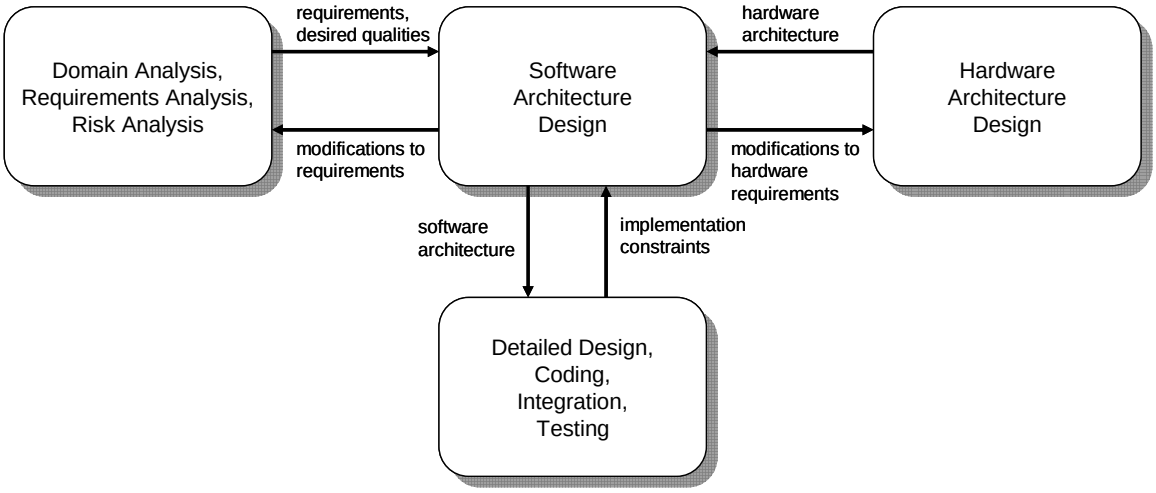

Figure 1: Relation of Software Architecture Task to Other Development Tasks

Software architecture represents the first step in the design process. It follows the definition of the problem space through domain, requirements, and risk analyses. The software architect uses these analyses to determine the software design roadmap for the project. The architect is responsible to ensure the solution's capability to meet the software requirements and provide feedback to be used in requirements modification if shortfalls are predicted. Software architecture provides the context and direction for the rest of the software implementation activities including detailed design, coding, integration, and testing.

Component-based software architecture is a type of architecture that provides support for the use of independent components each of which encapsulates some subset of the required software functionality. While similar to object-oriented technology, software components support stronger forms of modularity, lending themselves to a greater level of composition and reuse (Barros, 2004). The functionality of a combined set of components working together through a defined interface dictates the corporate functionality of objects, entities, and the system as a whole.

A component in a component-based architecture is an independent module that performs some functionality. Szyperski and Messerschmitt (2005) list the following five characteristics of a software component: multiple-use, non-context-specific, composable with other components, encapsulated, and independently deployable. To exhibit these characteristics each component in an implemented component-based framework must strictly adhere to a single interface definition. The component's interface is what allows it to be an independent, encapsulated set of functionality yet still have the ability to be used and reused along side other components in the framework.

Bass et. al. (2000) list several advantages gained from using component-based architectures. Because of the potential for reuse, component-based architectures improve programmer productivity from 30-50%. They provide a reduced time to market because they allow application construction through configuration and force the reduction of application complexity. Component-based architectures also provide a basis for reuse commerce through component distribution and marketing.

Component-based architectures are widely used in modern software applications. The Common Object Request Broker Architecture (CORBA) is an large-scale

component-based architecture which allows for distributed platform-independent object sharing and has been widely used in defense applications. COM and .NET are Microsoft's component based frameworks that provide the support for object reusability on and across Windows operating systems. J2EE is Sun's platform independent Java-based component architecture that is based on modular components running on an application server.

PC-Based Virtual Simulation

The personal computer is an independent computing unit that is normally intended for use by one person at a time. PCs are widely used both in the home and as business tools, and they do not require extensive technical expertise to operate. Typical uses for a PC include accessing email and the internet, running word processing and spreadsheet applications, listening to and viewing various forms of media, programming and software development, and playing games. PCs are inexpensive enough to fit into many home budgets and, over the last three decades, have achieved widespread popularity. A broad range of personal computing options are available to consumers including handheld devices, laptops, low-end and high-end desktops, multimedia centers, and dedicated gaming consoles.

The term "personal computer" can be traced to the early 1960s to a New York Times article reporting John W. Mauchly's speech to a group of industrial engineers where he said, "There is no reason to suppose the average boy or girl cannot be master of a personal computer." ("Pocket Computer May Replace Shopping List", *New York Times*, 3 November 1962.) With the development of the microprocessor and its subsequent

exponential gains in computing power along with increases in size and speed of dynamic and static memory, the lines have been blurred between PCs and high-end computers such as mainframes and servers. Indeed, many a PC can be retasked to play the role of a mainframe or server with only a change in its operating system.

Since their inception, PCs have been used to run virtual simulations of all types. A virtual simulation is a software application that allows users to interact with a computer-controlled virtual environment. The earliest virtual simulations were simple 2D arcade style games (*Tennis for Two, Pong, Spacewar*) and text-based games (*Hunt the Wumpus. Adventure*) where a user would interact with the non-visual virtual environment through keyboard input and gain a perception of the virtual world through computer generated text outputs. With the advances of PC-based graphics, sound, and haptic technology, virtual environments have been represented with an increasing amount of fidelity. Current virtual simulations running on current PCs can provide the user with a three dimensional view of an immersive virtual environment complete with realistic color and visual effects.

PC-based virtual simulations have taken on many forms and purposes. Most have taken the form of video games used for entertainment. Some of the genres of these video games include puzzles, strategy, sports, racing, first and third person shooter, adventure, and role-playing. Virtual simulations have also been used for education and training. Because it is often easier to understand a concept through visualization, there are simulations available that help students visualize and comprehend natural phenomena. Simulations like this are available for subjects including electricity, optics, quantum physics, and superconductivity to name a few. Virtual simulations also provide safe

mechanisms to train students how to control complex systems. They are used to train individuals how to drive cars and trucks, pilot aircraft, navigate vessels, and operate nuclear power plants. The simulations are useful for education and training because they can be used to model scenarios that cannot be easily or safely executed in the real world.

Because virtual simulations can be used to model the interactions of existing and proposed systems, they have also been used as a basis for analysis and as a tool for communication. When hurricane Katrina blew through New Orleans, CNN used a virtual simulation of the city's levee system to communicate to its viewers how the city became swamped with water. Because virtual simulations can provide an accurate logical and visual depiction of many aspects of the real world, they can be used as a basis to analyze parts of the real world as well as communicate those aspects to others.

Research Area

Since the beginning of the personal computing era, the military has been interested in taking advantage of the rapid advances of PC-based virtual simulation. The U.S. military is the world's largest consumer of digital game-based learning (Prensky 2001) and for good reason. Due to the increasing popularity of computer games and the consumer demand for the latest in technology, graphics, and game design at an affordable price, the video game industry is continuing to grow. It has in some respects passed both the movie industry and the traditional commercial computer industry as it has become a larger consumer of high-end computer hardware and software. The most sophisticated rendering hardware and the most responsive interactive simulation software is found in the machines used to power computer games (Lewis, Jacobsen 2002). Cost of game-

based hardware and software has been driven down dramatically by technology advances and consumer economics while quality and realism of desktop simulation technologies has continued to improve (Morris, Tarr, 2002).

Added to the military's interest in PC-based virtual simulation is a set of mutual objectives shared between the entertainment industry and the military related to the PC. At first glance it would not seem like the two have much in common. The entertainment industry is focused on providing video games that offer a diversion from the real world while the military's interest in simulation is to replicate many of their real world systems with as much fidelity as possible (Fong 2004). However, modeling and simulation technology lies at the heart of video games while also providing a low-cost means for the military to conduct joint training exercises, evaluate new tactics, and analyze new weapons systems (Alexa, 2004). One common interest of both the military and entertainment industries is the creation of low-cost, large-scale massively multiplayer online interactive simulations (Zyda and Sheehan, 1997). Because many of today's computer games are designed from the outset for network play they already have much in common with the military's large scale distributed simulations (Lewis, Jacobsen 2002). Another aspect of common interest stems from the fact that current and future generations of soldiers entering the military have grown up playing computer games. Using game-like applications for training provides a smooth transition for younger soldiers entering the military (Macedonia, 2002). The military branches are also interested in implementing COTS solutions, and PC technologies represent a COTS solutions that have supplanted many custom developed simulation and control applications (Baracos, 2001).

There are numerous examples of PC-based virtual simulation projects that represent a transfer of skills, knowledge, and technology between the military and the entertainment industry. The U.S. Navy found that students who used *Microsoft Flight Simulator* were more likely to receive above average scores in real flight tests. Both the Danish Army and the U.S. Army have used *Steel Beasts* to train tank commanders. An attempt was made by Peter Bonanni to use *Falcon 4.0* as a low-cost F-16 training alternative in the Virginia Air National Guard. *Spearhead II* was the result of a U.S. Marines contract to build both a commercial and military version of a PC-based tank simulation. *Real War*, a strategy simulation was a similar commercial game and military simulation project that allowed users to command joint military forces at the theater-level in a virtual war. *Marine Doom* was a successful adaptation of *Doom II* used to train Marines in a first-person team-based combat simulation. The commercial *Unreal* game engine was used to provide the framework for content created for the U.S. Army's commercially successful recruiting tool, *America's Army*. Simigon used the framework behind the flight simulation games *IAF* and *USAF* to create the PC-based military simulation platform *Airbook*.

The military's interest in PC-based virtual simulation, the common simulation-related objectives held by the military and the entertainment industry, and the history of common simulations that have been developed and implemented on both sides makes a strong argument for a common component-based software architecture for PC-based virtual simulation. Such an architecture could support the common interests and objectives listed while providing a single platform for the development of future commercial and military simulations. It would support the creation of reusable

components that could be employed without little or no change across both domains and across application contexts within each domain. It has the potential to reduce development and implementation costs across the domain boundary and could be structured to constantly adapt to and take advantage of the rapid advances of PC technology. The following chapters take an in-depth look at what work and research has been done in this area, identify work remaining, propose a process for producing and evaluating such an architecture, detail an implementation and analysis of the architecture, and provide conclusions on the research accomplished.

2. REVIEW OF RELATED PROJECTS AND LITERATURE

This chapter provides an examination of the various strategies used in the past to repurpose PC-based virtual simulations to help achieve a military objective. It examines the work accomplished as a product of the collaboration of the military and the entertainment industry. Questions and remaining research areas identified by the literature are listed. Finally an argument is made for the necessity of a common component-based software architecture for PC-based virtual simulation, detailing the problems it addresses and the benefits it provides.

Strategy for Common Use: Reuse

One strategy used to repurpose PC-based virtual simulations is reuse. Reuse applies to those methods where a COTS commercial game is deployed in its original domain but where its purpose has changed. Reuse requires no changes to the underlying framework, structure, or content of the game other than that allowed by non-expert consumer-oriented tools created by its developer for that purpose. Games reused for military applications are typically highly realistic simulations of specific, complex, real-world domains.

Microsoft Flight Simulator

Microsoft Flight Simulator (*MFS*) is a PC-based non-military flight simulation software application built for use with the Windows operating system. It allows users to pilot a wide variety of aircraft in an environment that represents the entire world in three dimensions. The original concept for *MFS* came about through a series of computer graphics papers written by Bruce Artwick in 1976. In 1979, his company, subLOGIC,

released *FS1 Flight Simulator* for the Apple II. In 1982 they licensed an IBM compatible copy to Microsoft which was distributed as *Microsoft Flight Simulator 1.01*.

Because it was developed by Microsoft and because it achieved widespread popularity, *MFS* has been able to stay on the cutting edge of PC and PC-based graphics technology since its inception in 1982. Throughout its lifetime it has remained an affordable software application ($40-$50). In 1982, its three dimensional world was rendered to monochrome wire frames, but today its visual fidelity rivals that of the best multi-million dollar simulators in the world.

While *MFS* is built and distributed as a computer game, it is widely used as a flight training aid in both commercial and military flight schools. The Flight Safety International Academy in Vero Beach, FL requires that students complete 27 hours of instruction using *MFS* as part of their Career Pilot Program. Students practice completing checklist procedures, observing ATC instructions, and performing basic VFR and IFR flight maneuvers. *MFS* is also issued by the U.S. Navy to each of its student pilots. (Microsoft 2005). An extensive study by the U.S. Navy found that students who used products like *MFS* were 54 percent more likely to finish above average in real flight tests than those students who had not used them. (Macedonia 2002).

Falcon 4.0

Falcon 4.0 is a PC-based commercial flight simulation game based on the military F-16 fighter jet. It was developed and published in 1998 by Microprose, a company that developed both strategy and simulation games. The game is widely recognized as an extremely realistic simulation of the Block 50/52 F-16 series with accurate cockpit interactions, flight model, and combat missions. The game comes with a 600 page

manual and some users consult the real "Dash 1" F-16 manual when flying the simulation. (Lenoir 2003).

The realism of the game makes it a natural candidate for use in military training. The latest version of the game supports multiplayer squadron-level play with dynamic scenario generation over realistic Korean peninsula terrain rendered from satellite imagery. Peter Bonanni, an instructor at the Virginia Air National Guard, worked with Microprose to license the game for use in training the National Guard students. Bonanni was impressed how *Falcon 4.0* mimics the look and feel of the real aircraft, supports team training, and provides a realistic virtual environment around the pilot (Lenoir 2003).

Unfortunately, *Falcon 4.0* has had an unstable history. Its original release in 1998 contained numerous bugs, many of which were fixed in a later software patch. The source code to the game was leaked soon after, and numerous companies took it upon themselves to make improvements to the game's code and release versions of their own. The original developer, Microprose, was under the control of a company named Spectrum Holobyte at the game's release date. Shortly after, in 1999, Spectrum Holobyte was acquired by Hasbro. Hasbro sold all development assets to French holding company Infogrames, owner of Atari, in 2001. Atari then issued a cease and desist directive to all companies creating improvements for *Falcon 4.0* and licensed rights to development company Lead Pursuit which released its own version of the Falcon line, *Falcon 4.0: Allied Force* in 2005.

Steel Beasts

Steel Beasts started as a PC-based tank simulation when it was released by eSim Games (formerly Shrapnel) in 2000. Players could operate one of two tanks as the

gunner or tank commander. The game has since evolved into a ground-only war game simulation including tanks, armored personnel carriers, and infantry where players can direct companies of tanks against computer controlled opposing tank forces. *Steel Beasts* has mundane graphics compared to other modern games but concentrates instead on providing an extremely realistic tank operation and combat experience. As such it is a difficult game to play and appeals to a small fan base interested in realistic tank simulations (Gamespot 2002). Released in 2005, *Steel Beasts II* incorporates helicopters, advanced map and AAR features, artillery, minefields, and more realistic graphics than the original.

Like *Falcon 4.0*, because of its realism, the game lends itself naturally to military training. Unlike *Falcon 4.0*, *Steel Beasts*' life since inception has been directed by one owner and developer, Alexander Delaney, and is currently in use by several military customers worldwide. Military users include the Finnish Combat School, the Dutch Cavalry School, the Swedish Combat School, the Danish Army Combat School, and Spain's Ejercito del Tierra. (eSim Games 2005). It was also used for a time by West Point Academy to train cadets (Macedonia 2002).

Advantages of Reuse

The immediately obvious and possibly most significant advantage of the reuse of a PC-based virtual simulation is their potential for low lifecycle costs (Morris, Tarr 2002). Initial licensing costs tend to be low because it is the same software that consumers can afford. Many professional commercial and military training organizations already issue students PCs when they begin training. For only a few dollars more the organizations can provide the student a COTS simulation product that provides them a

supplemental training aid. The COTS simulations are simple to integrate; set-up normally consists of double-clicking an icon and choosing a few installation options. They are also cheap to maintain as most development companies provide free patches and upgrades to their general consumer base.

Another advantage of reuse is the low development risks associated with creating custom content and scenarios (Fong 2004). Because the game's use for entertainment is often similar to the game's repurposed application, the tools and documentation created for the average consumer can be used to customize the game to suit its new environment. For example, *Microsoft Flight Simulator* comes with software that allows players to create and modify their own aircraft and scenarios as well as view playbacks. Organizations using *MFS* for pilot training could use the same tools to create appropriate aircraft and the required training scenarios with the capability for after action review. Since the tools were created for consumers, it does not normally require much expertise, other than domain knowledge, to adapt the simulations for training or other purposes.

Drawbacks to Reuse

While the risks for implementation and customization are low, the risks for continued support of a COTS PC-based virtual simulation are high. Companies like eSim Games which create highly realistic, specialized simulations may be small companies with a small commercial fan base. They may not have the resources available to provide support for a large military training installation. Small companies, like Microprose, are also more subject to buyout or financial trouble and may not be able to continue to support the product or may cease to exist altogether.

Another drawback to reuse is the lack of control over the reused COTS simulation product line. A certain amount of trust in the validity of the simulation models must exist before they can be used for purposes such as training or mission rehearsal. Paul and Taylor (2002) state that the process for the development of trust in a COTS simulation product may be expensive enough to outweigh advantages of its use.

Strategy for Common Use: Contracted Development

Another strategy used to provide PC-based virtual simulations for military is through contracted development. Contracted development refers to those situations where a military body subcontracts a software development company to create a PC-based virtual simulation from scratch or from a base application that must be drastically modified to meet the contract's requirements. These contracts might specify or allow the development of both a commercial and military version of the software.

Spearhead II

Spearhead II is a PC-based tactical trainer that simulates a real-time tank battle. Players are required to develop battle plans and then implement them in the game's synthetic environment. Exercises can be conducted in single-player mode or multi-player mode on a LAN or over the Internet. Users can communicate with other live players or direct the operations of automated forces. Exercises end with an after-action review, detailing successes and failures of the battle plan's implementation. The game was designed to train tank commanders in battle planning, decision making, and situational awareness, but it was also released commercially to allow civilian users to experience the combat expertise required of the professional soldiers that make up the Army's tank

crews. The military version incorporated more realistic weapons and tactics and was more difficult to play than the commercial version.

Spearhead II was developed by Mäk Technologies and published by Interactive Magic in 1998. The game was developed under a military contract for *Marine Expeditionary Unit 2000* (*MEU 2000*), an HLA compliant multiplayer PC-based tactical decision making game that was to be concurrently released as a commercial game. The contract was rewritten to specify a PC-based tank simulation game that resulted in *Spearhead II* (Lenoir 2003). While the game never achieved widespread commercial success, it was used by the U.S. Army Armor School at Fort Knox for tank crew and commander training and by the Army's Mounted Maneuver Battle Lab for experiments and analysis.

Real War

Real War is a PC-based real-time strategy game that allows player to control air, land, and sea forces of the United States or the fictitious adversary forces of the Independent Liberation Army. Players manipulate forces in the game by making theater-level decisions that affect their military campaigns, planning and executing joint forces coordinated attacks as well as building up and protecting supply lines. The game gives players control over traditional military arsenals such as infantry, tanks, aircraft, and weaponry, but also allows them to use specialized combat tactics including reconnaissance aircraft, electronic warfare, psychological warfare, and nuclear weapons.

Real War is a real-time strategy game developed by Virginia-based defense contractor OCI and video game developer Rival Interactive. The game was originally built under contract from the Joint Chiefs of Staff as a PC-based computer game taught

joint doctrine, offering guidance on coordinated war operations across all branches of the military. The military version of the game was titled *Joint Force Employment* and is used for training in military colleges including the U.S. Joint Forces Staff College, Joint Special Operations University, and Air University.

Advantages of Contracted Development

One of the reasons that contracted development of a PC-based virtual simulation is attractive is that it requires little in-house expertise and can potentially result in a computer game that can be sold commercially as well as meet military needs. It does not require the contracting agency to hire artists, programmers or computer technologists because all that expertise is outsourced. It requires only the technical expertise needed to write and evaluate the requirements for the product and the ability to evaluate the product itself to ensure requirements are met.

Contracting, unlike reuse or adaptation, can also result in a PC-based simulation that is built from the ground up to meet military needs. *Spearhead II* was built by Mäk as an HLA-compliant application to meet the training needs of tank commanders (Erwin 2000) while *Real War* was constructed to provide joint forces training (Cornerstone 2005). Neither was constrained by the limits imposed from reuse or adaptation of an existing product.

Drawbacks to Contracted Development

One of the drawbacks to contracted development is the expense incurred. Development costs for a modern, viable PC-based game are $2.1 million per year for the first 2-4 years (Prensky 2001). Because contracted solutions are custom-made and may be built from scratch, costs associated with the end product may be much greater than

costs associated with reusing an existing product. Reuse often occurs on a platform that is a commercial success, implying widespread use and low licensing costs. Contracted development assumes all the risk and cost of a startup operation.

Even though the risk and expense of contracted development may be high, the end product may not be a commercially viable simulation. Because the simulation was built with the end goals of the contracting agency, it may not be attractive to civilian gamers. For example, *Real War* earned a Poor rating (3.7 out of 10) from CNET and 3 out of 10 from Gamespot for, ironically, not being realistic. While the reviewers admitted that the game enforced the use and coordination of all military branches (the intention of the military version used for training), they complained of poor graphics and effects, poor AI, and a poor user interface (CNET 2001, Gamespot 2001). *Real War* met the objectives of its military contract but failed commercially because it did not offer a viable alternative to the other military strategy games of the day.

Strategy for Common Use: Adaptation

Adaptation of a virtual simulation involves modifying a commercially available product to suit an objective other than the simulation's original purpose. Because of the large cost associated with fronting a computer game, game engines are currently designed in such a way to separate functionality from content so that they can support a family of games (Lewis, Jacobsen, 2002). Development companies often release toolkits concurrently with or soon after the release of the game, giving end users a method to create new types of content (characters, vehicles, weapons, or scenarios) which can be run on the game's framework. This method can be used repurpose a game's content

while maintaining the use of its underlying engine. Another method of adaptation involves employing a game's developer, often under strict licensing agreements, to expand a game's content to apply to other objectives (Fong 2004). A third method of adaptation involves modifying a commercial game's underlying framework so that it can be used to develop content that serves a different purpose.

Doom to Marine Doom

One of the first 3D simulation games to be adapted to military training was the fantasy game *Doom* (Macedonia 2002). In 1995, the U.S. Marine Corps Modeling and Simulation Management Office (MCMSMO) created a new version of the game called *Marine Doom* which was retasked to build the effectiveness of 4-soldier fire teams (Fong 2004). The demons and firepower of Doom were replaced by enemy soldiers and Marine-issued firearms, and new scenarios were created to teach basic combat skills like conserving ammunition and observing the chain of command (Macedonia 2002). The scenarios also emphasized team coordination, communication, and decision making under pressure (Riddell 1997).

In 1997, Marine Corps Commandant General Krulak released Marine Corps Order 1500.55, a directive that encouraged the adaptation of specific commercial PC-based war games which could be used to develop military thinking and decision making. It identified the MCMSMO and their list of suitable customized commercial computer war games as resources to be used for such development. Gen. Krulak noted that PC-based war games provide a potential for Marines to develop decision making skills especially when live training opportunities are limited, and he authorized the use of government computers for approved PC-based war games. It assigned "responsibility for

the development, exploitation, and approval of PC-based war games to the Marine Combat Development Command" (MCO 1500.55 1997), emphasizing Gen. Krulak's position on the importance of commercial game adaptation.

Jane's USAF to Airbook

In 1998, Electronic Arts (EA) released a military-themed flight simulation game developed by Pixel Multimedia called *Jane's USAF*. The game includes variations of eight modern fighter aircraft to fly, four air campaigns to play, and an editor to quickly create standalone missions. Like *Falcon 4.0*, *Jane's USAF* introduces some amount of dynamic scenario generation within scripted missions to create a unique combat environment for each flight. *USAF* was deemed good enough to be ranked by Gamespy (2004) as one of the top PC-based flight simulations ever created.

In the same year the game was released, Pixel Multimedia spun off a new company named Simigon that was to repurpose and market the game for use in military training. The company's vision is to provide a "see it, do it" approach that allows pilots to train in the same environment, albeit a virtual one, in which they fly (Simigon 2005). Their management consists of a number of ex-military pilots that realized the potential of a PC-based virtual simulation solution to provide a viable source of low-cost flight training.

Shaul Samara, vice president of development and former A-4 pilot, relates some of the issues the company had repurposing *Jane's USAF* for military flight training. The original game had been highly optimized to provide the best graphics and fastest game play possible on the PC. Decisions had been made to develop the game as a streamlined, monolithic entity that used minimal processor and memory overhead. As such, the game

was extremely difficult to adapt to a training environment. There was no feasible way to integrate a learning management system, create new types of content or scenarios, or support required military standards such as the Shareable Content Object Reference Model (SCORM) or the High Level Architecture (HLA). Samara said the development team had to completely rewrite the underlying framework to support an extensible architecture (Samara 2005).

The new architecture, *Knowbook*, is oriented to support the content, tools, and environment required for all types of PC-based training. Simigon's flagship training application, *Airbook*, is a simulation-based tool created to track a military pilot's progress through initial flight training, weapons systems training, mission rehearsal and readiness, after action review, and recurrency training. While the aircraft graphics, cockpit interiors, visual effects, and terrain can trace their heritage to *Jane's USAF*, the underlying structure of Airbook is completely different than the game's. *Airbook*, unlike *Jane's USAF*, is flexible and extensible, supporting component-based simulation, diverse content types, learning management, HLA, SCORM, virtual instruction, and distributed mission training.

The Unreal Engine and America's Army

On July 4, 2002, the U.S. Army released *America's Army*, a free PC-based virtual simulation that was developed primarily as a recruiting tool. The game was built as part of $2.2 billion worth of funding allocated by the U.S. Congress to increase recruiting numbers in the armed forces (Sourcewatch 2005). It was the brainchild of Col. Casey Wardynski and was meant to provide a more accurate representation of combat than the traditional military games that were commercially available (Roth 2003). The game was

originally designed and built by the MOdeling, Virtual Environments, and Simulation (MOVES) Institute, part of the Naval Postgraduate School in Monterey, California. The MOVES Institute was founded under the direction of Dr. Michael Zyda with $45 million of U.S. Army funding after a 1997 report by the National Research Council that called attention to the fact that Department of Defense (DoD) simulations were lagging behind commercially available games and advised collaboration with the entertainment industry (Sourcewatch 2005).

The original version of *America's Army* was built on the framework of Epic Games' *Unreal* gaming engine. The game was built to reflect core U.S. Army values and open the door to reveal the world of the U.S. Army soldier to the public. Players can virtually experience many aspects of the lives of real American soldiers including boot camp, Ranger and Airborne training, Special Forces operations, rules of engagement, lifesaving, rules of war, and medical skills (America's Army 2005). The game is noted for its unusually realistic content including visuals, sounds, weapons modeling, and combat scenarios which are attributed to a combination of the strength of the *Unreal* engine and the influence of Army experts that worked with the game's developers (Gamespot 2002).

In 2004, the U.S. Army contracted Ubisoft, a French commercial video game company, to publish future versions of America's Army for console gaming platforms including Xbox and Playstation. The first game, *America's Army: Rise of a Soldier* was developed by San Francisco based Secret Level and released in 2005. *Rise of a Soldier* is a role-based virtual simulation that follows the player's character through the career of a U.S. Army soldier from a new recruit training at Fort Benning through the ranks to,

ultimately, lead of an elite Special Forces unit. The game was built to impart core U.S. Army values emphasizing teamwork and real infantry tactics. To succeed a player must learn how to make best use of firing posture, situational awareness, fire suppression, and teamwork (Secret Level 2005).

Strengths of Adaptation

Adaptation of commercial simulations to alternative uses has several advantages. Because of the market demand for PC technology, the low cost of games, and the ubiquity and low cost of PCs, adaptation of a PC game can be a cost-effective solution to complex problems ranging from system familiarization to training to mission rehearsal. The Marine Corps cost for each license of *Marine Doom* was $49.95 (Riddell, 1997). This represents a small initial material investment to produce prototype modifications and provide a basis for feasibility studies. It also represents a small outlay for materials required for deployment of the solution.

Another advantage of adaptation is the potential cost savings due to low development time and low implementation complexity. The Marine Corps development team, for example, stood up the initial release of *Marine Doom* in three months (Riddell, 1997). Game development toolkits provided by commercial game developers can be used to create custom content and scenarios with a short turn around time (Fong 2004). Because the tools and framework are not touched, there is often no need for code recompilation, integration, and testing. The content that is created often resides in text or resource files and can be used "as is" on the game engine. This greatly shortens development time and complexity and relieves the need for the expertise required for traditional application development.

Adaptation provides a vehicle for a military organization to enforce its ideology or doctrine in a simulation application in a setting that may be familiar to its users. An example of this is found in *America's Army* where success in game play is based around accepting and practicing the U.S. Army's proclaimed core values. The game only presents a controlled, one-sided, positive spin on U.S. military operations and avoids other issues such as the morality of war, collateral damage, and politics. As such it is the first overt example of the use computer gaming to espouse political aims (Sourcewatch, 2005). Based on the game's popularity and widespread use, it probably will not be the last.

Drawbacks to Adaptation

There are several drawbacks to adaptation of commercial simulations for alternative uses. One disadvantage is that the simulation is being altered to suit an objective for which it was not designed. Because the original commercial products were developed with specific objectives, they inherently contain built-in limitations that adaptation must overcome. *Marine Doom*, for example, had to replace the demonic enemy forces of *Doom* with human enemy soldiers and the other-worldly weapons with Marine-issued ones. Simigon found they had to rewrite the underlying structure of *Jane's USAF* to support the requirements of military training. While some adaptations may be possible, others may not be. It would be hard to adapt a flight simulation for complex ground-based interactions and probably impossible to turn it into something like an underwater submarine simulation.

Another drawback to adaptation is the lack of control over fundamental aspects of the simulation. The development tools that come with video games only allow

modification to a certain degree, and there is likely no possible access to the game engine's source code. The tools are often purposely designed with a decreased level of functionality so that users may not reproduce content and scenarios to the complexity and fidelity of the original designers. The tools may enforce limitations to the degree that the application and content can be modified and may not allow, for example, adaptation to provide implementation of a critical mission planning or after-action review phase (Fong 2004). In order for adaptation to provide a viable solution, the planned development work should not exceed the capabilities of the adaptation tools provided.

Research Through Collaboration at the ICT

The Institute for Creative Technologies (ICT) is a research center affiliated with the University of Southern California (USC) that fosters joint collaboration of the military and entertainment industry on developing new modeling and simulation technologies. ICT's mission is to achieve verisimilitude in synthetic experiences through a participant's physical, intellectual, and emotional immersion in a virtual three dimensional environment (Macedonia, 2001). To achieve this it provides a research environment where entertainment industry experts collaborate with military and academic researchers to leverage the strengths and skills of both domains (Lindheim, Swartout 2001).

ICT was initially funded by a grant from the U.S. Army given to USC in 1999 to create a research center focused on developing advanced military simulations. The contract was prompted by the 1997 National Research Council study that identified the benefits that could be obtained by military and entertainment industry collaboration

(Korris, 2004). One such benefit was the potential realism that could be obtained through the addition of emotion to the traditionally sterile military simulations through the addition of a compelling story line, a feature used in all movies and many computer games (Lenoir 2003).

Full Spectrum Command is a PC-based company command training simulation that resulted from a research project completed by ICT in 2003. Military students play the role of commander of a U.S. Army light infantry company who must comprehend the assigned mission, plan and organize the mission, and coordinate the execution of the mission with over 100 virtual soldiers. The missions were meant to develop cognitive skills such as tactical decision-making, resource management, and adaptive thinking (ICT 2005).

Another of ICT's research projects, *Full Spectrum Warrior* (*FSW*), became the first military training application developed for a commercial game console. Based on Microsoft's Xbox console *FSW* is a cognitive tactical trainer for the Army's smallest Light Infantry maneuver unit, the nine soldier squad. The application places players in a first-person role as a weaponless squad leader who must direct the movements of squad members through dismounted urban battle drills. Exercises are meant to hone the decision-making skills of infantry soldiers and increase their situational awareness in combat (Korris, 2004).

The ICT has also performed research for an architecture that will support PC-based military and commercial entertainment interests. The project, called the Integrating Architecture, leverages the strengths of game engines and military simulations to provide an infrastructure for research efforts in the areas of artificial

intelligence, graphics, sound, animation, and immersive display technologies. The Integrating Architecture centers on combining the *Unreal* commercial game engine with the *OneSAF Objective System* military simulation environment with the objective of providing a platform that allows researchers to facilitate the transition of new technologies into immersive training systems (ICT 2005). Michael van Lent (2004), the project lead, identifies the core design principles of the architecture as: supporting low-cost research in the latest simulation, animation, and game technologies, providing a pipeline from research to development, and providing a technological foundation for ICT that is custom-built to meet simulation researchers' needs.

Component-Based Modeling and Simulation

Some research has been done in the area of component-based modeling and simulation. Bunus and Fritzen (2004) propose a methodology to analyze static aspects of component-based equations used for mathematical modeling in the language Modelica. Delinchant et. al. (2004) describe a component-based approach and tools used for designing and composing subsystems used in electrical systems simulations. Samantarray et. al. (2004) present an ontology for classifying and connecting thermofluid process components. Hoffman (2004) specifies criteria for decomposing systems into components for use in modeling and simulation. Shibuya (2004) discusses a component-oriented grid-based framework that supports models representing humans, social situations, and spatial settings. Yilmaz (2004) identifies compositional consistency problems with DEVS components and submits an algorithm to verify their interaction policies.

Questions Being Asked

While a number of projects have successfully demonstrated military and entertainment industry collaboration and research concerning component-based simulation continues, much remains to be done. Specifically, the following needs have been identified:

- Barracos (2001) states that real-time simulation should be affordable, simulation architecture should be scalable, and simulation should be able to benefit from the market-driven advances in commercial technology, incorporating the latest technologies as soon as they appear.
- Pace et. al. (2001) suggest that it is increasingly important to find ways to make simulation frameworks adaptable because it allows them to cope with the continuous evolution of software and evolve to accommodate variations of a problem without much rework in previously developed components.
- Morris and Tarr (2002) state that there is a need for a strategic means to analyze and extract components of COTS synthetic environments for customized application capability.
- Zyda et. al. (2003), looking at the future of creating military-based massively multiplayer games, posited that one possibility was for the government to procure or develop a game engine capable of full-spectrum combat modeling and large-scale interoperability integration with a programming interface for modeling human behaviors and creating stories. They state that such a solution should also incorporate

a rapid prototyping interface that would allow missions to be created nearly overnight.

- Barros and Sarjoughian (2004) state that further research is needed to develop new methodologies that fully support component-based modeling and simulation aimed at representing a wider variety of systems.
- Fong (2004) states that one of the most difficult challenges to surmount is the ability to adapt COTS games for the military because of the lack of access to underlying source code which presents limitations to the degree that the game can be modified. She is seeking other ways that COTS computer games may crossover to meet the needs of military simulation.

Argument for a Common Component-Based Software Architecture

A number of drawbacks have been identified relating to strategies used in the past to create or repurpose commercial PC-based virtual simulations to meet military objectives. A number of areas of need identified by the literature related to cross-domain use and component-based simulation have been listed. This section makes an argument for the use of a common component-based architecture for PC-based virtual simulation by describing its ability to provide solutions for those drawbacks and areas of need.

Description of a New Strategy

A common component-based software architecture represents a new strategy for cross-domain development and reuse for PC-based virtual simulation. Implementation of such an architecture would require creating a framework from the ground up that would have the ability to meet the goals and constraints of both the military and the

entertainment industry simultaneously. The framework would provide a generic, common platform for the development of application-specific virtual simulation solutions.

Specifically, a common component-based software architecture for PC-based virtual simulation will:

- provide a common platform to create entertainment and military solutions,
- support reusable software components across varying simulations and domains,
- and allow for interchangeable simulation software components.

Solution to Previous Weaknesses

A common component-based software architecture for PC-based virtual simulation would address many of the drawbacks to previous strategies. It has the potential to allow full product line control as there would not be a heavy dependency on a third party product. While there would be an initial implementation cost, the component-based simulation framework would have a low lifetime cost because of the savings gained from a single development platform that could support multiple solution and savings gained from reusable components built to support multiple domains. Because the architecture will have been created to address the goals and priorities of both the entertainment industry and the military, it could be used to produce simulations that support both military contracts and viable commercial applications. This could be done without the re-engineering effort currently required to adapt a simulation from one domain to the other.

Further Strengths / Benefits

Such an architecture would also provide further benefits. Because it provides a solution built from independent components, it would be inherently scalable and quickly adaptable to a changing problem domain, changing requirements, and technology advances. It would have the ability to model a wide variety of systems through different applications of a core set of reusable components. Due to component independence, source code would not be required to adapt components to new uses. The resulting framework could provide both a programming and non-programming interface for application development and system composition. Because of its reliance on composability, the framework would also provide an environment for rapid prototyping and implementation.

3. METHODOLOGY

This chapter presents a methodology for the research that will be accomplished. It first summarizes the research concept and research goals and then outlines a four phase approach for creating and testing the software architecture. Phase I is an analysis phase that uses stakeholder input to identify risks and issues that the architecture will have to account for. Phase II presents a process for the design and documentation of the architecture. In Phase III, an implementation of the architecture will be completed through the development of two prototype applications. An evaluation of the prototypes will be carried out in Phase IV to verify that the architecture exhibits the traits and characteristics required to meet its objectives. Finally a summary of the original contributions made by this research will be given.

Research Concept

This research attempts to systematically create, document, and evaluate a common component-based software architecture for use in the design, development, and sustainment of a family of PC-based military and commercial virtual simulations. This research will be scoped by its focus on a single product line, or family, of small-scale virtual simulations used for military training and commercial entertainment. Principles discovered in this research, if proven valid, should be able to be generalized to other larger-scale virtual simulation architectures and product lines. The research will prioritize breadth over depth, meaning it will attempt to address the structures required to support a wide variety of virtual simulations implemented for entertainment or military use, but it will not provide a full decomposition of every structure. The research will

emphasize the development of design decisions over the specification of design details. It will not attempt to fully describe every component, specification, and protocol, but it will note where future work is required and provide direction for that work.

Several goals have been set for this research. It will attempt to identify the priorities and goals of many of the military and entertainment industry stakeholders involved in PC-based virtual simulation. It will identify the principal technical challenges faced in developing a common component-based software architecture for these stakeholders. It will develop and document architectural principles and design strategies used in the creation of the architecture. Finally the research will provide a basis for future work in the areas of component-based architecture, common military and commercial software, and PC-based virtual simulation.

Phase I: Analysis

An analysis phase prior to the development of a software architecture is important because it provides a solid basis from which to make architectural decisions. These decisions should not be made on the basis of an architectural style, design patterns, or in a vacuum; they should be made on the basis of a direction provided by the limitations of a set of important, driving architectural issues. The analysis phase will provide a documented, traceable link from stakeholder requirements through the underlying problems the architecture must address to the architecture's development, implementation, and evaluation.

Hofmeister, Nord, and Soni (2000) present a process for architectural analysis based on a risk-driven approach which will be adapted for use in this research. They

outline an analysis phase that analyzes the risks, or *factors*, that could influence the software architecture. In this research a set of structured interviews will be used to identify those factors. Factors will be analyzed in order to generate a set of underlying issues that the architecture must address.

Structured Interviews

A set of interviews will be conducted with domain experts. Twelve experts from one of the following three categories will be interviewed: military PC-based virtual simulation (4), PC-based virtual simulation for commercial entertainment and gaming(4), and software architecture and software component technologies (4). Their responses will be used to identify the major risks facing the proposed architecture.

Questions will be formulated to bring to light the most significant issues facing the architecture in its various domains. They will be aimed at identifying the largest challenges and risks to the architecture based on the opinions of the experts. Interviews will be conducted face-to-face or over the phone and, if needed, questions will be adapted on-the-fly to suit the information received from the expert. Responses will be hand-recorded, or if possible, machine-recorded for later review.

The following questions will be used for the experts from each of the respective categories:

Military PC-based virtual simulation:

1. How many years of experience have you had working with PC technologies and PC-based simulation in the military?
2. What types of PC-based simulation projects have you led or worked on?

3. What major issues or drawbacks have you encountered using PC-based technologies in the military?
4. What do you consider to be the most significant risks to using PC simulation to achieve military objectives?
5. What major strengths and advantages have you encountered using PC-based technologies in the military?
6. What advice and recommendations do you have for someone using PC simulation for military objectives (training, education, communication, analysis, etc)?
7. Are you familiar with the concept of a common software architecture for PC-based simulation?
8. How would you define a common software architecture for PC-based simulation?
9. Are you familiar with any common software architectures for PC-based simulation?
10. Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?
11. Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?
12. Are those components the most easy to reuse?
13. Are there components that are more difficult to reuse? If so, what are those components?

14. What do you consider to be the most significant risks to using a common software architecture for PC simulation to achieve military objectives?
15. What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?

PC-based virtual simulation for commercial entertainment and gaming:

1. How many years of experience have you had working with PC technologies and PC-based simulation in the gaming industry?
2. What types of PC-based simulation projects have you led or worked on? What type of development environment? What type of software architecture?
3. What are the largest challenges facing game developers and game development companies today?
4. Have you used any component technologies in PC game development projects? If so, explain.
5. Are you familiar with the concept of a common software architecture for PC-based simulation?
6. How would you define a common software architecture for PC-based simulation?
7. Are you familiar with any common software architectures for PC-based simulation?
8. Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?

9. Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?
10. Are those components the most easy to reuse?
11. Are there components that are more difficult to reuse? If so, what are those components?
12. What experience have you had using military technologies for gaming? What are the most significant business and technical risks involved in doing this?
13. What types of military technologies would you like to see in a PC simulation game?
14. What do you see as the greatest business and technical risks for using PC simulation and gaming technology in the military?
15. What advice and recommendations do you have for someone creating a single framework for PC gaming and military simulation
16. What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?

Software architecture and component software technologies:

1. How many years of experience have you had working with component-based software engineering (CBSE) and component based software architectures (CBSA)?
2. What types of CBSE and CBSA projects have you led or worked on? Have you worked on any gaming or military CBSE projects?
3. What major issues or drawbacks have you encountered using CBSE?

4. What do you consider to be the most significant risks to using CBSE?
5. What major strengths and advantages have you encountered using CBSE?
6. Are you familiar with the concept of a common software architecture for PC-based simulation?
7. How would you define a common software architecture for PC-based simulation?
8. Are you familiar with any common software architectures for PC-based simulation?
9. Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?
10. Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?
11. Are those components the most easy to reuse?
12. Are there components that are more difficult to reuse? If so, what are those components?
13. What do you consider to be the most significant risks to using a common component-based platform for PC simulations in both gaming and the military?
14. What advice and recommendations do you have for someone creating a common component based architecture for gaming and for military simulation needs?

Factor Identification

Once the interviews have been completed, the experts' answers will be analyzed to produce the main factors they think will affect the architecture. There are three steps involved in discovering and analyzing the factors:

1. *Identify and describe the factors:* Document those factors that have a driving global influence on the architecture, those that could change during development, and those that are difficult to accomplish.
2. *Categorize the factors:* Factors will fall into one of three categories:
    - *Organizational factors:* Organizational factors are hose factors relating to the development and customer organizations that might include schedule, budget, attitudes, culture, software process, required standards, or business development direction.
    - *Technological factors:* Technological factors are those factors relating to the hardware, software, deployment environment, tools, or other technologies available for use.
    - *Product factors:* Product factors are those factors relating to the functional (what it will do) and non-functional (performance, maintainability, dependability, etc) requirements of the delivered system that have been identified or assumed.
3. *Characterize factor flexibility:* Identify how likely the factor is to change over the course of development and how much the factor can be influenced to change by the architect.

4. *Analyze each factor's impact:* Identify those parts of the architecture that are affected by the factor or changes to the factor.

Each factor will be recorded in the following format:

Table 1: Factor Recording Template

<No.>	Name:	<Factor Name>
	Type:	<Organizational, Technological, Product>
	Description:	<Factor Description>
	Flexibility:	<What aspects of the factor are flexible and changeable?>
	Impact:	<Components affected by the factor or changes to it>

Issue Generation

Once the set of influencing factors has been identified, a set of issues derived from those factors will be generated. An issue is a single problem that arises based on a factor or set of factors and must be explicitly addressed by the architecture.

Issues will be recorded in the following format:

Table 2: Issue Recording Template

<No.>	Name:	<Issue Name>
	Description:	<Issue Description>
	Influencing factors:	<List of factors that affect this design issue>

This table will be expanded in the design phase to include specific design solutions and architectural strategies that will address each issue.

Phase II: Design and Documentation of the Architecture

The software architecture development effort is typically documented in a number of artifacts that represent the architectural description. Kruchten (1995) introduces Rational's popular "4+1" views of software architecture which include the use case, logical, process, development, and physical views. Other architectural views include the data, execution, module, code, functional, structural, and deployment views. While many possible views and corresponding notations exist for the description of architectural concepts, there are currently no architectural description standards, notations, or languages that have been widely accepted (Clements, Kazman, Klein 2002; Clements 2005).

The architectural view that will be used in this research is one developed by Hofmeister et. al. (2000) called the conceptual view. The conceptual view is an ideal one for describing a component-based architecture because it is documented solely through the use of components and connectors. While the terms *component* and *connector* are broadly used in the context of software architecture, they are defined narrowly in the conceptual view. Components and connectors will be created and joined based on strategies developed from issues identified in the analysis phase.

Solution and Strategy Development

For every issue identified in the analysis phase, a corresponding strategy will be developed to account for the influence and impact of the documented factors. There are three steps involved in developing strategies.

1. *Develop design solutions:* A solution represents the decision to use a general design pattern, approach, or technique to resolve a particular issue.

2. *Develop architectural strategies:* A strategy is the specific architectural implementation of a solution that addresses an issue and mitigates or localizes the impact of the set of related factors.
3. *Identify related strategies:* Related strategies are those strategies that may be similar to, affect, or are affected by the strategy at hand.

Each issue table will be expanded to include its corresponding design solution and architectural strategy as follows:

Table 3: Expanded Issue Recording Template

<No.>	Name:	<Issue Name>
	Description:	<Issue Description>
	Influencing factors:	<List of factors that affect this design issue>
	Design solution:	<Discussion of the general solution to the design issue>
	Architectural strategy:	<Explanation of the strategy>
	Related strategies:	<References to related strategies and a description of how they are related>

Every strategy will drive decisions on the selection of component types, component contents, and the level of component decomposition. These strategy-based decisions are an important part of the architectural development process because they will provide a documented link between the problem space and the design solution. Because each component's contents and logical boundaries will be determined by a documented strategy, each will help resolve at lease one identified issue.

Component Identification

A component is an independently executing bundle of functionality that has a peer relationship with other components. Components are independently executing in that they have no dependencies on other components in the architecture in order to operate. They encapsulate all aspects of the effort required to perform some function. Each component has one or more ports that define its interface to the rest of the architecture.

The UML meta model describing a component is given as:

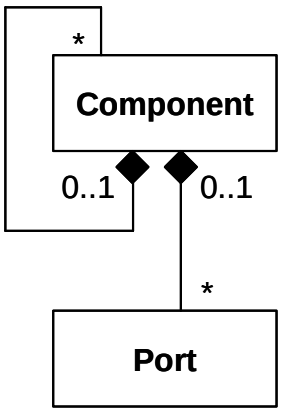

Figure 2: UML Component Model

Components will be identified using several methods. The easiest way to identify components in the context of virtual simulation is to identify required domain-specific functionality. Components will also be identified by decomposing broadly-functioning components into those that encapsulate a specific subset of that functionality. They will also be chosen based on their capability to be reused across multiple applications.

Connector Identification

A component's communication path to other components and the rest of the architecture is defined through connectors. Components (unlike objects in object-oriented design) do not exhibit a "provides" or "uses" relationship with each other; they

exhibit an independent peer-to-peer relationship. Connectors encapsulate the data, events, and control information passed in and out of a component to make it function and deliver its results. Connectors provide the controlling influence on the functionality provided by components. Like a component's port, a connector's role defines its interface to the rest of the architecture

The UML meta model describing a connector is given as:

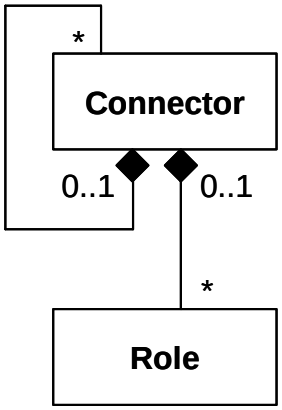

Figure 3: UML Connector Model

Component-Connector Relationships

The UML meta model describing the relationship between components and connectors is given as:

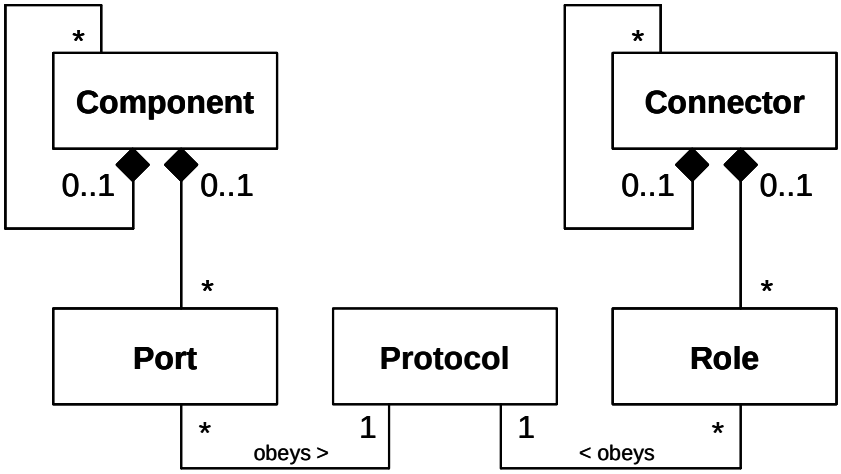

Figure 4: UML Component - Connector Relationship Model

Component ports and connector roles define the communication bridge between a component and connector. Ports and roles are created to meet some specification or protocol. As long as the protocol is shared, a given component port and given connector role have the ability communicate.

This research aims to describe a common component-based architecture for military and commercial PC-based virtual simulation through the conceptual architectural view. The majority of the architectural description will consist of a number of arrangements of components and connectors, their associated ports and roles, and the related protocols. Accompanying the depictions of the conceptual view will be explanations detailing how the architectural decisions made in the global analysis phase have been implemented.

Phase III: Implementation of Prototypes

The intent of the implementation phase is to provide a basis for an evaluation of the component-based architecture. To verify the characteristics and traits of the software architecture, two prototype software applications will be implemented that will conform to the architecture designed and documented during the previous phase. The prototypes will be built to test the hypothesis, presented in Chapter 2, that a common component-based software architecture for PC-based virtual simulation will:

- provide a common platform to create entertainment and military solutions,
- support reusable software components across varying simulations and domains,
- and allow for interchangeable simulation software components.

Following is a description and list of requirements for each of the prototypes:

Requirements for Prototype 1: Putt-putt

Putt-putt is a game that simulates a larger-than-life-size miniature golf course. Each hole in the course will consist of items and a layout similar to a real miniature golf course hole (e.g. tee, sidewalls, slopes, obstacles, greens, hole). Instead of using a golf club and golf ball, players will control a vehicle which must be used to direct a large beach ball into the hole. Instead of a par based on a number of shots, par for each hole will be based on the time required to get the beach ball from the tee to the hole.

Table 4: Prototype 1 Requirements

No.	Requirement
1	Putt-putt will incorporate a vehicle with a physics model similar to a dune-buggy.
2	Putt-putt will use a beach ball with a realistic beach ball physics model.
3	Putt-putt will use the following obstacles: barriers and trees.
4	The Putt-putt prototype will implement 1 hole which will start with the beach ball on a tee and end when the player has directed the ball to the hole with his vehicle.
5	Putt-putt will display the current time that the player has spent on the hole.
6	Putt-putt will display the par time for the hole.
7	Putt-putt will provide the user with a 3rd person 3D view that follows behind the vehicle.

Requirements for Prototype 2: Pac-Bot Trainer

Pac-Bot Trainer (PBT) is a military training simulation for IRobot, a robotics company that supplies robots for the U.S. military. These robots are used for remotely controlled exploration, audio and video capture, and improvised explosive device (IED) detonation. They have been used extensively in Afghanistan and Iraq. The training simulation will allow a user to simulate controlling one of these robots remotely.

Table 5: Requirements for Prototype 2

No.	Requirement
1	PBT will incorporate a vehicle with a track-based physics model, an on-board video camera, and an appendage to manipulate objects.
2	PBT will incorporate a user control station with robot motion controls (forward, back, left, right), appendage controls (grasp, let go), and robot video camera display.
2	PBT will use the following obstacles: buildings and trees.
3	The PBT prototype will implement 1 session that will require a user to (1) drive the robot to a remote object and pick it up, (2) remove the remote object, and (3) return to the robot's starting position.
4	Each PBT session will start by listing the session's training objectives for the user and end when the user has completed the training objectives.
5	PBT will display the current training objective and instructions on how to achieve the objective.
6	At the end of the session, PBT will display the time required to achieve each training objective.
7	PBT will provide the user with a 1st person 3D view that simulates the display, position, and orientation of the robot's video camera

Architectural Requirements for the Prototypes

The prototypes have been chosen to prove the architecture's characteristics and push its limits. To that end, an additional set of requirements will be imposed on the prototypes that will be used in the Evaluation phase to ensure that the architecture has met its goals.

Table 6: Architectural Requirements for the Prototypes

No.	Architecture Goal	Requirement
1	The architecture will provide a common platform to create entertainment and military solutions.	Both Putt-putt and PBT will be documented to conform to the software architecture description and specification created in Phase II

No.	Architecture Goal	Requirement
2	The architecture will support reusable software components across varying simulations and domains.	Putt-putt and PBT will share at least the following software components: camera motion component camera display component course description component static obstacle behavior component physics component
3	The architecture will allow for interchangeable simulation software components.	Putt-putt and PBT will both incorporate the following interchangeable software components: low-fidelity turf component high-fidelity grass component

The prototypes' implementations of these requirements will be analyzed in the next phase to ensure that they have been met and the architecture has achieved its goals.

Phase IV: Evaluation

An earnest evaluation of the software architecture is important because it determines whether the architectural effort has met its goals. The evaluation verifies that the architecture has addressed the risks or factors imposed on it. The evaluation also validates the design decisions behind the architecture, ensuring the appropriate quality attributes are supported and good design practices are observed.

The evaluation phase will consist of three steps. Each step will evaluate the prototypes developed in the previous phase against a successively higher order of objectives. In the first step the prototypes will be verified against their original requirements. The second step will ensure that each of the strategies developed in the Analysis phase had a direct or indirect impact on the design and implementation of the prototypes. Finally, and most importantly, the prototypes (and thus the architecture) will

be validated against the original research objectives to ensure those objectives have been met.

Step 1: Verification of the Prototypes

In order to verify that Prototype 1 was built to specifications, a new column will be added to its requirements table that documents whether each of its requirements has been met. Once the prototype has been completed, the prototype will be verified against each requirement as follows:

Table 7: Prototype 1 Requirements Verification Template

No.	Requirement	Met? (Y/N)
1	Putt-putt will incorporate a vehicle with a physics model similar to a dune-buggy.	
2	Putt-putt will use a beach ball with a realistic beach ball physics model.	
3	Putt-putt will use the following obstacles: barriers and trees.	
4	The Putt-putt prototype will implement 1 hole which will start with the beach ball on a tee and end when the player has directed the ball to the hole with his vehicle.	
5	Putt-putt will display the current time that the player has spent on the hole.	
6	Putt-putt will display the par time for the hole.	
7	Putt-putt will provide the user with a 3rd person 3D view that follows behind the vehicle.	

In order to verify that Prototype 2 was built to specifications, a new column will be added to its requirements table that documents whether each of its requirements has been met. Once the prototype has been completed, the prototype will be verified against each requirement as follows:

Table 8: Prototype 2 Requirements Verification Template

No.	Requirement	Met? (Y/N)
1	PBT will incorporate a vehicle with a track-based physics model, an on-board video camera, and an appendage to manipulate objects.	
2	PBT will incorporate a user control station with robot motion controls (forward, back, left, right), appendage controls (grasp, let go), and robot video camera display.	
2	PBT will use the following obstacles: buildings and trees.	
3	The PBT prototype will implement 1 session that will require a user to (1) drive the robot to a remote object and pick it up, (2) remove the remote object, and (3) return to the robot's starting position.	
4	Each PBT session will start by listing the session's training objectives for the user and end when the user has completed the training objectives.	
5	PBT will display the current training objective and instructions on how to achieve the objective.	
6	At the end of the session, PBT will display the time required to achieve each training objective.	
7	PBT will provide the user with a 1st person 3D view that simulates the display, position, and orientation of the robot's video camera	

Step 2: Evaluation of Strategy Implementation

Because the architecture was built based on a set of strategies developed from stakeholder input, it is important that the implementation of the strategies be verified. Each strategy was documented in the Analysis phase, and each will be verified by documenting its impact on the two prototypes. The original strategies table will be expanded as follows:

Table 9: Strategy Evaluation Template

<No.>	**Name:**	**<Issue Name>**
	Description:	<Issue Description>

<No.>	Name:	<Issue Name>
	Influencing factors:	<List of factors that affect this design issue>
	Design solution:	<Discussion of the general solution to the design issue>
	Architectural strategy:	<Explanation of the strategy>
	Related strategies:	<References to related strategies and a description of how they are related>
	Effect:	<How the implementation of this strategy directly affected the implementation of Putt-putt and PBT>

Step 3: Validation of the Architecture

In order to ensure that the architecture has met the original research objectives, it is necessary to show how each of the prototypes has met those objectives. This will be done by ensuring that the prototypes have met the architectural requirements specified in Phase III in the following tables:

Table 10: Architecture Validation Template Requirement 1

No.	Architecture Goal	Requirement	Met? (Y/N)
1	The architecture will provide a common platform to create entertainment and military solutions.	Both Putt-putt and PBT will be documented to conform to the software architecture description and specification created in Phase II	
<Description>			

Table 11: Architecture Validation Template Requirement 2

No.	Architecture Goal	Requirement	Met? (Y/N)
2	The architecture will support reusable software components across varying simulations and domains.	Putt-putt and PBT will share at least the following software components: camera motion component camera display component course description component static obstacle behavior component physics component	
<Description>			

Table 12: Architecture Validation Template Requirement 3

No.	Architecture Goal	Requirement	Met? (Y/N)
3	The architecture will allow for interchangeable simulation software components.	Putt-putt and PBT will both incorporate the following interchangeable software components: low-fidelity turf component high-fidelity grass component	
<Description>			

Contribution of the Research

This research represents the implementation of a new strategy for creating PC-based virtual simulations for military and commercial use. It is based on the analysis of the priorities and goals of entertainment and military industry stakeholders. This strategy for common use provides a solution for the drawbacks encountered through other common-use strategies like reuse, contracted development, and adaptation.

A new component-based software architecture will be developed that, when implemented, will provide a generic platform from which application-specific virtual

simulation solutions can be created. It will be created systematically and documented through notation presented in the literature.

A new framework will be created that could be used as the basis for future work. It will be capable of supporting future virtual simulations of many types and will be capable of incorporating new simulation-related technologies. It will provide opportunity and direction for future research in component-based military and commercial virtual simulation.

4. RESULTS

This chapter presents the results gathered and documented from executing the process outlined in the previous chapter. Phase I Results includes the write-ups from interviews with twelve experts, and it documents the risks and issues they identified that the architecture must address. The design decisions and architecture description diagrams are documented in Phase II Results. Phase III Results describes the architecture implementation and prototype development. The evaluation of the prototypes is presented in Phase IV Results as a verification that the implemented architecture exhibits the characteristics required to meet its original objectives.

Phase I Results: Analysis

In the Analysis Phase twelve interviews were conducted with experts in the fields of military virtual simulation, gaming, and component-based software architecture. Analysis of the interview results produced a list of factors that would affect the type of architecture created in this project. The factors were grouped together to produce a set of fundamental issues that the architecture would need to address.

Interviews

The following twelve experts were interviewed:

Group 1: Experts in Software Architecture and Component Software Technologies

- Didi Garfunkel, Simigon Inc.
- Darren Humphrey, Disti Inc.
- Robert Norton, Thoughtworks Inc.
- Dr. Clemens Szyperski, Microsoft Inc.

Group 2: Experts in Military PC-Based Virtual Simulation

- Curtis Conkey, NAVAIR
- Peter Smith, NAVAIR
- Dr. Roger Smith, Sparta Inc.
- Dr. Michael Zyda, ISI at USC

Group 3: Experts in Virtual Simulation for Commercial Entertainment and Gaming

- Tom Carbone, FIEA
- Stephen Eckman, Disti Inc.
- Dr. Michael Gourlay, FIEA
- Keelan Stuart, Disti Inc.

The full write-ups for each of the interviews can be found in Appendix A.

Factors

An analysis of the interviews produced the following list of factors that the experts believed would provide risk to the architecture or would affect its design.

1. Leveraging middleware
2. Competitive advantage
3. Product line reuse
4. Black box component use
5. Confidentiality of military technology in games
6. Differing gaming and military content shelf life
7. Differing gaming and military content quality
8. Lack of science behind military gaming technology
9. Differing gaming and military content objectives

10. Training objectives drive technology choices
11. Differing gaming and military content fidelity
12. Increasingly realistic gaming graphics
13. Tie-in to learning management system
14. Increasing game budgets and team sizes
15. Component reuse difficulties: different purpose and different interface
16. Component reuse difficulties: close ties to domain and context
17. Differing gaming and military content optimization
18. Backwards compatibility and version upgrades
19. Component engineering effort
20. Component performance
21. Component framework complexity
22. Component reuse difficulties: many dependencies
23. Legacy code integration
24. Domain model componentization
25. Development in a vacuum or lab environment
26. Gaming interoperability
27. Built-in assumptions of a generic platform
28. Military is averse to risky new technologies
29. Lack of originality in serious games
30. Divergence of technology
31. Abstract over-engineering
32. Self-driven components

33. Security in the PC environment

34. Component reuse difficulties: interface complexity

35. Component protection and licensing

36. Emergence of dedicated physics cards

The full description of each factor including categorization, characterization, and analysis of impact can be found in Appendix B.

Issues

Similar factors were grouped together to help identify the fundamental issues that the architecture must address. Following is the list of issues that were identified:

1. Adoption of a component-based architecture
2. Market forces facing game studios
3. Differences between gaming and military content
4. Support for military training
5. Component reuse
6. Component architecture development
7. Component framework implementation
8. Security and military technology
9. Technology trends

A full description of each issue and its associated influencing factors can be found in Appendix C.

Phase II Results: Design and Documentation of the Architecture

The design and documentation of the architecture presents the results of the effort to create a resolution to the issues identified in Phase I. Solutions and design strategies are identified and the architecture's components and connectors are defined.

Architecture Design: Solutions and Strategies

For each issue identified in the analysis phase, a general architectural solution was identified that would be used to resolve the issue or mitigate its impact. For each solution, one or more specific design strategies was developed that would help define the structure of the architecture.

The proposed solutions to each of the numbered issues along with associated design strategies are as follows:

1. Adoption of a component-based architecture

   Solution: Make integration with the framework simple and encourage componentization, but do not enforce it. Provide a dedicated infrastructure for the use of non-componentized libraries.

   Strategy: Use one or more framework components dedicated to interfacing with non-componentized code.

2. Market forces facing game studios

   Solution: The architecture will support customization of infrastructure and 3rd party components to help resolve scalability issues and allow companies to maintain distinction of content.

   Strategy: Use configuration-based component customization.

   Strategy: Use a replaceable event manager.

Strategy: Use tailored events.

3. Differences between gaming and military content

   Solution: Allow for the difference between gaming and military content, but minimize the impact of replacing content and minimize content dependencies.

   Strategy: Separate content components from framework components.

4. Support for military training

   Solution: Use an infrastructure that supports the requirements for logging, playback, and learning management system tie-in required by military training systems. Encapsulate risky technology in separate components.

   Strategy: Use persistent events.

5. Component reuse

   Solution: Create a component interface that is simple, flexible and negotiable.

   Strategy: Use an event-based component interface.

   Strategy: Use configurable event data.

6. Component architecture development

   Solution: Handle protection, licensing, and versioning together. Support individual component licenses. Ensure only one version of a component is active at a time but allow version negotiation and replacement.

   Strategy: Use registration and licensing managers.

7. Component framework implementation

   Solution: Ensure that the architecture supports the major virtual simulation domain models currently in use.

Strategy: Componentized virtual simulation domain models.

8. Security and military technology

   Solution: Implement basic security policies to help counteract malicious use of the infrastructure.

   Strategy: Implement component interface constraints.

9. Technology trends

   Solution: Encapsulate new and diverging technology to help mitigate the risks of using it.

   Strategy: Wrap risky technologies in components.

A full description of each solution, each solution's design strategies, and related strategies can be found in Appendix C.

Architecture Documentation: Components

This section provides a brief description of the components designed for the architecture based on the strategies developed in this phase.

Component Types and Responsibilities

Three types of components were designed for the architecture: infrastructure components, framework components, and content components. Their assigned responsibilities are listed below.

- *Infrastructure:* These components provide the underlying core functionality of the simulation. They are responsible for component licensing, registration, and configuration as well as simulation, event, and time management.
- *Framework:* These components provide non-entity-based behaviors during the simulation. Each framework component is run as a singleton per scenario.

Examples of framework components include the physics engine, graphics engine, collision detection, environment control, camera views, terrain engine, simulation tools, and instructional design tools.

- *Content:* These components provide entity-based behaviors during the simulation. Each entity is an aggregation of content components. Examples of content components include the entity motion model, lifecycle model, damage model, instrument displays, external displays, and subsystem models.

Component Lifecycles

Each type of component has a distinct lifecycle. The description of each component type's lifecycle is listed below.

- *Infrastructure:* Infrastructure components are loaded and instantiated at the beginning of execution and are destroyed and unloaded at the end of execution.
- *Framework:* Framework components are loaded and instantiated before the beginning of a scenario and are destroyed and unloaded after the end of a scenario.
- *Content:* Content components are loaded at the beginning of a scenario and are instantiated at entity creation. They are destroyed and unloaded after the end of a scenario.

Component Administration

A number of administration procedures must be performed on each component before it can be used. Following is the list of those procedures:

- *Licensing:* Since each component is licensed individually, a licensing mechanism must be in place to ensure that a valid license exists before the component is used. This functionality is provided by the LicenseMgmt infrastructure component.
- *Registration:* Each component must be registered to ensure each component's interface is compliant with the simulation and allow the enforcement of component interface constraints. This functionality is provided by the RegistrationMgmt infrastructure component.
- *Configuration:* The configuration mechanism provides initial states for all components and is used to define entity and scenario characteristics. This functionality is provided by the ConfigurationMgmt infrastructure component.

A full description of each type of component and its lifecycle including class and sequence diagrams can be found in Appendix D. A full definition of each type of component can be found in Appendix E.

Architecture Documentation: Connectors

This section provides a brief description of the connectors designed for the architecture based on the strategies developed in this phase.

Connector Types and Responsibilities

Two types of components were designed for the architecture: infrastructure component reference and events. Their assigned responsibilities are listed below.

- *Infrastructure Component Reference:* This connector is used for infrastructure component peer-to-peer communication. Each infrastructure component communicates through an interface that conforms to a

predetermined interface contract for that component. Each infrastructure component can interface to every other infrastructure component based on knowledge of its contract.

- *Persistent Events:* This connector is used for framework and content component peer-to-peer communication. These components communicate through publishing and subscribing to events that conform to a predetermined event contract. Each content and framework component can interface to every other content and framework component based on the agreement of an event's name and data structure.

Connector Lifecycles

Each type of connector has a distinct lifecycle. The description of each connector's lifecycle is listed below.

- *Infrastructure Component Reference:* The infrastructure component reference connector is established by the execution layer and passed to each infrastructure component at the beginning of execution. It is destroyed at the end of execution.
- *Persistent Events:* Each event or associated callback is instantiated when its owner component is instantiated. When an event is published each of its associated callbacks is run and the event and its data is time-stamped and stored for access by any of its subscribing components. The event and its data is persistent until superseded by a newer event of the same name. Events are destroyed after the end of a scenario.

Simulation Administration

A number of administration procedures must be performed by the simulation to ensure that the components can communicate through the connectors. Following is the list of those procedures:

- *Event Management:* The event management mechanism controls the event communication process for all simulation components. It matches registered event publishers to registered event subscribers. This functionality is provided by the EventMgmt infrastructure component.
- *Time Management:* The time management mechanism maintains current simulation time. This functionality is provided by the TimeMgmt infrastructure component.
- *Simulation Management:* Initializes, runs, and terminates each scenario through the instantiation and destruction of all simulation components and the control of the EventMgmt and TimeMgmt infrastructure components. This functionality is provided by the SimulationMgmt infrastructure component.

A full description of each type of connector and its lifecycle including class and sequence diagrams can be found in Appendix D. A full definition of each type of connector can be found in Appendix E.

Phase III Results: Implementation Of Prototypes

This section provides details on the development and run-time environment of the implemented prototypes. It provides concrete specifications relating to how the implementation of the architecture directly mapped to the architecture description.

Finally it gives a brief description of the components that were developed and how they were used in Prototypes 1 and 2.

Implementation Environment Details

The implementation environment for the architecture and prototypes was chosen based on its ability to allow extensive work on architectural implementation details while requiring minimal work to meet the requirements of the prototypes.

- *Torque Game Engine:* The Torque Game Engine (TGE) v1.4 by Garage Games provides a software development kit targeted at low budget games. All of the C++ source code is provided for the game engine allowing modifications as required. TGE also ships with a set of basic content and scenarios that can be modified and reused as needed. TGE uses its own scripting language called TorqueScript that can be used to define content and scenario behaviors. Torsion is a free program that was used to modify and debug those scripts.
- *Windows XP:* The Windows operating system was chosen as a platform for the implementation because many PC-based games and virtual simulations are currently built for Windows and TGE runs natively on Windows.
- *Visual Studio .NET 2003:* Visual Studio .NET 2003 was used as the development environment for the implementation. It was used to manipulate and build TGE, implement the architecture, and create all the components for the prototypes.

- *Hardware:* The hardware used to create and test the implementation was a Dell XPS with a Pentium IV 3.4 GHz processor, 1 GB RAM, and an ATI RADEON 9700 video card with 128 MB video RAM.

Architecture Implementation

The component-based architecture was implemented according to the architecture description, specifications, and modeling diagrams provided in Phase II.

- *Execution Layer:* The runtime environment for the component-based virtual simulation was provided by TGE. TGE-based TorqueScript was used to initialize, execute, and terminate the simulation.
- *Infrastructure Layer:* The six infrastructure components specified in the architecture description were implemented and named CS_EventManagement, CS_LicenseManagement, CS_RegistrationManagement, CS_TimeManagement CS_ConfigurationManagement, and CS_SimulationManagement. Each infrastructure component was implemented as a Windows DLL (dynamic loading library) and loaded at run-time by the execution layer.
- *Framework Layer:* Five framework components were created for use by the prototypes, implemented as Windows DLLs, and loaded at run-time by the infrastructure component CS_SimulationManagement:
  - *FC_MissionDataLoader:* Framework component responsible for loading Torque mission data for the scenario (name, description, object locations, scenario layout, terrain file, etc).

- *FC_ODEComponent:* Framework component that encapsulates the Open Dynamics Engine physics engine that can be used for adding physics-based behavior to objects.
- *FC_FoliageLowFid:* Framework component that provides data for representing low fidelity foliage. It is interchangeable with FC_FoliageHighFid.
- *FC_FoliageHighFid:* Framework component that provides data for representing high fidelity foliage. It is interchangeable with FC_FoliageLowFid.
- *FC_Torque_Component:* Framework component that provided the interface between the component-based simulation and Torque. It is responsible for turning Torque-based function calls into publishable events and turning event subscription callbacks into data accessible by Torque.

- *Content Layer:* Three content components were created for use by the prototypes, implemented as Windows DLLs, and loaded at run-time by the infrastructure component CS_SimulationManagement:
  - *CC_StaticBehavior:* Content component that provides static behavior to each entity that is attached. Responsible for broadcasting position and object boundaries.
  - *CC_CameraMotionCtrl:* Content component that provides camera motion control to each camera entity to which it is attached by receiving keyboard inputs and camera mode events and sending events to change the camera's perspective.

- *CC_CameraDisplayCtrl:* Content component that provides camera display data for each camera entity to which it is attached that is used for rendering the camera perspective for the player or trainee in the virtual simulation.

Prototype 1 Implementation

Putt-putt is a game that simulates a larger-than-life-size miniature golf course. Each hole in the course consists of items and a layout similar to a real miniature golf course hole (e.g. tee, sidewalls, slopes, obstacles, greens, hole). Instead of using a golf club and golf ball, players control a vehicle which must be used to direct a large beach ball into the hole. Instead of a par based on a number of shots, par for each hole is based on the time required to get the beach ball from the tee to the hole.

A screenshot of Prototype 1 is shown below:

Figure 5: Screenshot of Prototype 1

Prototype 1 makes use of the following components:

- *FC_MissionDataLoader*: loads data required to run scenarios
- *FC_ODEComponent*: provides the physics model for the beach ball
- *FC_FoliageLowFid*: provides low fidelity static turf
- *FC_FoliageHighFid*: provides high fidelity swaying grass
- *FC_Torque_Component*: interfaces with Torque
- *CC_StaticBehavior*: controls position of obstacles
- *CC_CameraMotionCtrl*: controls motion of the main camera
- *CC_CameraDisplayCtrl*: controls display of the main camera

Prototype 2 Implementation

Pac-Bot Trainer (PBT) is a military training simulation for IRobot, a robotics company that supplies robots for the U.S. military. These robots are used for remotely controlled exploration, audio and video capture, and improvised explosive device (IED) detonation. They have been used extensively in Afghanistan and Iraq. The training simulation allows a user to simulate controlling one of these robots remotely.

A screenshot of Prototype 2 is shown below:

Figure 6: Screenshot of Prototype 2

Prototype 2 makes use of the following components:

- *FC_MissionDataLoader*: loads data required to run scenarios

- *FC_ODEComponent*: provides the physics model for the crates and barrels
- *FC_FoliageLowFid*: provides low fidelity static turf
- *FC_FoliageHighFid*: provides high fidelity swaying grass
- *FC_Torque_Component*: interfaces with Torque
- *CC_StaticBehavior*: controls position of trees and buildings
- *CC_CameraMotionCtrl*: controls motion of main and robot cameras
- *CC_CameraDisplayCtrl*: controls display of main and robot cameras

Phase IV Results: Evaluation

The Evaluation phase documents whether the original research objectives have been realized. The prototypes are first verified against their original requirements, and then the design strategies developed in Phase II are evaluated for impact on the architecture and the prototypes. Finally the research is validated by assessing the implementation details of the prototypes against the original objectives for the architecture.

Step 1: Prototype Verification

Prototype 1 can be verified against each of its requirement as follows:

Table 13: Prototype 1 Verification

No.	Requirement	Met? (Y/N)
1	Putt-putt will incorporate a vehicle with a physics model similar to a dune-buggy.	Y
2	Putt-putt will use a beach ball with a realistic beach ball physics model.	Y
3	Putt-putt will use the following obstacles: barriers and trees.	Y
4	The Putt-putt prototype will implement 1 hole which will start with the beach ball on a tee and end when the player has directed the ball to the hole with his vehicle.	Y

No.	Requirement	Met? (Y/N)
5	Putt-putt will display the current time that the player has spent on the hole.	Y
6	Putt-putt will display the par time for the hole.	Y
7	Putt-putt will provide the user with a 3rd person 3D view that follows behind the vehicle.	Y

Prototype 2 can be verified against each of its requirement as follows:

Table 14: Prototype 2 Verification

No.	Requirement	Met? (Y/N)
1	PBT will incorporate a vehicle with a track-based physics model, an on-board video camera, and an appendage to manipulate objects.	Y
2	PBT will incorporate a user control station with robot motion controls (forward, back, left, right), appendage controls (grasp, let go), and robot video camera display.	Y
2	PBT will use the following obstacles: buildings and trees.	Y
3	The PBT prototype will implement 1 session that will require a user to (1) drive the robot to a remote object and pick it up, (2) remove the remote object, and (3) return to the robot's starting position.	Y
4	Each PBT session will start by listing the session's training objectives for the user and end when the user has completed the training objectives.	Y
5	PBT will display the current training objective and instructions on how to achieve the objective.	Y
6	At the end of the session, PBT will display the time required to achieve each training objective.	Y
7	PBT will provide the user with a 1st person 3D view that simulates the display, position, and orientation of the robot's video camera	Y

Step 2: Strategy Implementation Verification

Because the architecture was built based on a set of strategies developed from stakeholder input, it is important that the implementation of the strategies be verified.

Each strategy documented in Phase II can be verified by documenting its impact on the implementation of the architecture and each of the prototypes.

- ***Strategy:*** Use one or more framework components dedicated to interfacing with non-componentized code.

  Impact: Created component FC_Torque_Component that was responsible for interfacing the component-based simulation with the non-componentized Torque environment.

- ***Strategy:*** Use configuration-based component customization.

  Impact: CS_ConfigurationManagement was created as an infrastructure component that allows simulation component customization through the storage and retrieval of a set of initial component configuration events.

- ***Strategy:*** Use a replaceable event manager.

  Impact: CS_EventManagement was created as an infrastructure component that is fully replaceable.

- ***Strategy:*** Use tailored events.

  Impact: ISimEvent does not specify any data type or size restrictions for events. As long as event data can be serialized and deserialized, events can contain any amount of any type of data.

- ***Strategy:*** Separate content components from framework components.

  Impact: Framework and content components both inherit SimulationComponent but they are treated differently. Framework components are configured per scenario and span the lifecycle of the scenario,

while content components are configured per entity and span the lifecycle of each entity.

- ***Strategy:*** Use persistent events.

  Impact: Once an event has been published the event and its is data available for continuous access through the registered event callback function GetLastEvent( ).

- ***Strategy:*** Use an event-based component interface.

  Impact: Simulation components only communicate through events. The event mechanism represents a simulation component's interface to the rest of the component-based simulation.

- ***Strategy:*** Use configurable event data.

  Impact: Since ISimEvent does not specify any data type or size restrictions, events can be configurable at compile time, pre-run-time, and run-time.

- ***Strategy:*** Use registration and licensing managers.

  Impact: CS_RegistrationManagement and CS_LicenseManagement are infrastructure components that are responsible for simulation component registration and licensing respectively.

- ***Strategy:*** Componentized virtual simulation domain models.

  Impact: The core functionality of the virtual simulation domain models is divided among the infrastructure components and is specifically represented by CS_TimeManagement, CS_SimulationManagement, CS_EventManagement, and CS_ConfigurationManagement.

- ***Strategy:*** Implement component interface constraints.

Impact: The published and subscribed events that make up a component's interface are strictly controlled by the event and registration managers. Every event published and subscribed to by a simulation component is verified by registration management to ensure that the component has permissions to publish or subscribe to that event.

- ***Strategy:*** Wrap risky technologies in components.

  Impact: Simulation components can be used to wrap any risky technologies that are used in the simulation. Examples would include components that wrap a physics engine, a graphics engine, and network technology.

Step 3: Architecture Validation

In order to ensure that the architecture has met the original research objectives, it is necessary to show how each of the prototypes has met those objectives. This is accomplished by ensuring that the prototypes have met the architectural requirements specified in Phase III.

Table 15: Architecture Validation Requirement 1

No.	**Architecture Goal**	**Requirement**	**Met? (Y/N)**
1	The architecture will provide a common platform to create entertainment and military solutions.	Both Putt-putt and PBT will be documented to conform to the software architecture description and specification created in Phase II	Y
Description Both prototypes have been documented in the Phase III Results to conform to the architecture described and documented in Phase II Results.			

Table 16: Architecture Validation Requirement 2

No.	Architecture Goal	Requirement	Met? (Y/N)
2	The architecture will support reusable software components across varying simulations and domains.	Putt-putt and PBT will share at least the following software components: camera motion component camera display component course description component static obstacle behavior component physics component	Y
Description Both prototypes make use of the following components: CC_CameraMotionCtrl (camera motion component) CC_CameraDisplayCtrl (camera display component) FC_MissionDataLoader (course description component) CC_StaticBehavior (static obstacle behavior component) FC_ODEComponent (physics component)			

Table 17: Architecture Validation Requirement 3

No.	Architecture Goal	Requirement	Met? (Y/N)
3	The architecture will allow for interchangeable simulation software components.	Putt-putt and PBT will both incorporate the following interchangeable software components: low-fidelity turf component high-fidelity grass component	Y
Description Both prototypes make use of the following components: FC_FoliageLowFid (low-fidelity turf component) FC_FoliageHighFid (high-fidelity grass component)			

5. CONCLUSION

This chapter draws to conclusion the research work and documented results of the previous chapters. A summary of the results of the completed research and its original contributions are presented. A set of limitations of the architecture as implemented in this research are presented and discussed. Finally a number of topics are identified for future research efforts in the component-based virtual simulation domain.

Summary of Results

This section provides a summary of the results obtained in each phase of the research and documents that the research objectives have been attained.

Phase I represented an analysis of the problem space relating to military and commercial use of component-based virtual simulation. Twelve experts were interviewed in the domains of software architecture and component software technologies, military PC-based virtual simulation, and virtual simulation for commercial entertainment and gaming. From these interviews a list of thirty-six factors were extracted which represented the set of highest risk items that would face a common PC-based software architecture for component-based virtual simulation. The isolated factors were categorized into nine major issues that the software architecture needed to address.

Phase II involved the design and documentation of the software architecture for the component-based virtual simulation. For each issue identified in Phase I a corresponding solution was developed whose purpose was to resolve the issue and help mitigate its associated factors. Each solution was supported by one or more specific design strategies that directly impacted the design and implementation of the software

architecture. Finally the software architecture was developed and then documented with UML diagrams and supporting text.

In Phase III the software architecture was implemented on a PC-based Windows platform in the run-time environment provided by the Torque Game Engine. Two prototype component-based virtual simulations, a simple game and a simple military training aid, were created based on the design strategies and architecture developed in Phase II. The prototypes shared a number of different types of components and also supported interchangeable components.

An evaluation of the implemented architecture and prototypes was conducted in Phase IV to ensure the original objectives of the research had been achieved. First, the prototypes were verified against their original requirements. Second, the implementation of each design strategy was analyzed to ensure each had a direct effect on the implemented architecture and prototypes. Third, the architecture was validated by comparing the results achieved with the prototypes against the original tenets of the thesis. Specifically, it was demonstrated that a common component-based software architecture for PC-based virtual simulation:

- provides a common platform to create entertainment and military solutions,
- supports reusable software components across varying simulations and domains,
- and allows for interchangeable simulation software components.

Original Contributions

A number of original contributions have been made by the research. This research represents the design and implementation of a new component-based software architecture for creating PC-based virtual simulations for military and commercial use. It is based on a new consolidated analysis of the priorities and goals of entertainment and military industry stakeholders. A new set of solutions and design strategies have been created and tested to meet these priorities and goals. Finally, a new framework has been implemented that is capable of supporting virtual simulations of many types and flexible enough to support incorporation of new simulation-related technologies.

Limitations of the Architecture Implementation

While the architecture has demonstrated that it is capable of meeting the objectives of this research, a number of topics have been identified that the architecture does not explicitly address, and a number of limitations are known that constrain this implementation. These topics have been identified both by the author and by other architecture experts that have reviewed the design documentation, and they are briefly discussed here:

- *Proof of concept only:* While architectural design decisions were made based on input from experienced gaming and military personnel, no attempt has been made to test the prototypes in their respective domains. Until the architecture has proven itself in the field it remains a proof of concept.
- *Immaturity of the infrastructure components:* A minimalist approach was taken in designing the infrastructure components with the intention of

developing only what was absolutely required for the component-based simulation to execute. Specifically a significant amount of work is needed on the registration management and license management components to support full component registration and licensing.

- *Inefficiencies:* The architecture currently demands component registration and license verification at every scenario execution on component initialization. While this would work, it unnecessarily extends scenario load times. The event manager currently allows the propagation of all events to all simulation components. It would be more efficient to scope event propagation as required.
- *Multiple simulation instances:* The current implementation does not support multiple instances of the component-based simulation running concurrently on a single personal computer.
- *Support for a continuous world:* There is currently no explicit support for the concept of a continuous world. This is an increasingly popular concept, especially in massively multiplayer games, that allows users to enter and leave a persistent virtual world which is simulated for an indefinite amount of time.
- *Scalability and aggregate models:* Advanced military training scenarios sometimes require thousands of entities with the ability to aggregate and de-aggregate them into platoons, companies, battalions, etc. and display different behavior patterns accordingly. No documentation was provided in the architecture description that would support this scale and class of simulation.

- *Architecture evolution:* Szyperski (1998) states that a component architecture must not only be defined but maintained to suit the evolution of its components. At this time no methodology has been identified to support the evolution and maturation of the architecture or its adaptation to different projects and development processes.
- *Tools:* While considerable thought was given to how the architecture would support simulation-oriented development tools, no attempt was made to develop tools that would aid in developing and integrating components or building a simulation-based application. Examples of such tools could include a component licensing tool, a component registration and interface negotiation tool, a scenario generation tool, and SCORM compliant courseware wrapping tools to name a few.
- *Deployment:* Little effort has been given to designing a deployment environment for the component-based virtual simulation. Significant effort would be required to create a commercially deployable framework with independently marketable components. Ideally deployment and installation would be simple for the end user and linked to a commercially viable business and marketing effort.

Future Research

There are several areas of interest in which future research could be conducted. The architecture should be tested in both a real commercial gaming and military training environment. While this research proved the concept of a common implementation for

component-based virtual simulation for a PC platform, it did not demand performance from the architecture as a real-world implementation would.

In a related area, the architecture, and specifically the event mechanism, should be stressed to analyze the types of loads it has the ability to handle. Currently it has proven it can support the simplest of games and training aids. However it would be interesting to see if it can support the behavior of hundreds or thousands of entities with complex interactions and behavior patterns.

Finally, with the recent prevalence of integrated online solutions, it would be beneficial to research the possibility of implementing a full client-server implementation of the architecture. Component-based virtual simulation has the potential to allow disparate components of a single application to be hosted on separate servers in different parts of the world while clients would be presented with a seamless virtual environment. It would be worthwhile to test out this distributed-component- based virtual simulation concept as it has the potential to alleviate many of the problems associated with software piracy, copyrighting, and licensing.

APPENDIX A: PHASE I RESULTS – INTERVIEWS

Group 1: Experts in software architecture and component software technologies

Interview with Didi Garfunkel, Simigon Inc.

1. *How many years of experience have you had working with component-based software engineering (CBSE) and component based software architectures (CBSA)?*

   7 years for Simigon and Pixel, before that doing network management.

2. *What types of CBSE and CBSA projects have you led or worked on? Have you worked on any gaming or military CBSE projects?*

   At Pixel we were working on a component-based civilian flight simulator, but that was never released. After that I led the architecture and software engineering effort for Simigon's Airbook. We developed component-based sims for the Israeli Air Force F-16 and F-15. We did an integration with the F-4 platform and also worked with Rafael doing a ship bridge simulation.

3. *What major issues or drawbacks have you encountered using CBSE?*

   Backward compatibility is a huge issue. When you create a framework with interfaces so that customers can build their own components, those interfaces have to be sustained through multiple versions of the product to maintain backward compatibility. Regarding that same issue, our product has some dependencies on 3rd party tools and libraries. When those tools go through upgrade cycles we have to adapt our components and help our customers adapt their components for the new toolset. CBSE requires significantly more effort than creating something monolithic. It's almost always easier to build one big block than to split it up into components and spend all that effort on

interfacing. There are also performance issues – loading potentially hundreds of components at run-time is not ideal, but it's something that is required for large complex simulations. Finally there are licensing issues. When you use an open architecture in a component-based software framework you are supporting components written by many people. All that content must be protected and licensed properly.

4. *What do you consider to be the most significant risks to using CBSE?*

   Complexity. CBSE by nature can add a lot of complication to a simulation. This is a risk especially when dealing with new developers and new customers.

5. *What major strengths and advantages have you encountered using CBSE?*

   Every software word with an "-ility". Reusability – you can take one software component and reuse it in many different environments. Maintainability – because components are created with a limited scope it's possible to maintain each component individually without affecting the whole framework and without changing interfaces. CBSE provides a flexible long-term solution – new pieces of functionality can always be added without re-designing the whole system. Also it allows 3rd parties like customers and subcontractors to work from a single platform and simulation environment that you developed. And it can allow for both classified and unclassified work in the same framework often using the same components.

6. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

Simigon's Airbook is the only one out there for military PC-based simulation. I'm not aware of one out there for gaming or entertainment. There are a lot of open architectures and gaming engines but they are not component-based.

7. *How would you define a common software architecture for PC-based simulation?*

   Not asked as this question was poorly worded and confusing to the experts.

8. *Are you familiar with any common software architectures for PC-based simulation?*

   CAE has an open architecture for PC-based simulation.

9. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   Yes – Airbook.

10. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    Yes, in the past we've taken gaming technologies and built them into components. We took software built for automated red air and build them into component-based flight models and behaviors.

11. *Are those components the most easy to reuse?*

    Components that have the least number of dependencies are the easiest to reuse.

12. *Are there components that are more difficult to reuse? If so, what are those components?*

Components that can affect lots of other components are less easy to reuse. One of the main reasons for this is that they require more complicated interfaces.

13. *What do you consider to be the most significant risks to using a common component-based platform for PC simulations in both gaming and the military?*

    Real-life training is very different from playing a game – there are different issues and different priorities. Games have to be fun. Training simulations should be the highest fidelity possible. If the fidelity isn't high enough you can get negative training. This would never be a problem for a game.

14. *What advice and recommendations do you have for someone creating a common component based architecture for gaming and for military simulation needs?*

    Anyone creating this type of platform needs to have very good domain knowledge and extensive experience in both gaming and military simulation. Also, keep the architecture as simple as possible – many simulations have unnecessary complication.

Interview with Darren Humphrey, Disti Inc.

1. *How many years of experience have you had working with component-based software engineering (CBSE) and component based software architectures (CBSA)?*

   10 years.

2. *What types of CBSE and CBSA projects have you led or worked on? Have you worked on any gaming or military CBSE projects?*

   I've worked on GLStudio at Disti and a product called ModIOS for Motorola before that.

3. *What major issues or drawbacks have you encountered using CBSE?*

   The organization you are doing it for might not necessarily be set up to do CBSE. On the Lockheed Martin MEADS program – in the simulation based acquisition part of the project – we tried fitting in legacy non-componentized systems into a component architecture. This was difficult.

4. *What do you consider to be the most significant risks to using CBSE?*

   Where you partition your data or domain model is a big risk. When you partition it into components you make assumptions that may or may not be true in the future. You're architecture must be explicitly designed around the domain you are working in, and you have to do a lot more analysis and engineering work up front to make it work.

5. *What major strengths and advantages have you encountered using CBSE?*

   CBSE gives you all the things object-oriented technologies were supposed to give you – encapsulation, modularity.

6. *Are you familiar with any common software architectures for PC-based simulation?*

   Yes, Zedasoft uses an architecture they call Container Based Architecture. There is also an architecture called the Common Simulation Framework. It

was originally designed for solving differential equations to facilitate missile path modeling and simulations, so it's very domain specific.

7. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   At Disti we have a product called GLStudio that is component-based and is used in both the commercial and military simulation worlds.

8. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

   Yes, we use GLStudio to develop reusable components for aircraft cockpit instruments.

9. *Are those components the most easy to reuse?*

   Types of components that are easiest to reuse are ones that are more context free and have well-defined interfaces – things like physics and math models. However you have to be careful about the fidelity differences required by different uses of the same components.

10. *Are there components that are more difficult to reuse? If so, what are those components?*

    Components that are harder to reuse are ones that interface with real world I/O connections – like grips and hardware connectors. These are so domain specific that you can't really use them in a different context.

11. *What do you consider to be the most significant risks to using a common component-based platform for PC simulations in both gaming and the military?*

    Getting someone to adopt it and use it. There are going to be a lot of incompatible components across different architectures. Getting them all to work with a specific framework will be difficult. You also have to look at the lack of robustness and fidelity in gaming technology – a lot of that code is optimized and pre-computed because much of the environment doesn't change. In a military simulation on the other hand, everything in the environment can change in real-time. Also, games are built for their entertainment value and do not meet many of the scientific metrics the military requires like display fidelity and refresh rate.

12. *What advice and recommendations do you have for someone creating a common component based architecture for gaming and for military simulation needs?*

    Look at existing architectures out there that can be adapted or reused. Don't create something from scratch if you don't have to because a lot of work has already been done.

Interview with Robert Norton, Thoughtworks Inc.

1. *How many years of experience have you had working with component-based software engineering (CBSE) and component based software architectures (CBSA)?*

    5 years

2. *What types of CBSE and CBSA projects have you led or worked on? Have you worked on any gaming or military CBSE projects?*

   .NET, J2EE. Shuttle Engineering Simulator 3 at NASA; F-16 Block 60 Pilot Trainer at LM Aero.

3. *What major issues or drawbacks have you encountered using CBSE?*

   Software reuse is the underlying motivation for CBSE. Will Tracz identifies 3 "Con's" of software reuse, Concept, Content, and Context. The last of these 3 often thwarts reuse efforts: A component taken out of it's initial conceptual, operational, and implementation context if often hard to get working in a different context without significant effort – effort that may exceed what it would have taken to develop the required functionality from scratch. So while it may seem theoretically plausible to reuse, for example, a landing gear model from one simulator in another, in practice the implementation context (required inputs; output format; functional requirements, etc) may be different enough to make reuse impossible.

4. *What do you consider to be the most significant risks to using CBSE?*

   Any component to which the original source code is not available presents significant risks to a project. Using a black box component could introduce errors, security holes, or a performance bottleneck, and these problems will be difficult to correct without the ability for the integrator to review the component's source code.

5. *What major strengths and advantages have you encountered using CBSE?*

Reuse, despite its elusiveness, is still the biggest motivator and advantage to CBSE. When a component has been thoroughly tested and documented and its operation is well understood and clearly defined, the component will not have to be developed from scratch, thus avoiding problems such as inserting faults. Frameworks such those found in .NET and J2EE are perhaps the best example of widespread component reuse.

6. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

   Yes, in relation to constructive simulations of aircraft.

7. *How would you define a common software architecture for PC-based simulation?*

   I will shape this answer in relation the constructive flight simulations I've seen. An executive component handles frame sequencing, event handling, and synchronizing. The executive gets input events from sensor components and provides input to model components, such as aero, thermo, guidance, weapons, and landing gear. These models handle physics and data generation for their area of responsibility. A common simulation architecture would define standard interfaces and contracts for executive, sensor, and model components, along with a standard means of input and output. This would probably extend within a certain domain of simulation, i.e. aircraft simulation or four-wheeled vehicle simulation.

8. *Are you familiar with any common software architectures for PC-based simulation?*

The one I used at NASA was known as Trick. In the SES3 project at NASA, Trick provided a run-time executive that supported real time and faster then real time HIL simulations. I'll also provide an introduction to Bill Othon at the engineering directorate of JSC via email this week: He can give you extensive knowledge of Trick. Also look up Brian Hoelscher in the LM directory and send me his email address, and I'll do the same intro. He's a Staff Aero Eng at LM Space Operations who's been developing space sims for the past 15 years – he can get down to more specifics than I can.

9. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   No.

10. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    Trick has been used on several other systems and was an established executive component.

11. *Are those components the most easy to reuse?*

    Trick was relatively straightforward to reuse, since it had been used on 5-10 different projects of which I'm aware. Some models used in SES3 were also reused from other shuttle simulators, since they are developed at great expense and undergo significant validation.

12. *Are there components that are more difficult to reuse? If so, what are those components?*

Components that manipulate hardware are difficult to reuse, since each hardware interface is likely to be different from one simulator to the next. For example, I remember additional work having to be done to integrate the SES3 simulator with a new type of joystick. In general, any component that is highly coupled to many other components is also difficult to reuse, since all its dependencies would have to be reused as well.

13. *What do you consider to be the most significant risks to using a common component-based platform for PC simulations in both gaming and the military?*

    The most significant risk in any planned reuse effort is that the solution is developed in a lab without consultation with actual production projects. The lab solution is then forced upon production projects to justify the expense and chase after ROI. The production projects then fall behind as they struggle to learn the platform, fix aspects of it that are broken, and add new functionality where needed. A better solution is offered below, in which candidate projects are identified early in the platform's development cycle.

14. *What advice and recommendations do you have for someone creating a common component based architecture for gaming and for military simulation needs?*

    Understand the mantra that something built for reuse will typically follow the rule of 3's: It will take 3 times as long to develop, will cost 3 times as much, and must be used in 3 different systems before it can be dubbed reusable. Someone creating a common component based simulation architecture would

have to have these three initial system identified so that the generality of the architecture could be validated. These initial systems provide functional and nonfunctional requirements that may not have been obvious from initial analysis. Another aspect of this on which to focus are the underlying architectural patterns behind simulation. Reuse is happening more successfully at the design level than the component/implementation level in the software industry as evinced by the rise in adoption and identification of design patterns since the 90's. If you can identify simulation architecture patterns (as Fowler has done for enterprise architecture in Patterns of Enterprise Application Architecture), then you'll have made an enduring contribution that won't just sit on the shelf as another proof-of-concept prototype.

Interview with Dr. Clemens Szyperski, Microsoft Inc.

1. *How many years of experience have you had working with component-based software engineering (CBSE) and component based software architectures (CBSA)?*

   17 years.

2. *What types of CBSE and CBSA projects have you led or worked on? Have you worked on any gaming or military CBSE projects?*

   Prior to joining Microsoft I worked in the area in the research context on both real-time and non-real-time component-based systems. I am currently working on a project at the incubation level at Mircrosoft that involves a component-based platform – but the details of that project are still under

wraps. I haven't done anything in gaming or for the military. I have done some COM related projects and some .NET projects.

3. *What major issues or drawbacks have you encountered using CBSE?*

   CBSE really requires extensive up-front analysis, more so than other software engineering disciplines. The up-front analysis and early component-based engineering effort provides a set of cones to work between. If you start deviating from the path marked by the cones as the project progress, the things get very difficult. For that reason, CBSE is good for relatively mature fields – where the domain is clearly understood

4. *What do you consider to be the most significant risks to using CBSE?*

   One major risk of CBSE is that you can fall into the trap of doing a lot of abstract over-engineering that has little to do with real solutions to real problems – the effort in the end may not allow you to deliver anything concrete. The trick is to develop a set of end-to-end prototypes that mature as the engineering effort progresses.

5. *What major strengths and advantages have you encountered using CBSE?*

   If you get the design and execution right, the result is a much more solid engineering effort than typical. This is true because CBSE goes hand in hand with process and end-product maturity – the constraints imposed by CBSE cause engineers and developers to think much more carefully about what they are doing.

6. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

Yes, but I haven't worked on anything like that. I am familiar with at least one project of this type.

7. *How would you define a common software architecture for PC-based simulation?*

   Not asked as this question was poorly worded and confusing to the experts.

8. *Are you familiar with any common software architectures for PC-based simulation?*

   There was a project called CSRIO – an R&D project done by the Australian ministry of defense. It was a component-based architecture on the PC used for visualization.

9. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   Back when I was doing research at the university level I worked on a real-time component-based project relating to distributed time warp. Basically it was a time-bound simulation that used queuing policies on a set of registered components.

10. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    No, not really.

11. *Are those components the most easy to reuse?*

    Reuse is really an inappropriate term when talking about component-based architectures. Reuse implies adaptation to a new environment. Components

should be built for a certain purpose – if they are used for that purpose in two different environments then that is part of their "use", not reuse.

12. *Are there components that are more difficult to reuse? If so, what are those components?*

    When doing the upfront work for any component-based project, a good analysis of the domain is necessary. For a domain-specific platform, each component will have its place under the natural taxonomy of the domain. Components that fit this taxonomy will be easy to use. Components that try to break the boundaries will not.

13. *What do you consider to be the most significant risks to using a common component-based platform for PC simulations in both gaming and the military?*

    The biggest risk will be not doing a good job at domain analysis along with deep prototyping to ensure concrete solutions for both industries. There is also a risk from the project management side of not thinking over the complete timeline of the CBSE project.

14. *What advice and recommendations do you have for someone creating a common component based architecture for gaming and for military simulation needs?*

    Give some thought to building a component framework. Components by nature cannot be domain-independent. Also you will lose control of the deployed environment if you use self-driven components – those components

that dynamically try to interface with other components at run-time. Insist on defining component configurations before run-time, not during run-time.

Group 2: Experts in military PC-based virtual simulation

Interview with Curtis Conkey, NAVAIR

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the military?*

   I have worked in gaming and virtual simulation research for the military from 2002 to the present. But I worked for 20 years for Bell Labs before that in the field of PC-based communications network simulations.

2. *What types of PC-based simulation projects have you led or worked on?*

   We worked on a on-board team-based training simulation for the U.S.S. Virginia, a Navy submarine. Recently I've been working on research and experimentation projects with the open-source Delta3D game engine developed at the Naval Postgraduate School.

3. *What major issues or drawbacks have you encountered using PC-based technologies in the military?*

   There is a lack of science behind it right now. Because there hasn't been much scientific examination and experimentation in the field, we don't know if and when PC-based technologies will be useful for military applications. There are no guidelines found through a scientific process that tells anyone when the technology can be applied and which training objectives it can be used for. There are also no return-on-investment numbers out there to let the

military do any sort of trade-off analysis between PC gaming technologies and alternative solutions.

4. *What do you consider to be the most significant risks to using PC simulation to achieve military objectives?*

   Current commercial games and simulations on the market are not designed with the ISD process or objectives in mind – they are designed for entertainment only. The current thought in a lot of military circles is that if the game as a war theme, it can be used as a trainer. That's simply not true in many circumstances.

5. *What major strengths and advantages have you encountered using PC-based technologies in the military?*

   The immediate advantage that comes to mind is the military's ability to leverage the huge amount of research and development money and effort the entertainment industry as used to develop today's games. We have the opportunity to get it all for free. The other strength we get is the emphasis on the importance of the story – something that greatly aids immersion in the virtual world but is not used extensively in military training simulations.

6. *What advice and recommendations do you have for someone using PC simulation for military objectives (training, education, communication, analysis, etc)?*

   Definitely understand what you are attempting to train for before you try anything. It is important to have the training objectives laid out before any work is done.

7. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

   Yes.

8. *How would you define a common software architecture for PC-based simulation?*

   It is one that is used to create both entertainment and training material.

9. *Are you familiar with any common software architectures for PC-based simulation?*

   Sure. The Unreal engine, Delta3D and Gamebryo all come to mind. They are engines that have all been used to provide entertainment and have been used in training as well. In fact the guys writing Delta3D are looking at making it a modular architecture – allowing you to swap its physics engine with another one. They have used it recently as a firefighter trainer.

10. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

    No, not personally, but America's Army is a good example of this. Many of the tools used to create content for entertainment and being converted to create training content for the military. It's being used as a platform to do both.

11. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

No, but there are fundamental difference between similar products used for gaming and training. For example, the difficulty level in military training scenarios will be higher - more realistic. It wouldn't be as fun to play at this level for entertainment.

12. *Are those components the most easy to reuse?*

    Not applicable.

13. *Are there components that are more difficult to reuse? If so, what are those components?*

    Not applicable.

14. *What do you consider to be the most significant risks to using a common software architecture for PC simulation to achieve military objectives?*

    There are different underlying motivations when you are doing entertainment versus when you are doing training. The objectives are different. Also, there will probably be fidelity issues – what you can get by with for gaming may not be satisfactory for training.

15. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    The gaming world is coming out with increasingly realistic graphics effects – sometimes even building scenarios around them to show them off. They add a lot to the realism of the scene and the military needs to look at incorporating these technologies. From the hardware side, gaming is increasingly going to multi-core parallel processors. Also we need to look at ways to tie what is

happening in scenarios back to a learning management system for tracking – aiding in training and remediation.

Interview with Peter Smith, NAVAIR

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the military?*

   2 years.

2. *What types of PC-based simulation projects have you led or worked on?*

   Most recently I've worked on the Delta3D project. Before that I worked for SAIC doing database development for CCTT and other PC sim projects.

3. *What major issues or drawbacks have you encountered using PC-based technologies in the military?*

   Interestingly there's no real resistance to using PC technologies in the military. It used to be the older generations were opposed to it but that's not true anymore. One of the assumptions we've made in the past is that the younger generations will adapt to game-based training easily – which overall seems to be true for the U.S. But in the U.K. we've found that many recruits are not gamers and don't adapt as easily.

4. *What do you consider to be the most significant risks to using PC simulation to achieve military objectives?*

   Sometimes in training fidelity really does matter. If you tried to do certain tasks in PC-based sims without the required fidelity, you would produce negative training.

5. *What major strengths and advantages have you encountered using PC-based technologies in the military?*

   PC-based gaming, especially recently, has really encouraged emotional involvement. If we apply this to military training, we can make trainers that are compelling for recruits to use – they would sit and want to train themselves – especially if there is a competition element involved.

6. *What advice and recommendations do you have for someone using PC simulation for military objectives (training, education, communication, analysis, etc)?*

   Just because it's a simulation doesn't mean it's a game. It's important to have compelling content and scenarios – to bring the aspect of game play into simulations.

7. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

   Sure, the Torque engine is an example of one.

8. *How would you define a common software architecture for PC-based simulation?*

   Not asked as this question was poorly worded and confusing to the experts.

9. *Are you familiar with any common software architectures for PC-based simulation?*

   Recently from the handheld personal computing side, there is the GP2X platform, the Zodiak, and mobile learning platforms.

10. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

    No, but I've used both Full Spectrum Warrior and Full Spectrum Commander. These are platforms that have been purposely built to provide both entertainment and training.

11. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    Referring back to Full Spectrum Warrior – there is actually very little difference between the training side and the entertainment side. But there are some important differences – things that are more realistic in the training version. Enemies won't give themselves away like they do in the game – you won't get a second chance to turn a corner when there is an enemy on the other side.

12. *Are those components the most easy to reuse?*

    Not applicable.

13. *Are there components that are more difficult to reuse? If so, what are those components?*

    Not applicable.

14. *What do you consider to be the most significant risks to using a common software architecture for PC simulation to achieve military objectives?*

    You will want to make sure that you don't pass around components to people you don't want to have them – there would be a security risk of people reusing

components who are not authorized to reuse them. Also the entertainment industry and the military have very different concerns and objectives when it comes to how their products are built and deployed.

15. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    Shader languages and graphics acceleration hardware is the most immediate thing coming down the pipe from the gaming world. A lot of advances are being made in that area that the military could benefit from.

Interview with Dr. Roger Smith, Sparta Inc.

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the military?*

   20.

2. *What types of PC-based simulation projects have you led or worked on?*

   I haven't worked on any gaming projects.

3. *What major issues or drawbacks have you encountered using PC-based technologies in the military?*

   I think the industry is still figuring that out. With the serious games out now there's no real originality. Basically they all are training soldiers how to use equipment – this was started with Marine Doom and hasn't changed since. Also, the military creates requirements that define what they want in a training solution – they will accept games now, but unless it's written as a requirement they don't want it.

4. *What do you consider to be the most significant risks to using PC simulation to achieve military objectives?*

   More originality is needed. We need more serious games that concentrate on team training or training to interact with other cultures – a big need area right now.

5. *What major strengths and advantages have you encountered using PC-based technologies in the military?*

   It can provide small low-budget training aids and devices that wouldn't be there otherwise. For example I saw a small training simulation where they used the America's Army platform to train mobile robot operators to destroy IEDs. This wouldn't have been developed on a typical large-scale military contract, but PC-based gaming technologies allowed it to be achieved at a low cost.

6. *What advice and recommendations do you have for someone using PC simulation for military objectives (training, education, communication, analysis, etc)?*

   The military is convinced to use something a little at a time. Marine Doom came about not through the acquisition environment but because a couple of Marines who knew how to use the tools created something from a game to train themselves. It's risky to try to be the first one to do something for military training – the military tends to use things that have proven themselves out already. However, people with gaming experience are now moving into

positions of responsibility in the military so gaming solutions continue to gain more and more widespread acceptance as viable training solutions.

7. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

   Refused to answer.

8. *How would you define a common software architecture for PC-based simulation?*

   Refused to answer.

9. *Are you familiar with any common software architectures for PC-based simulation?*

   Refused to answer.

10. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

    Refused to answer.

11. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    Refused to answer.

12. *Are those components the most easy to reuse?*

    Refused to answer.

13. *Are there components that are more difficult to reuse? If so, what are those components?*

    Refused to answer.

14. *What do you consider to be the most significant risks to using a common software architecture for PC simulation to achieve military objectives?*

    From the gaming side, there is no motivation for interoperability. Companies that make games don't want to have there games talk to each other. Games are meant to be entertaining, while training is meant to transfer knowledge. A serious risk you may face when building a generic platform like this is that you start making assumptions early on that might get built into the software – assumptions relating to scale, size, fidelity, etc.

15. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    Distributed multi-player games are played on the open internet and are accessible from just about anywhere now thanks to wireless technologies. There are no limits on connectivity. While there are security issues still to be addressed this type of connectivity will be useful for military training. There are also some really nice scenario development tools on the market for games right now that could be used to create training scenarios.

Interview with Dr. Michael Zyda, ISI at USC

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the military?*

   20 years – from 1986.

2. *What types of PC-based simulation projects have you led or worked on?*

   I was director of the America's Army project. I also worked on the NPSNET visual simulation system at the Naval Postgraduate School

3. *What major issues or drawbacks have you encountered using PC-based technologies in the military?*

   PCs cannot at this time provide good solutions for simulations with multiple visual streams. While the military has been interested for a long time to use gaming technology, there is not a huge commercial market to support PC-based military simulations.

4. *What do you consider to be the most significant risks to using PC simulation to achieve military objectives?*

   For any simulation running on Microsoft's Windows operating system, there is no guarantee of security. Simulations are susceptible to local and network-based viruses and other security flaws that continue to be found in the operating system.

5. *What major strengths and advantages have you encountered using PC-based technologies in the military?*

   PC-based technologies can be very portable. Solutions can be fielded on laptops and used in any military theater.

6. *What advice and recommendations do you have for someone using PC simulation for military objectives (training, education, communication, analysis, etc)?*

   Watch out for engineers with a lot of experience in visual simulations systems that call themselves game developers. There is a lot of misrepresentation on the military side, and the reality is that not many defense contractors or military branches have much experience developing gaming technologies. At

this point the government doesn't want to pay the real cost or invest time in developing real gaming solutions for the military.

7. *Are you familiar with the concept of a common software architecture for PC-based simulation?*

   Yes. HLA was originally intended for both military and commercial entertainment use. But it was too heavy weight for gaming companies to incorporate. It also required the use of black box systems and software and it had relatively poor performance. New objects were statically, not dynamically defined, so incorporating them required access to the source code and a rebuild from all parties.

8. *How would you define a common software architecture for PC-based simulation?*

   Not asked as this question was poorly worded and confusing to the experts.

9. *Are you familiar with any common software architectures for PC-based simulation?*

   Darwars was developing something along those lines. Unfortunately it only amounted to some good demos – nothing substantial came out of the project. I also crafted the original operating plan for ICT who are now working on a project called the Integrating architecture.

10. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

    No. But the Unreal 3 engine has been used on both sides to develop solutions for both gaming and training. Also, back at the Naval Postgraduate School, a

Ph.D. student named Ken Watsen was working on something of this sort, but he never finished.

11. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

    No, but CBSA is important. We need simulations that are dynamically extensible and syntactically interoperable.

12. *Are those components the most easy to reuse?*

    Not applicable.

13. *Are there components that are more difficult to reuse? If so, what are those components?*

    Not applicable.

14. *What do you consider to be the most significant risks to using a common software architecture for PC simulation to achieve military objectives?*

    If you are planning to use CBSA then verifying security is an important issue. Because CBSA relies on black box components there needs to be a way to ensure that security of the resulting simulations will not be compromised. It could potentially be a breeding ground for malicious viruses. You need to use trusted certificates or some alternate technology to ensure only safe code is running.

15. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

Gaming technology is starting to use sensors of human emotional state. This could be used to huge advantage in military training. Specifically you could measure training affect. Right now the gaming industry is spending much more R&D money than the military is in these areas. They will continue to drive the industry.

Group 3: Experts in Virtual Simulation for Commercial Entertainment and Gaming

Interview with Tom Carbone, FIEA

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the gaming industry?*

   15 years in the gaming industry.

2. *What types of PC-based simulation projects have you led or worked on? What type of development environment? What type of software architecture?*

   I worked on Madden 05 and 06, Space Jam, and the NHL series at EA Tiburon. They were actually all console games so we developed on the console platform tools that were provided to us from the manufacturer.

3. *What are the largest challenges facing game developers and game development companies today?*

   It's getting harder for them to find qualified people to build next generation gaming content. Team sizes are huge, budgets are huge, and managing all that has become difficult. From a technical standpoint it is difficult to leverage middle ware that's on the market without causing disruptions in the game development cycle. It's also challenging for them to make games that stand

out from the rest of the market and continue to reuse as much as possible from other product lines.

4. *Have you used any component technologies in PC game development projects? If so, explain.*

   Havok is a middleware physics engine that they were using on Madden. But it was pretty hard to use because you didn't know what was going on behind the scenes. It was hard to optimize what your code because you didn't have full control of the environment.

5. *Are you familiar with any common software architectures for PC-based simulation?*

   America's Army immediately comes to mind.

6. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   No.

7. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

   No.

8. *Are those components the most easy to reuse?*

   Not applicable.

9. *Are there components that are more difficult to reuse? If so, what are those components?*

   Not applicable.

10. *What experience have you had using military technologies for gaming? What are the most significant business and technical risks involved in doing this?*

    I haven't had any experience developing games with military technologies. We just started working with Lockheed to develop a PC game for the F-35 and we're using some code from their cockpit simulation. One of the issues with using military technologies in the gaming world is confidentiality – obviously military secrets need to be protected but games get distributed all over the world and can be reverse engineered. On the other hand gamers that buy military games want to know that they what they are using is exactly like in the real military. Bridging that gap is difficult – you don't want to lie to the consumers and say the game is realistic when it's not, but you don't want to give away military secrets either.

11. *What types of military technologies would you like to see in a PC simulation game?*

    It would be good to see some more realistic flight sims out there. I'd also like to see commercial war games be based on real battles that actually happened with real data collected from the battle. Companies like Lockheed shouldn't try to compete head to head with established gaming companies like EA, but should try to offer something that EA can't offer.

12. *What do you see as the greatest business and technical risks for using PC simulation and gaming technology in the military?*

    Games are not thoroughly tested or documented and they are not meant to be maintained because their shelf life is so short. The military on the other hand

has a huge approval process all products must go through before they are used. While gamers can live with bugs here and there, the military can't afford to put gaming software straight into use.

13. *What advice and recommendations do you have for someone creating a single framework for PC gaming and military simulation?*

    Make sure to find people that know the games industry well – that can help develop the philosophies and ideas required to make it successful on the commercial gaming side. The technical challenges really aren't the most important in a project like this. You're also going to have to deal with the limitations of data and content provided by the military. The gaming industry relies a lot on customization – everything is tailored and optimized to achieve their objective – entertainment. By nature this goes against principles of reusability.

Interview with Stephen Eckman, Disti Inc.

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the gaming industry?*

   I worked 3 years in the games industry for EA and about 1 year hear at Disti.

2. *What types of PC-based simulation projects have you led or worked on? What type of development environment? What type of software architecture?*

   I worked on Madden 03 and Madden 04. I also worked with my brother on a massively-multiplayer online game. We wrote it from scratch using Cold Fusion as a platform.

3. *What are the largest challenges facing game developers and game development companies today?*

   For game developers it's become increasingly hard to create independent games – the budgets are so large these days. Interestingly, because games are so expensive it's also unlikely that an established game studio will try something brand new because it's so risky. You don't want to spend millions of dollars trying something new when you already have a product line that you know will sell.

4. *Have you used any component technologies in PC game development projects? If so, explain.*

   We built our own network library that we reused a lot in our online game, but it was a compile-time reuse. We didn't really have it set up to use components.

5. *Are you familiar with any common software architectures for PC-based simulation?*

   There's a company called Pandemic that released Full Spectrum Command and Full Spectrum Warrior – those were used for both gaming and military use. Also America's Army did it. But as far as I know, there's no single development platform out there that was built to create content for both.

6. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   No, I haven't.

7. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

   We do that with our GLStudio product – our components are called RSOs – reusable simulation objects. I've worked on a couple of military contracts here where we reused RSOs from previous projects. We did an F-15 and a T-45 cockpit that reused some instruments that we had previously built for other projects.

8. *Are those components the most easy to reuse?*

   In my experience with those projects just mentioned it was easiest to reuse the components when the new project required the exact same pieces as the old project. The ADI and the VSI in the T-45 were exactly the same instruments that we had already built for another aircraft which meant they required the exact same interface so we just dropped them in and they worked.

9. *Are there components that are more difficult to reuse? If so, what are those components?*

   Components that are tied too closely to a particular simulation or domain are hard to reuse. We have a menu scripting component that was built for one simulation but it was built in a way that tied it to the platform in which it was used – so reusing it was difficult.

10. *What experience have you had using military technologies for gaming? What are the most significant business and technical risks involved in doing this?*

I haven't had much experience developing things for the military. I do know that a lot of code built for the military is not built efficiently – it's not optimized like most gaming content is. There's also a different level of fidelity involved.

11. *What types of military technologies would you like to see in a PC simulation game?*

    I'd like to see the scale of the large distributed military simulation come over to the gaming side – games with like 3000+ entities in them that can all be controlled by different people. In Battlefield 2 you can play with 64 different players and that's a big deal to the gaming industry – we're not even close to the military in that aspect of distributed virtual simulation. Also the AI in games isn't really human – the fidelity just isn't there like the require for military simulations.

12. *What do you see as the greatest business and technical risks for using PC simulation and gaming technology in the military?*

    The gaming industry doesn't have nearly the fidelity required for military simulations. They do a minimal amount of testing and a minimal amount of QA on their products. It's still acceptable for a game to be released with bugs as long as the gamers know that patches will be released eventually – this is not ok for military simulations.

13. *What advice and recommendations do you have for someone creating a single framework for PC gaming and military simulation?*

Come up with a standard definition for different objects in the simulation – similar to what they did with HLA. This would probably be a lot harder to implement than the traditional approach – and it might still be easier for someone to create the same object from scratch and not conform to your definition.

14. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    I think a good look should be taken at the gaming hardware interfaces that are out there. We have a whole generation of kids out there that know how to interact effectively with game controllers from the PS2 and Xbox – the military should look at taking advantage of that knowledge.

Interview with Dr. Michael Gourlay, FIEA

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the gaming industry?*

   I've spent 4 years in the gaming industry working for EA and now teach gaming classes at FIEA.

2. *What types of PC-based simulation projects have you led or worked on? What type of development environment? What type of software architecture?*

   In my Ph.D. program I worked on computational fluid dynamics – basically it was a simulation and visualization application of physics algorithms. I was also the network architect for EA's Nascar and worked on AI and physics as well. On EA's Madden I was the lead for the graphics engine team.

3. *What are the largest challenges facing game developers and game development companies today?*

   It's hard to find experienced programmers that can write console games. Most of the programmers that come out of school are inexperienced and have done no project work in embedded systems or consoles. The academic world is leaning towards teaching programming in Java, not C – the language used for most games. And the students have no experience optimizing code for performance or a consistent frame rate.

4. *Have you used any component technologies in PC game development projects? If so, explain.*

   EA doesn't use any component technologies because it's too much of a risk. Technology is changing so fast and console hardware has about a 5 year shelf life before it's replaced, so it's not practical to spend the effort required to develop reusable components when they also have a small shelf life. The gaming industry is not yet mature enough to use component technologies. Also, components don't work well across platforms, especially in gaming where each piece of software needs to be optimized for the particular platform it is running on. EA however has a standard set of libraries that they use in many of their games. In my research I used IRIS Explorer which is a component based visualization tool and framework. VTK – Visualization Toolkit, another framework for making visualization applications is component-based.

5. *Are you familiar with any common software architectures for PC-based simulation?*

   Criterion used to make Renderware, now owned by EA. That was a widely used middleware package for games that could be used by the military as well. There is also the Havok physics engine for games that could be used for both.

6. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   No, I haven't developed anything that could be used for the military.

7. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

   The Madden network group reused the network architecture that we developed for Nascar. The visual software and graphics engine we built for Madden was intended to be reused in other games. It got reused by some games and is still in used, but not everyone knew enough about it to use it.

8. *Are those components the most easy to reuse?*

   I think that generic context free components would be the most easy to reuse – those things dealing with math or file I/O. It's possible you could make a reusable physics engine component but things like this have strong dependencies on the platform they are developed for so reuse is limited.

9. *Are there components that are more difficult to reuse? If so, what are those components?*

I don't think something like a rendering engine could be used as a component. It is tied closely to the hardware and platform so you couldn't use it across platforms. It's also domain specific – a rendering engine built for rendering cars may not be suitable for rendering football players. However a component like this could be used across a particular product line.

10. *What experience have you had using military technologies for gaming? What are the most significant business and technical risks involved in doing this?*

    Gaming has gotten some good ideas and technology from the military's DIS networking protocol. However things like this that are built for the military deal with a lot of complexity, and games try to avoid complexity as much as possible.

11. *What types of military technologies would you like to see in a PC simulation game?*

    Since I don't have experience with military technologies I'm not sure what's out there. From the hardware side it would be nice to see some full motion high fidelity simulators at someplace like Dave and Buster's or in a high end arcade. However, a direct use of a military simulation would always be a problem for gaming – it's not entertaining.

12. *What do you see as the greatest business and technical risks for using PC simulation and gaming technology in the military?*

    From the military's standpoint there aren't too many risks other then possibly quality and robustness. But if they are only using the technology for training and not mission critical applications, even quality and robustness might not

matter much. Because gaming technology is cheap the military can always write it off if the project goes wrong. From the perspective of a gaming company that wants to start using gaming technology on military contracts there is a big risk of failure and the loss of a lot of money – especially if it's a small company. Gaming companies aren't used to collecting specific requirements that must be met – they typically don't go through that kind of a structured process and aren't used to working that way.

13. *What advice and recommendations do you have for someone creating a single framework for PC gaming and military simulation?*

    Make sure that you know how to do architecture. Software architecture is only an issue for very large projects with large teams and lots of code. I don't think the gaming industry is mature enough to create and use solid software architectures.

14. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    One of the things that's just starting to emerge for PC gaming is a dedicated physics card. Like a graphics card, the physics card uses hardware and a dedicated processing unit to calculate physics models and equations in a game or simulation. This could be used to add a lot of realism to scenarios.

Interview with Keelan Stuart, Disti Inc.

1. *How many years of experience have you had working with PC technologies and PC-based simulation in the gaming industry?*

    About 10 years.

2. *What types of PC-based simulation projects have you led or worked on? What type of development environment? What type of software architecture?*

   I worked on Nuclear Strike and Madden 99 for EA, a game called Revenant for Cinematix. Worked for a couple years on a game called Legend of 5 Suns for Whispering Tree Studios – it was originally a PC-based RTS game but morphed into an Xbox RPG game – but in the end it was never released. I also worked on a MMP game for Artifact called Horizons.

3. *What are the largest challenges facing game developers and game development companies today?*

   This really depends on the size and budget of the game studio. Overall though, they are always trying to determine what makes a hit, what types of games will sell and what types will not. They are also struggling to make sure that they meet gamers' expectations of what the games will look and play like. There has also been a divergence of technology from things like hardware, shader languages, and the graphics technologies based around OpenGL and DirectX. Competing tools, languages, and products are starting to look more and more different which by nature make games less compatible across different systems.

4. *Have you used any component technologies in PC game development projects? If so, explain.*

   They started to do something like that at Cinematix when I worked for them but I'm not sure how far that project got. I've built and currently maintain my own game engine which is somewhat component-based. I created a plug-in

system where each plug-in uses a predefined set of macros to communicate with the game engine. The game engine can then auto-register the plug-ins and use them directly at run-time – they don't need to be compiled in.

5. *Are you familiar with any common software architectures for PC-based simulation?*

   I believe America's Army does something similar – it's used for both gaming and for military uses. They based that project on the Unreal engine. Recently at I/ITSEC I also saw a naval combat project that was used both as a game for entertainment and for military training – but I don't remember the name of it.

6. *Have you attempted to develop PC-based simulation that might be considered a basis for a common software architecture?*

   No, I haven't done much military work at all.

7. *Have you built PC-based simulations that used major components from previous simulations that you have constructed? If so, what were those components?*

   Sure, I built plug-ins for my engine for particle systems and for weather. Theoretically you could use the same mechanism to create plug-ins for characters and game structures.

8. *Are those components the most easy to reuse?*

   I've found that components that are context-free and non-domain specific to be the most easy to reuse. Things like I mentioned before – particle systems, weather, and potentially terrain would all be things that would have a standard set of interfaces that could be reused easily.

9. *Are there components that are more difficult to reuse? If so, what are those components?*

   Things that are specific to a particular game or objective would be harder to reuse. For example a boat object I built for a boat-racing game only got used for that one game – it wasn't something that got used a lot.

10. *What experience have you had using military technologies for gaming? What are the most significant business and technical risks involved in doing this?*

    I haven't really had much experience with military technologies at all.

11. *What types of military technologies would you like to see in a PC simulation game?*

    It would be nice to see some really large scale games like the large scale military simulations. They could use real world satellite data and have connectivity like they do with HLA. Unfortunately I don't think this kind of connectivity, especially between games, is going to happen any time soon. Each game is so customized, and unlike the military, there is a lot of fierce competition and concerns about intellectual property protection.

12. *What do you see as the greatest business and technical risks for using PC simulation and gaming technology in the military?*

    I think the majority of the risk is business risk and not technical risk. Compared to the gaming world there is a lot of resistance to change and new technology in the military. I imagine it would be difficult to convince the military of the benefits of all the new technologies coming out in the short amount of time involved.

13. *What advice and recommendations do you have for someone creating a single framework for PC gaming and military simulation?*

    Definitely make it modular – plug-ins for everything. Also, make it data driven – meaning load the data dynamically at run-time – don't use static resources. Make some of it script-based too – that way there's less need for recompiling.

14. *What are the types of things that you see in the PC gaming / entertainment world that you believe would be useful for the military?*

    I think the military could learn a lot from what the gaming community is doing with user interfaces. I was at I/ITSEC this year and I thought everything was so utilitarian and much of what I was seeing was based on technology that was 10 years old.

APPENDIX B: PHASE I RESULTS – FACTORS

Table 18: Factor - Leveraging middleware

1	Name:	Leveraging middleware
	Type:	Organizational
	Description:	Game companies are finding it difficult to leverage middleware without causing disruptions in the development lifecycle.
	Reference:	Carbone interview, q. 3.
	Flexibility:	As time progresses, as the industry matures, and as more middleware tools become available, this factor will become more and more important.
	Impact:	Impacts interfaces to 3rd party products and processes, organization's own development and integration processes.

Table 19: Factor - Competitive advantage

2	Name:	Competitive advantage
	Type:	Product
	Description:	Game companies must continue to maintain their competitive advantage. They must customize their content to be able to stand out in the market, and by nature this goes against the principles of reusability.
	Reference:	Carbone interview, q. 3. Carbone interview, q. 13. Stuart interview, q. 11.
	Flexibility:	In a competitive marketplace, maintaining competitive advantage will always be a big factor
	Impact:	Large impact on types of games produced and game company's willingness to use tools and components everyone else is using.

Table 20: Factor - Product line reuse

3	Name:	Product line reuse
	Type:	Organizational
	Description:	To operate efficiently, game companies need to continue to reuse as much as possible within and across their own product lines
	Reference:	Carbone interview, q. 3.
	Flexibility:	There will always be cost savings associated with reusing at least some previously developed in house content. However it will be offset by the ease with which technology allows new products can be created from scratch.

	Impact:	Moderate impact on products and development processes, but presents a case for a single architecture's ability to sustain product lines.

Table 21: Factor - Black box component reuse

4	Name:	Black box component use
	Type:	Organizational
	Description:	There will be some resistance to black box use because developers won't know what's going on behind the scenes, won't necessarily know how to optimize their code, and may not be able to debug into the black box component.
	Reference:	Carbone interview, q. 4. Norton interview, q. 4.
	Flexibility:	Black box suspicion will always be a factor. Ability of 3rd party black box creator to instill trust will alleviate this factor.
	Impact:	Moderate impact on development processes.

Table 22: Factor - Confidentiality of military technology in games

5	Name:	Confidentiality of military technology in games
	Type:	Product
	Description:	Confidentiality of military technology in games needs to be maintained but commercial games always have the potential to be reverse engineered.
	Reference:	Carbone interview, q. 10.
	Flexibility:	Unlikely to change. Technology may prevent reverse engineering in the future.
	Impact:	Any military component used and distributed in a commercial context will be affected strongly by this factor.

Table 23: Factor - Differing gaming and military content shelf lf life

6	Name:	Differing gaming and military content shelf life
	Type:	Product
	Description:	Sharing content is difficult because gaming content shelf life is very short and military content shelf life is very long.
	Reference:	Carbone interview, q. 12. Gourlay interview, q. 4

	Flexibility:	Game shelf life may increase as technology advance rate slows. Military content shelf life will probably not change.
	Impact:	Components with the highest reuse rate are forced to stabilize with increased shelf life.

Table 24: Factor - Differing gaming and military content quality

7	Name:	Differing gaming and military content quality
	Type:	Product
	Description:	Sharing content is difficult because gaming content has not been through extensive testing or quality assurance like military content has.
	Reference:	Carbone interview, q. 12. Eckman interview, q. 12. Humphrey interview, q. 11. Gourlay interview, q. 12.
	Flexibility:	While game shelf life is low, game content quality will remain low. If game shelf life increases, quality of content will increase.
	Impact:	Large impact on the ability of the military to use content developed for games.

Table 25: Factor - Lack of science behind military gaming technology

8	Name:	Lack of science behind military gaming technology
	Type:	Technological
	Description:	There is a lack of science behind the military's use of gaming technology. There are no guidelines based on a scientific process directing when and how to use the technology. There are also no return on investment numbers to determine if the technology is worth using.
	Reference:	Conkey interview, q. 3.
	Flexibility:	This is already changing
	Impact:	Will determine if and when gaming components can be used for military purposes.

Table 26: Factor - Differing gaming and military content objectives

9	Name:	Differing gaming and military content objectives
	Type:	Organizational

	Description:	Games are designed for entertainment while the military uses games for other objectives like analysis or training. Different objectives implies different underlying assumptions, designs, and end products.
	Reference:	Conkey interview, q. 4. Conkey interview, q. 14. Garfunkel interview, q. 13. Humphrey interview, q. 11. Smith, Peter interview, q. 14. Smith, Roger interview, q. 14.
	Flexibility:	Probably fundamentally unchangeable
	Impact:	Moderate impact on content type, large impact on architecture and design direction.

Table 27: Factor - Training objectives drive technology choices

10	Name:	Training objectives drive technology choices
	Type:	Technological
	Description:	In the context of military training, the training objectives should drive what technology is used and how it is implemented.
	Reference:	Conkey interview, q. 6.
	Flexibility:	May or may not be a factor depending on what the results of the ISD process determine about the technology.
	Impact:	Small impact on architecture and design, large impact on content type.

Table 28: Factor - Differing gaming and military content fidelity

11	Name:	Differing gaming and military content fidelity
	Type:	Product
	Description:	Difficulty level and fidelity required by military training usually would not result in entertaining game play.
	Reference:	Conkey interview, q. 11. Conkey interview, q. 14. Eckman interview, q. 10. Eckman interview, q. 12. Garfunkel interview, q. 13. Humphrey interview, q. 9. Humphrey interview, q. 11. Smith, Peter interview, q. 4. Smith, Peter interview, q. 11.

	Flexibility:	Games continue to increase in fidelity while maintaining entertaining game play. Many games now have the ability to scale the level of fidelity during game play. May not be a problem in the future.
	Impact:	Minor impact – components can be built to support different levels of fidelity.

Table 29: Factor - Increasingly realistic game graphics

12	Name:	Increasingly realistic gaming graphics
	Type:	Technological
	Description:	The military should take advantage of the increasingly realistic graphics technologies, shader effects, and video hardware available in the gaming world.
	Reference:	Conkey interview, q. 15. Smith, Peter, interview q. 15.
	Flexibility:	Will continue to be a factor until technological advances slow.
	Impact:	Any components related to the graphics pipeline will be affected and will need to continue to adapt with the technology or risk becoming outdated.

Table 30: Factor - Tie-in to learning management system

13	Name:	Tie-in to learning management system
	Type:	Product
	Description:	What takes place in military training scenarios must be tied back in to a learning management system to drive follow-up analysis and remedial training.
	Reference:	Conkey interview, q. 15.
	Flexibility:	Will become increasingly important as LMSs become widespread in military training.
	Impact:	Large impact on all components and much of the architecture. The architecture must support the ability to track and record scenario play for later playback and analysis.

Table 31: Factor - Increasing game budgets and team sizes

14	Name:	Increasing game budgets and team sizes
	Type:	Organizational

	Description:	Because of increasingly large budgets and team sizes required to create a game that will sell, it is hard to create independent games. It is also unlikely that established gaming studios will risk trying something new and different, so game content becomes stale.
	Reference:	Eckman interview, q. 3.
	Flexibility:	Unchangeable for the foreseeable future.
	Impact:	Minor impact on architecture and design. Major impact on the pressure for reuse because huge teams and budgets means much content can be created from scratch.

Table 32: Factor - Component reuse difficulties: different purpose and different interface

15	Name:	Component reuse difficulties: different purpose and different interface
	Type:	Product
	Description:	Component reuse is easiest when the component is reused for the exact same purpose and requires the exact same interface as the original use. Components that interface to specific hardware devices are especially difficult to reuse.
	Reference:	Eckman interview, q. 8. Humphrey interview, q. 9. Norton interview, q. 12. Stuart interview, q. 8.
	Flexibility:	Not likely to change especially in regards to re-purposing. Tools may be developed to automate interfacing.
	Impact:	Minor impact on component design, major impact on component interface.

Table 33: Factor - Component reuse difficulties: close ties to domain and context

16	Name:	Component reuse difficulties: close ties to domain and context
	Type:	Product
	Description:	Component tied too closely to a particular simulation or domain are hard to reuse.

	Reference:	Eckman interview, q. 9. Humphrey interview, q. 9. Humphrey interview, q. 10. Norton interview, q. 3. Stuart interview, q. 8. Stuart interview, q. 9. Gourlay interview, q. 9.
	Flexibility:	Will change as developers become more educated how to design components and the simulation domain becomes more rigidly partitioned.
	Impact:	Major impact to component design and simulation domain partitioning or componentizing.

Table 34: Factor - Differing gaming and military content optimization

17	Name:	Differing gaming and military content optimization
	Type:	Product
	Description:	Code built for the military is not built as efficiently as gaming content is – it is not optimized for the deployment environment like gaming content is.
	Reference:	Eckman interview, q. 10. Humphrey interview, q. 11.
	Flexibility:	Unlikely to change
	Impact:	Major impact on the gaming industry's willingness to use military components.

Table 35: Factor - Backwards compatibility and version upgrades

18	Name:	Backwards compatibility and version upgrades
	Type:	Product
	Description:	In a component framework, interfaces have to be sustained through multiple versions of your product and 3rd party products so that clients of the framework can continue to use components without having to change them.
	Reference:	Garfunkel interview, q. 3.
	Flexibility:	Unlikely to change unless technology advances slow.
	Impact:	Major impact on all component designs and interfaces and the design of the architecture.

Table 36: Factor - Component engineering effort

19	Name:	Component engineering effort
	Type:	Technological

	Description:	It's almost always easier to build something in one monolithic block than split up into components and spend all that effort on interfacing. CBSE really requires extensive up-front analysis, more so than other software engineering disciplines.
	Reference:	Garfunkel interview, q. 3. Humphrey interview, q. 4. Szyperski interview, q. 3.
	Flexibility:	Unchangeable – component based software engineering requires extensive analysis of the domain.
	Impact:	Large impact on the ability of the architecture to meet the needs and demands of the virtual simulation domain.

Table 37: Factor - Component performance

20	Name:	Component performance
	Type:	Product
	Description:	Loading potentially hundreds of components at run-time is not ideal, but it's something that is required for large complex simulations. There is also no good way to guarantee performance within a component.
	Reference:	Garfunkel interview, q. 3. Norton interview, q. 4.
	Flexibility:	Will become less of a factor as technology advances and as component performance measures become standardized.
	Impact:	Moderate impact on initialization and component loading processes. May impact component manufacturers to create quality of service and performance contracts.

Table 38: Factor - Component framework complexity

21	Name:	Component framework complexity
	Type:	Product
	Description:	Component based software engineering by nature can add a lot of complication to a simulation. This is a risk especially when dealing with new developers and new customers.
	Reference:	Garfunkel interview, q. 4. Garfunkel interview, q. 14.

	Flexibility:	From the standpoint of the framework it is unchangeable. From the standpoint of new developers and customers, it can be mitigated if their interaction with the component framework is made as simple and straightforward as possible.
	Impact:	Major impact on the component simulation framework. Moderate impact on the environment presented to developers.

Table 39: Factor - Component reuse difficulties: many dependencies

22	Name:	Component reuse difficulties: many dependencies
	Type:	Product
	Description:	Components that have the least number of dependencies are the easiest to reuse while components that can affect lots of other components are less easy to reuse.
	Reference:	Garfunkel interview, q. 11. Garfunkel interview, q. 12. Norton interview, q. 12. Gourlay interview, q. 8.
	Flexibility:	May be changeable by reducing component dependencies or partitioning components with many dependencies.
	Impact:	Moderate impact on components with many dependencies and content that depends on those components.

Table 40: Factor - Legacy code integration

23	Name:	Legacy code integration
	Type:	Organizational
	Description:	An organization using the component-based framework may want to incorporate legacy non-componentized systems.
	Reference:	Humphrey interview, q. 3. Humphrey interview, q. 11.
	Flexibility:	May be changeable if it is easy to recreate functionality of the legacy systems.
	Impact:	Moderate impact on the framework architecture and design if it is to support non-componentized systems.

Table 41: Factor - Domain model componentization

24	Name:	Domain model componentization

	Type:	Product
	Description:	The architecture must be explicitly designed around the domain, but where you partition your data and domain model into components is a big risk. When you partition into components you make assumptions that may not be true in the future.
	Reference:	Humphrey interview, q. 4. Szyperski interview, q. 12.
	Flexibility:	Unchangeable – component based software engineering requires extensive analysis of the domain.
	Impact:	Large impact on the ability of the architecture to meet the needs and demands of the virtual simulation domain.

Table 42: Factor - Development in a vacuum or lab environment

25	Name:	Development in a vacuum or lab environment
	Type:	Product
	Description:	The architecture should not be developed in a vacuum. Use or adapt architectures that are already in use and test the platform out on production projects, not just in the lab. Otherwise the end result may have to be significantly altered to be of use.
	Reference:	Humphrey interview, q. 12. Norton, q. 13.
	Flexibility:	Unlikely to change – the framework should be tested on real world problems.
	Impact:	Moderate impact on framework and design decisions.

Table 43: Factor - Gaming interoperability

26	Name:	Gaming interoperability
	Type:	Organizational
	Description:	Gaming companies have no motivation for interoperability – they don't want to have their games talk to each other.
	Reference:	Smith, Roger interview, q. 14.
	Flexibility:	May change if consumers demand it or if market forces promote interoperability.
	Impact:	Moderate impact on components dealing with interoperability issues.

Table 44: Factor - Built-in assumptions of a generic platform

27	Name:	Built-in assumptions of a generic platform
	Type:	Product
	Description:	A serious risk you may face when building a generic platform like this is that you start making assumptions early on that might get built into the software – assumptions relating to scale, size, fidelity, etc.
	Reference:	Smith, Roger interview, q. 14.
	Flexibility:	Unlikely to change – the framework should be tested on real world problems.
	Impact:	Moderate impact on framework and design decisions.

Table 45: Factor - Military is averse to risky new technologies

28	Name:	Military is averse to risky new technologies
	Type:	Organizational
	Description:	It's risky to try to be the first one to do something for military training – the military tends to use things that have proven themselves out already. Compared to the gaming world there is a lot of resistance to change and new technology in the military.
	Reference:	Smith, Roger interview, q. 6. Stuart interview, q. 12.
	Flexibility:	Unlikely to change
	Impact:	Moderate impact on types of components built for military use.

Table 46: Factor - Lack of originality in serious games

29	Name:	Lack of originality in serious games
	Type:	Product
	Description:	With the serious games out now there's no real originality - they all are training soldiers how to use equipment. We need more serious games that concentrate on team training or training to interact with other cultures – a big need area right now.
	Reference:	Smith, Roger interview, q. 3. Smith, Roger interview, q. 4.
	Flexibility:	May change as gaming companies lead the way in these areas.
	Impact:	Minor impact on content.

Table 47: Factor - Divergence of technology

30	Name:	Divergence of technology
	Type:	Technological
	Description:	There has been a divergence of technology from things like hardware, shader languages, and the graphics technologies based around OpenGL and DirectX. Competing tools, languages, and products are starting to look more and more different which by nature make games less compatible across different systems.
	Reference:	Stuart interview, q. 3.
	Flexibility:	Difficult to know if this trend will continue and to what extent it will affect the industry. Will probably change as game studio demand the ability to release similar content on multiple platforms.
	Impact:	Large impact on the framework's capability to support multiple platforms and large impact on limits of component reusability.

Table 48: Factor - Abstract over-engineering

31	Name:	Abstract over-engineering
	Type:	Product
	Description:	One major risk of CBSE is that you can fall into the trap of doing a lot of abstract over-engineering that has little to do with real solutions to real problems – the effort in the end may not allow you to deliver anything concrete
	Reference:	Szyperski interview, q. 4. Szyperski interview, q. 13.
	Flexibility:	Unlikely to change – the framework should be tested on real world problems.
	Impact:	Moderate impact on framework and design decisions.

Table 49: Factor - Self-driven components

32	Name:	Self-driven components
	Type:	Product
	Description:	You will lose control of the deployed environment if you use self-driven components – those components that dynamically try to interface with other components at run-time.
	Reference:	Szyperski interview, q. 14.
	Flexibility:	Inflexible – this will likely always be the case.

	Impact:	Moderate impact to architecture decisions and component design.

Table 50: Factor - Security in the PC environment

33	Name:	Security in the PC environment
	Type:	Technological
	Description:	There is no guarantee of security on a PC. Simulations are susceptible to local and network-based viruses and other security flaws that continue to be found in the operating system. Verifying security is an important issue. Within the component framework you need to ensure only safe code is run.
	Reference:	Zyda interview, q. 4. Zyda interview, q. 14.
	Flexibility:	As technology advances this factor will ideally be mitigated, but if history is a guide this will always be an issue.
	Impact:	Large impact on architecture and design decisions relating to component security and deployed run-time environment security.

Table 51: Factor - Component reuse difficulties: interface complexity

34	Name:	Component reuse difficulties: interface complexity
	Type:	Product
	Description:	Components that have more complicated interfaces are more difficult to reuse. Keep things as simple as possible.
	Reference:	Garfunkel interview, q. 11. Garfunkel interview, q. 12.
	Flexibility:	May be changeable by reducing component interface complexity
	Impact:	Moderate impact on components with complex interfaces and other components with those interface dependencies.

Table 52: Factor - Component protection and licensing

35	Name:	Component protection and licensing
	Type:	Product
	Description:	When you use an open architecture in a component-based software framework you are supporting components written by many people. All that content must be protected and licensed properly.

	Reference:	Garfunkel interview, q. 3. Smith, Peter interview, q. 14.
	Flexibility:	Unchangeable – this is a fundamental factor that a component architecture must address
	Impact:	Large impact – affects all components.

Table 53: Factor - Emergence of dedicated physics cards

36	Name:	Emergence of dedicated physics cards
	Type:	Product
	Description:	One of the things that's just starting to emerge for PC gaming is a dedicated physics card. Like a graphics card, the physics card uses hardware and a dedicated processing unit to calculate physics models and equations in a game or simulation
	Reference:	Gourlay interview, q. 14
	Flexibility:	Physics cards may or may not be a success – it is too early to tell
	Impact:	Large impact on how the framework and other components use physics.

APPENDIX C: PHASE II RESULTS – ISSUES, SOLUTIONS AND STRATEGIES

Table 54: Issue - Adoption of a component-based architecture

1	Name:	Adoption of a component-based architecture	
	Description:	For an organization to be able to adopt the processes and invest the effort required to re-engineer existing and future product lines around a component-based architecture, a number of concerns will have to be addressed. Specifically it must be shown that the CBSA allows product line reuse, mitigates the risks of black box component use, allows the organization to effectively leverage existing and future middleware, and supports the structure and environment required to interface with legacy systems and use legacy code.	
	Influencing factors:	No.	Name
		23	Legacy code integration
		4	Black box component use
		3	Product line reuse
		1	Leveraging middleware
	Design Solution:	Make integration with the framework simple and encourage componentization, but do not enforce it. Provide a dedicated infrastructure for the use of non-componentized libraries.	
	Architectural Strategies:	***Framework component dedicated to interfacing with non-componentized code:*** Create a single framework component that will be tailored to providing interaction between the component-based simulation and non-component, legacy software. This component will present an API that will allow non-component code to call into and receive events from the component-based infrastructure.	
	Related Strategies:	-Separate content components from framework components	

Table 55: Issue - Market forces facing game studios

2	Name:	Market forces facing game studios	
	Description:	In a time of increasing game budgets and corresponding team sizes, game studios face difficulties managing increasingly large projects, a lack of talented developers and artists, and a problem maintaining content originality, technological competitive advantage, and distinction in an expanding marketplace.	
	Influencing factors:	No.	Name
		2	Competitive advantage
		14	Increasing game budgets and team sizes
		26	Gaming interoperability
	Design Solution:	The architecture will support customization of infrastructure and 3rd party components to help resolve scalability issues and allow companies to maintain distinction of content.	
	Architectural Strategies:	***Configuration-based component customization:*** Each component will be configurable, allowing each user of the component to customize how the component is initialized and run. ***Replaceable event manager:*** The event manager will be implemented as a replaceable component. This will allow scalability and flexibility in the core simulation communication infrastructure. ***Tailored events:*** The data structure corresponding to each simulation event can be changed and updated to reflect the priorities of each implementation of the simulation.	
	Related Strategies:	-Event-based component interface -Configurable event data	

Table 56: Issue - Differences between gaming and military content

3	Name:	Differences between gaming and military content	
	Description:	Fundamental differences exist between gaming and military content in terms of its objectives, shelf life, quality, optimization, and fidelity. A common architecture used as a platform to develop content for both industries must explicitly address and support these differences.	
	Influencing factors:	No.	Name
		9	Differing gaming and military content objectives
		6	Differing gaming and military content shelf life
		7	Differing gaming and military content quality
		17	Differing gaming and military content optimization
		11	Differing gaming and military content fidelity
	Design Solution:	Allow for the difference between gaming and military content, but minimize the impact of replacing content and minimize content dependencies.	
	Architectural Strategies:	***Separate content components from framework components:*** Framework components are singletons in the simulation that provide core services: graphics engine, physics engine, network communication, event management. Content components are non-singletons that pertain to the function and state of entities in the simulation: flight models, systems models, graphical displays, graphical entity representations. Framework components are those that may be developed once and reused across many simulations in a particular domain. Content components may be replaced as often as needed to suit the needs of a particular simulation implementation.	
	Related Strategies:	-Wrap risky technologies in components	

Table 57: Issue - Support for military training

4	Name:	Support for military training	
	Description:	In order for a virtual simulation platform supporting game technology to be used for military training, a number of concerns need to be addressed. The architecture should support future decisions made regarding the scientific study of the application of gaming technology to military training objectives. It should support creation of content driven by training objectives as well as analysis of student progress through scenarios.	
	Influencing factors:	No.	Name
		13	Tie-in to learning management system
		8	Lack of science behind military gaming technology
		10	Training objectives drive technology choices
		29	Lack of originality in serious games
	Design Solution:	Use an infrastructure that supports the requirements for logging, playback, and learning management system tie-in required by military training systems. Encapsulate risky technology in separate components.	
	Architectural Strategies:	***Persistent events:*** Each event will incorporate data that represents component state. Data corresponding to the event will be persistent and accessible until a new instance of the event replaces the data.	
	Related Strategies:	-Event-based component interface	

Table 58: Issue - Component reuse

5	Name:	Component reuse	
	Description:	A number of challenges face the reuse of components within virtual simulations. These challenges may also face component design in general. The architecture should, however, explicitly address how this issue will be mitigated in the virtual simulation domain.	
	Influencing factors:	No.	Name
		16	Component reuse difficulties: close ties to domain and context.
		15	Component reuse difficulties: different purpose and different interface.
		22	Component reuse difficulties: many dependencies
		34	Component reuse difficulties: interface complexity
	Design Solution:	Create a component interface that is simple, flexible and negotiable.	
	Architectural Strategies:	***Event-based component interface:*** All components will communicate to each other and the rest of the infrastructure through events. Events will be uniquely identified and incorporate data that represents component state. ***Configurable event data:*** Components will notify the infrastructure of required events and event data when the component is registered. Specific events and event data that will actually run in the implementation of the simulation will be determined at configuration-time.	
	Related Strategies:	-Tailored events -Configuration-based component customization	

Table 59: Issue - Component architecture development

6	Name:	Component architecture development	
	Description:	A number of challenges face the use of a component architecture for virtual simulations. These challenges may also face the discipline of component architecture in general. The architecture should, however, explicitly address how these problems will be addressed in the virtual simulation domain.	
	Influencing factors:	No.	Name
		35	Component protection and licensing
		18	Backwards compatibility and version upgrades
		19	Component engineering effort
		20	Component performance
		21	Component framework complexity
	Design Solution:	Handle protection, licensing, and versioning together. Support individual component licenses. Ensure only one version of a component is active at a time but allow version negotiation and replacement.	
	Architectural Strategies:	***Registration and licensing managers:*** Registration and licensing managers will be used to handle component licensing, versioning, and interface specification. When a component registers with the infrastructure it will identify its run-time licensing requirements, its version, and its interface requirements.	
	Related Strategies:	-Implement component interface constraints	

Table 60: Issue - Component framework implementation

7	Name:	Component framework implementation	
	Description:	A number of challenges face the implementation of a component framework for virtual simulations. These challenges may also face the implementation of a generic component framework. The architecture should, however, explicitly address how these problems will be addressed in the virtual simulation domain.	
	Influencing factors:	No.	Name
		24	Domain model componentization
		27	Built-in assumptions of a generic platform
		31	Abstract over-engineering
		25	Development in a vacuum or lab environment
	Design Solution:	Ensure that the architecture supports the major virtual simulation domain models currently in use.	
	Architectural Strategies:	***Componentized virtual simulation domain models:*** Break the major gaming and military virtual simulation genres into components to ensure the architecture offers the potential to support them: action, role-player, real-world, strategy, and distributed.	
	Related Strategies:	-Separate content components from framework components.	

Table 61: Issue - Security and military technology

8	Name:	Security and military technology	
	Description:	While the military is interested in the benefits gained from using gaming technology, it is equally or more concerned with protecting its own technology from being abused, indiscriminately distributed, and potentially misused to compromise national secrets.	
	Influencing factors:	No.	Name
		28	Military is averse to risky new technologies
		33	Security in the PC environment
		5	Confidentiality of military technology in games
		32	Self-driven components
	Design Solution:	Implement basic security policies to help counteract malicious use of the infrastructure.	
	Architectural Strategies:	***Implement component interface constraints:*** Do not allow run-time component and event registration, subscription, or publication. Allow run-time event subscription activation and deactivation only.	
	Related Strategies:	-Event-based component interface -Registration layer -Wrap risky technologies in components	

Table 62: Issue - Technology trends

9	Name:	Technology trends	
	Description:	It is important that the architecture consider current technology trends in virtual simulation and gaming related to graphics, physics, distributed environments, as well as trends in other related fields.	
	Influencing factors:	No.	Name
		30	Divergence of technology
		12	Increasingly realistic gaming graphics
		36	Emergence of dedicated physics cards
	Design Solution:	Encapsulate new and diverging technology to help mitigate the risks of using it.	
	Architectural Strategies:	***Wrap risky technologies in components:*** Ensure those parts of the simulation that represent the use of technology that is likely to change are wrapped within a single component: physics engine component, OpenGL graphics engine component, DirectX graphics engine component	
	Related Strategies:	-Separate content components from framework components	

APPENDIX D: PHASE II RESULTS – ARCHITECTURE DESCRIPTION

This appendix provides the full description of the software architecture.

Context

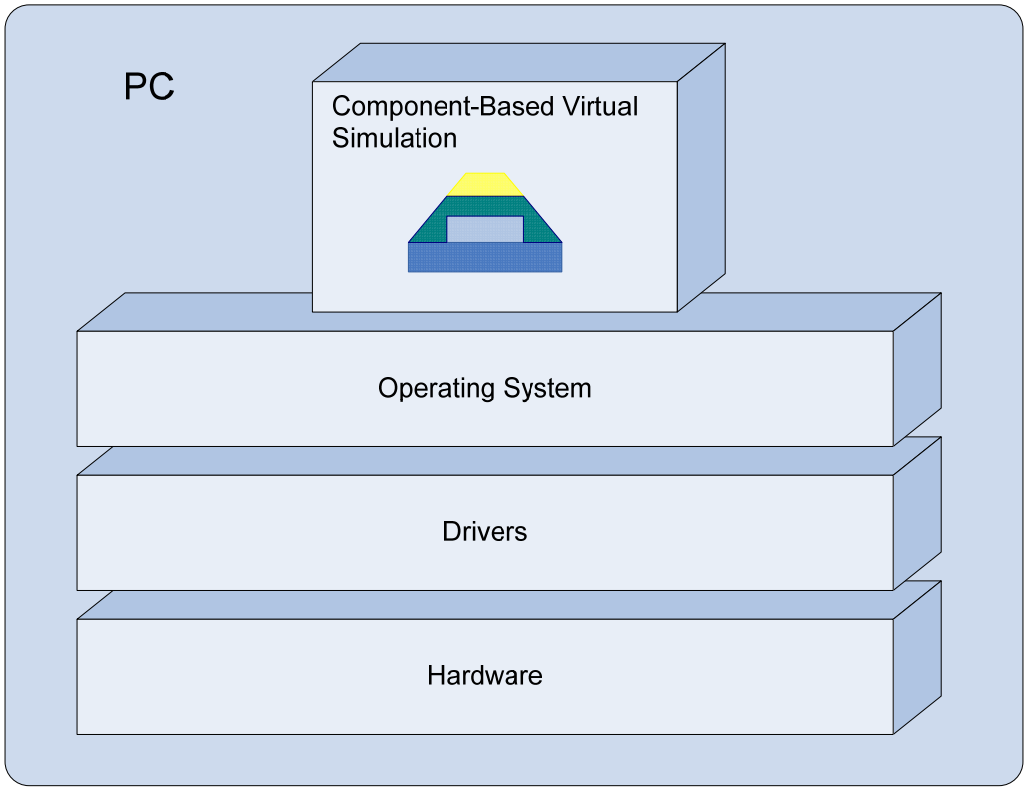

Figure 7: Context of the Component-Based Virtual Simulation

In the interest of providing context and perspective, the deployed component-based virtual simulation is designed to run on a PC-based operating system, but it is not meant to be strongly tied to that environment. It supports deployment as a stand alone application or as a set of libraries run in the context of a larger application.

Layers

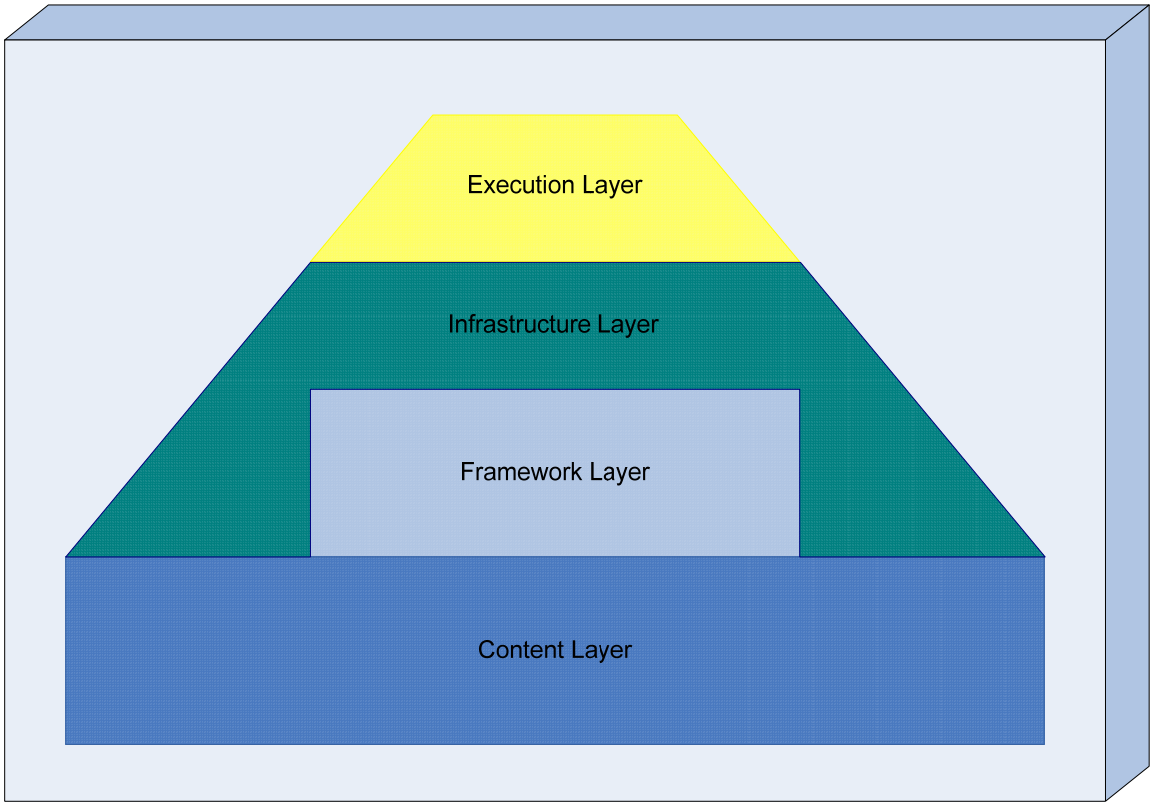

Figure 8: Component-Based Virtual Simulation Layers

The architecture consists of four layers whose responsibilities are as follows:

- *Execution Layer:* The execution layer establishes the run-time environment for the component-based simulation. It is responsible for initialization, window and rendering context setup, execution context and environment, and termination.
- *Infrastructure Layer:* The infrastructure layer consists of core components which handle simulation component loading, registration, licensing, configuration, and execution, as well as management of the simulation lifecycle.

- *Framework Layer:* The framework layer consists of singleton simulation components whose lifetimes span the lifecycle of each simulation scenario. It is responsible for defining behavior of the simulation's virtual environment and tools.
- *Content Layer:* The content layer consists of simulation components whose lifetimes span the lifecycles of entities within the simulation. It is responsible for defining entity behaviors.

Component Classes

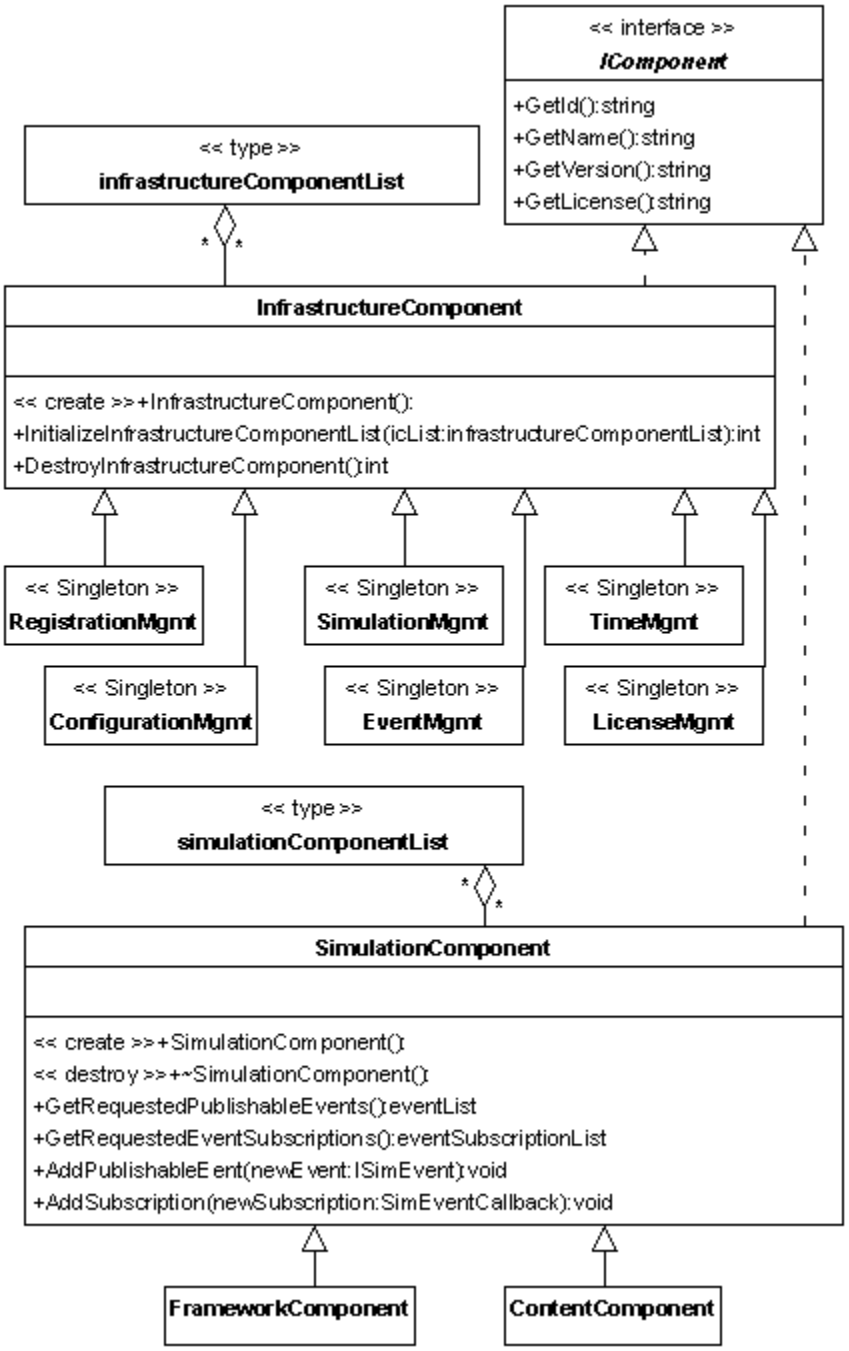

Figure 9: Components Class Diagram

A component is a uniquely identified and versioned, independently deployable, licensed subset of simulation functionality. Components must conform to interface

IComponent for purposes of identification, version resolution, registration, and license negotiation. There are two types of components:

- *Infrastructure components:* There are six infrastructure components that reside in the infrastructure layer which provide the underlying core functionality of the simulation. Each infrastructure component conforms to a contract that defines its interface to its peers. These components are loaded and initialized by the execution layer which provides each with a reference to a list of all loaded infrastructure components. The six infrastructure components are RegistrationMgmt, ConfigurationMgmt, SimulationMgmt, EventMgmt, TimeMgmt, and LicenseMgmt.
- *Simulation components:* There are an undefined number of simulation components that reside in the framework and content layers which define the overall behavior of the simulation. Simulation components communicate to their peers solely through the use of events. The two types of simulation components, framework and content components, reside in their respective layers.
- *Framework components:* Framework components are configured and run as singletons per scenario, and their lifecycle length is the same as that of the scenario.
- *Content components:* Content components are configured and run per entity in each scenario. Each entity in a scenario is an aggregate of the functionality of all of its content components. A content component's lifecycle corresponds to that of the entity to which it is attached.

Infrastructure Layer

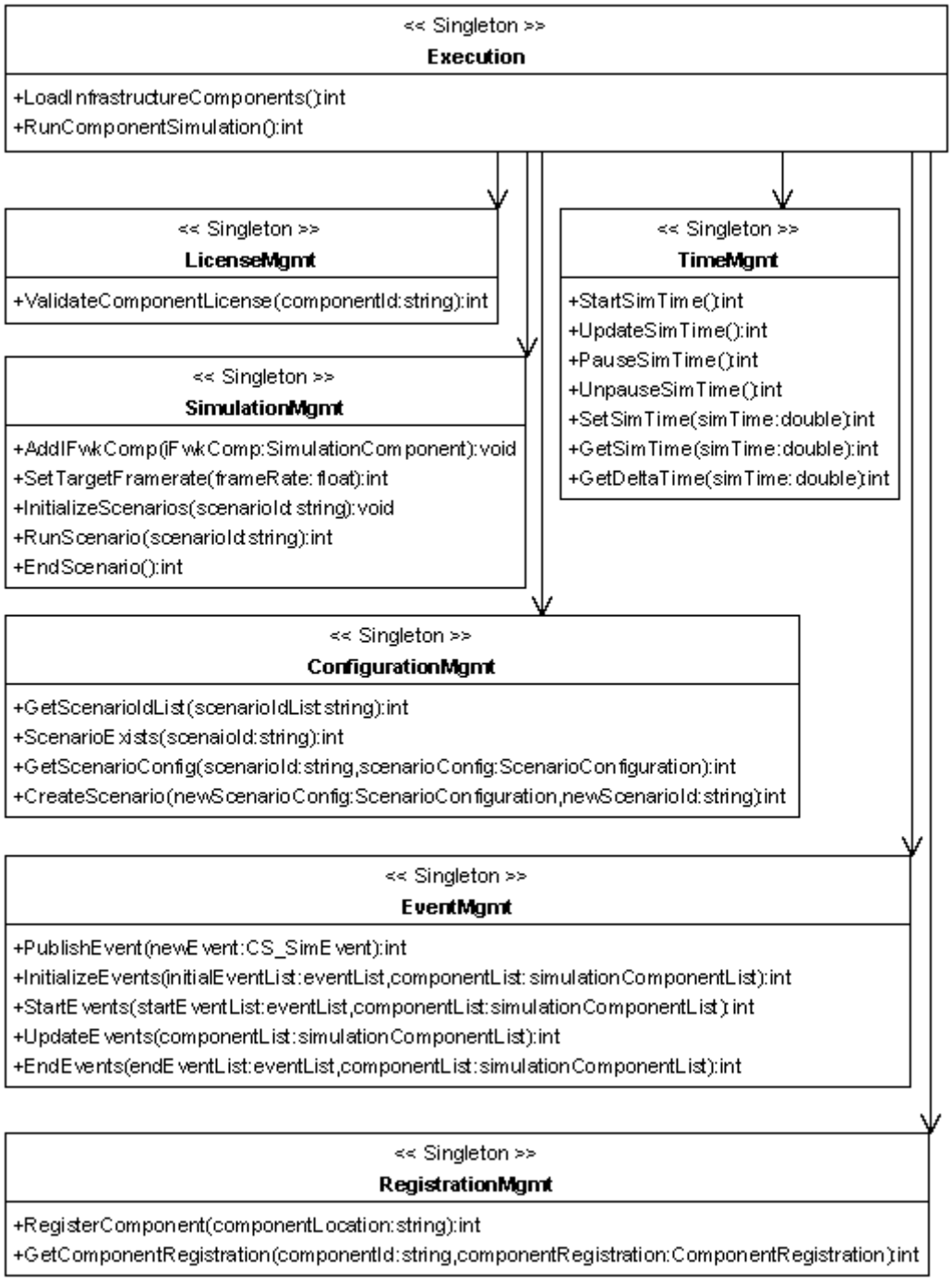

Figure 10: Infrastructure Layer Class Diagram

The following list provides a description of the six infrastructure components that make up the infrastructure layer:

- *RegistrationMgmt:* Responsible for registering each component, maintaining and updating that information, and providing component registration information to other infrastructure components. Information gathered by RegistrationMgmt can be used to resolve component identification, communication, and security issues.
- *LicenseMgmt:* Responsible to ensure each component that will run has a valid license.
- *ConfigurationMgmt:* Responsible for providing initial states for all components and maintaining entity and scenario configurations.
- *SimulationMgmt:* Responsible for initializing, running, and terminating each scenario through the instantiation and destruction of all simulation components and the control of the TimeMgmt and EventMgmt infrastructure components.
- *TimeMgmt:* Responsible for maintaining current simulation time.
- *EventMgmt:* Responsible for providing the event communication mechanism for all simulation components. EventMgmt uses persistent overwritten events with a one-to-many event-to-callback relationship. Events are persistent in that once an event is published it is not destroyed until the end of the scenario. The most recent event of a given event name, along with its data, is retained as the current event, overwriting the previously published event of the same

name. Multiple simulation components can subscribe to a given event name and register callback functions to the event.

Infrastructure Component Lifecycle

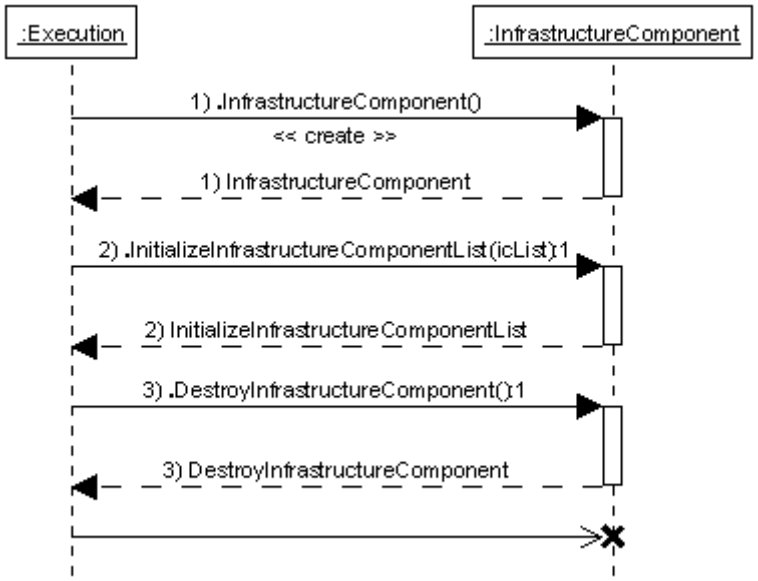

Figure 11: Infrastructure Component Lifecycle

The six infrastructure components are located, loaded, and instantiated by the execution layer. Once loaded, each infrastructure component is initialized with a list of references to the other infrastructure components to allow peer-to-peer communication. The interfaces for this communication are governed by an interface contract for each of the components.

Component Registration Sequence

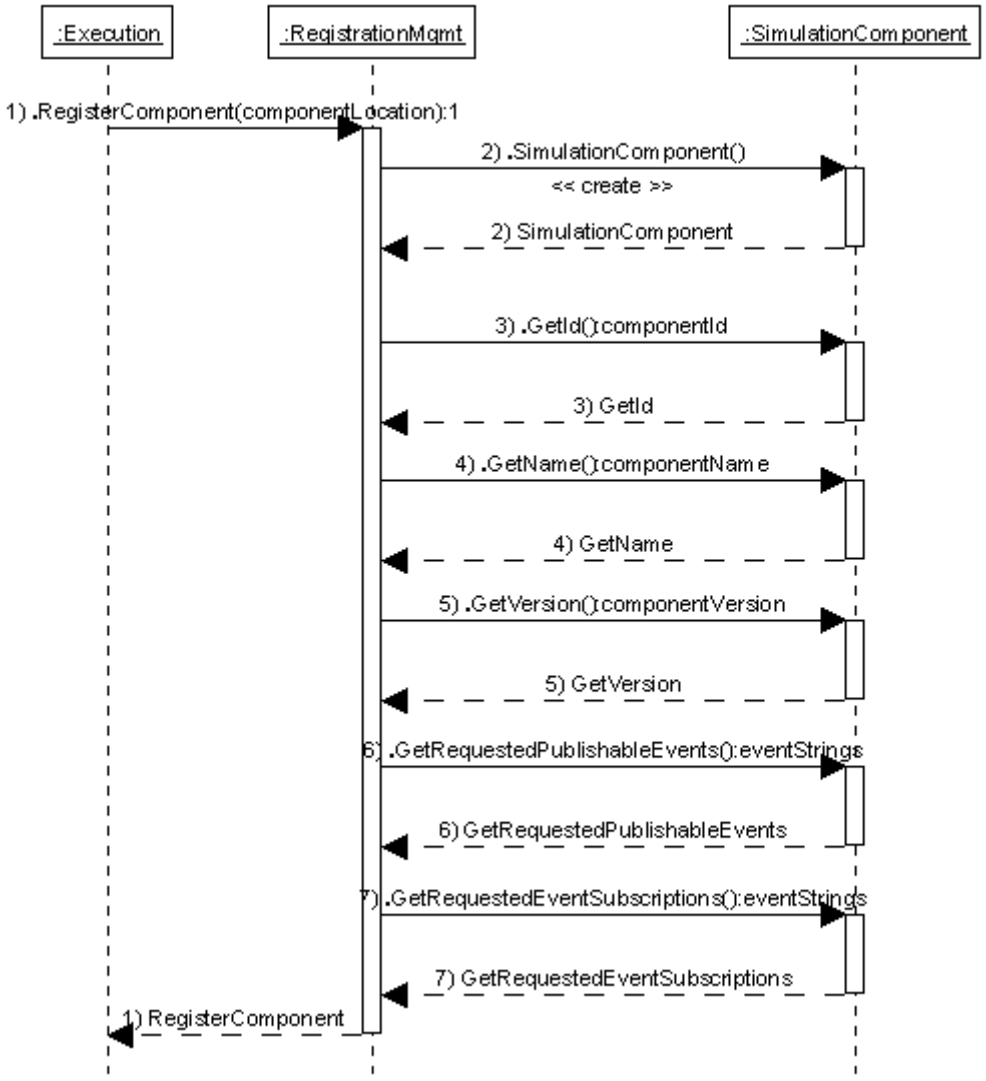

Figure 12: Component Registration Sequence Diagram

In order for a component to be configured and run in a scenario, it must be registered with RegistrationMgmt. RegistrationMgmt gets the component's id, name, version, and a list of requested publishable and subscribable events. This information is used for configuring entities and scenarios, enforcing simulation component event

communication paths and security, and resolving component id, version, and naming conflicts.

Configuration Classes

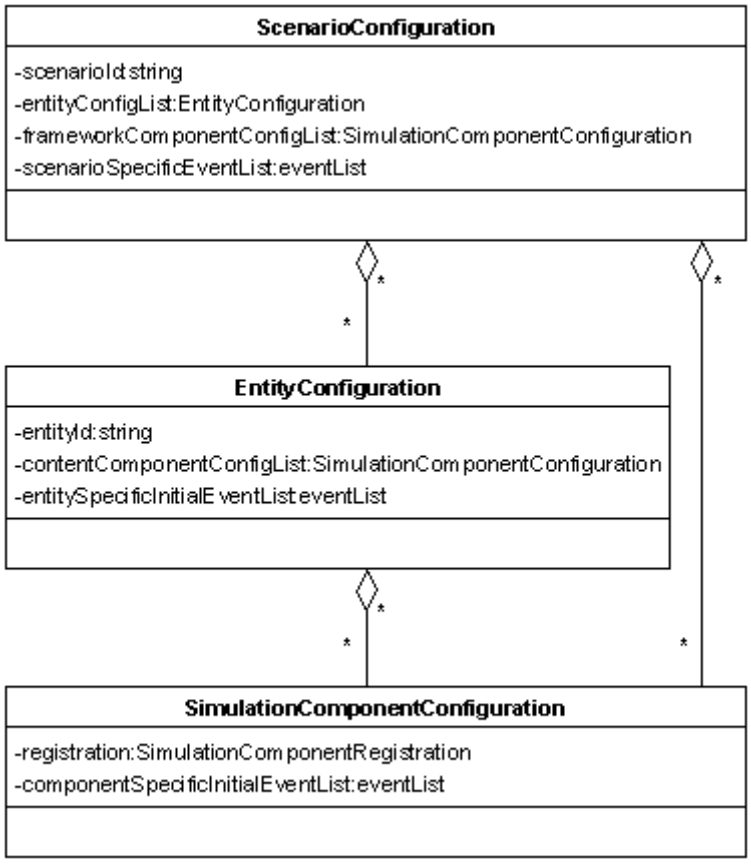

Figure 13: Configurations Class Diagram

There are three types of configurations:

- *Simulation Component Configuration:* contains a reference to a simulation component's registration information and specifies an initial event list for the component.
- *Entity Configuration:* contains an entity id, a list of simulation content component configurations that specifies content components that make up the entity, and an initial event list for the entity.

- *Scenario Configuration:* contains a scenario id, a list of entity configurations that make up the scenario, a list of simulation framework component configurations that specifies framework components that will run in the scenario, and an initial event list for the scenario.

Scenario Lifecycle

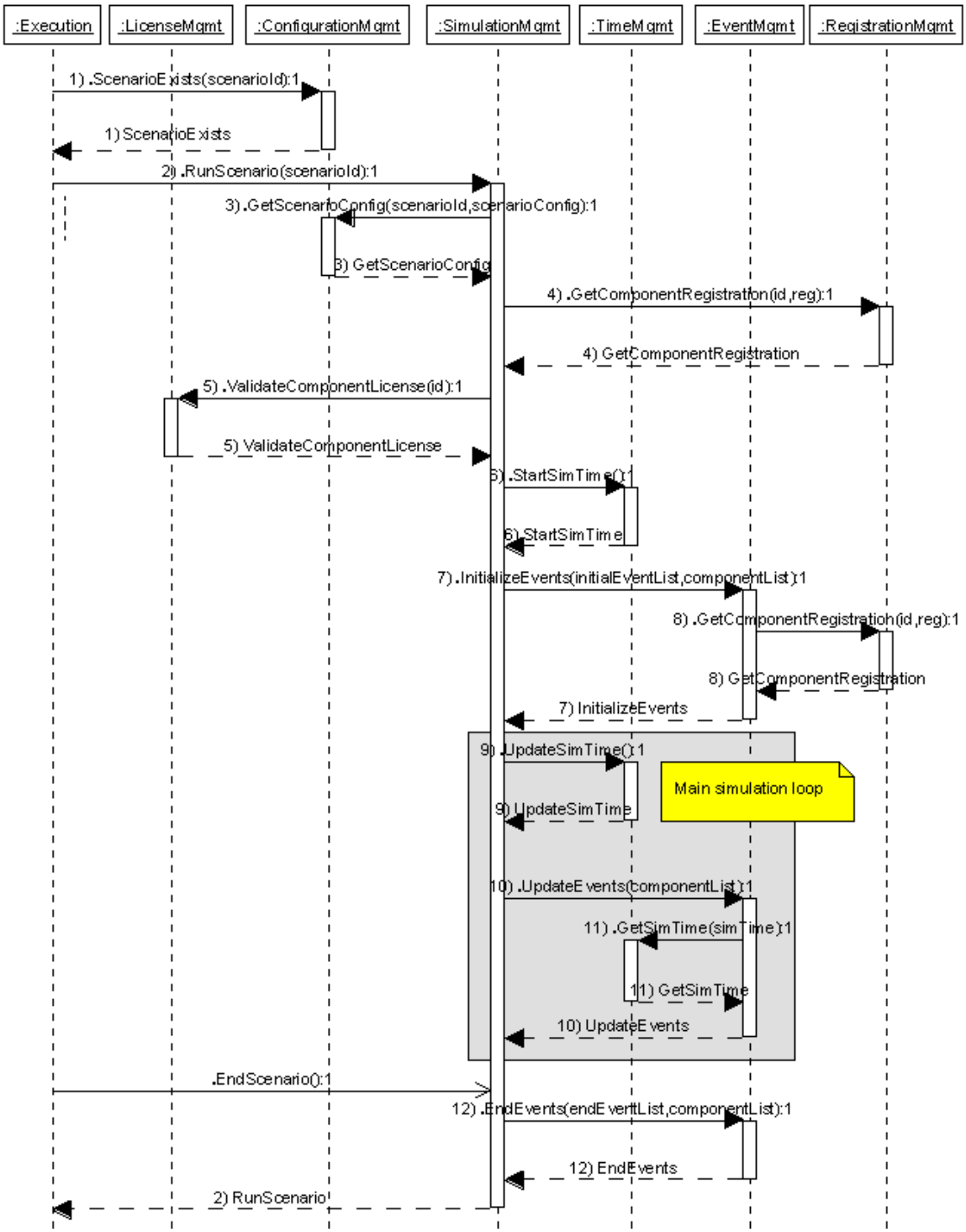

Figure 14: Scenario Lifecycle Diagram

Following is a summary of the scenario lifecycle:

- *Initialization:* The execution layer chooses a scenario to run and tells SimulationMgmt to run that scenario. SimulationMgmt starts the thread for that scenario, gets the scenario configuration information from ConfigurationMgmt, and ensures all the simulation components specified by the scenario configuration are properly registered and licensed. It then starts the simulation clock and tells EventMgmt to fire initial events on the given list of simulation components. EventMgmt pulls the registration information for each component, records which events simulation components may publish and which they may subscribe to, then fires initial events for each simulation component.
- *Main Simulation Loop:* After EventMgmt completes initialization, SimulationMgmt enters the main simulation loop. On each cycle it updates the simulation clock and then tells EventMgmt to update all events in the given list of simulation components. EventMgmt finds all published events from each component, checks each against the component's registered publishable events, then updates each component with the new events, again checking against the component's registered subscribable events. Once the event update is complete, EventMgmt returns control to SimulationMgmt.
- *Destruction:* When SimulationMgmt gets a call to end the scenario from the execution layer, it halts the main simulation loop and tells EventMgmt to fire end events on the given list of simulation components. Once EventMgmt

returns, SimulationMgmt ends the scenario thread and returns control to the execution layer.

Simulation Component Lifecycle

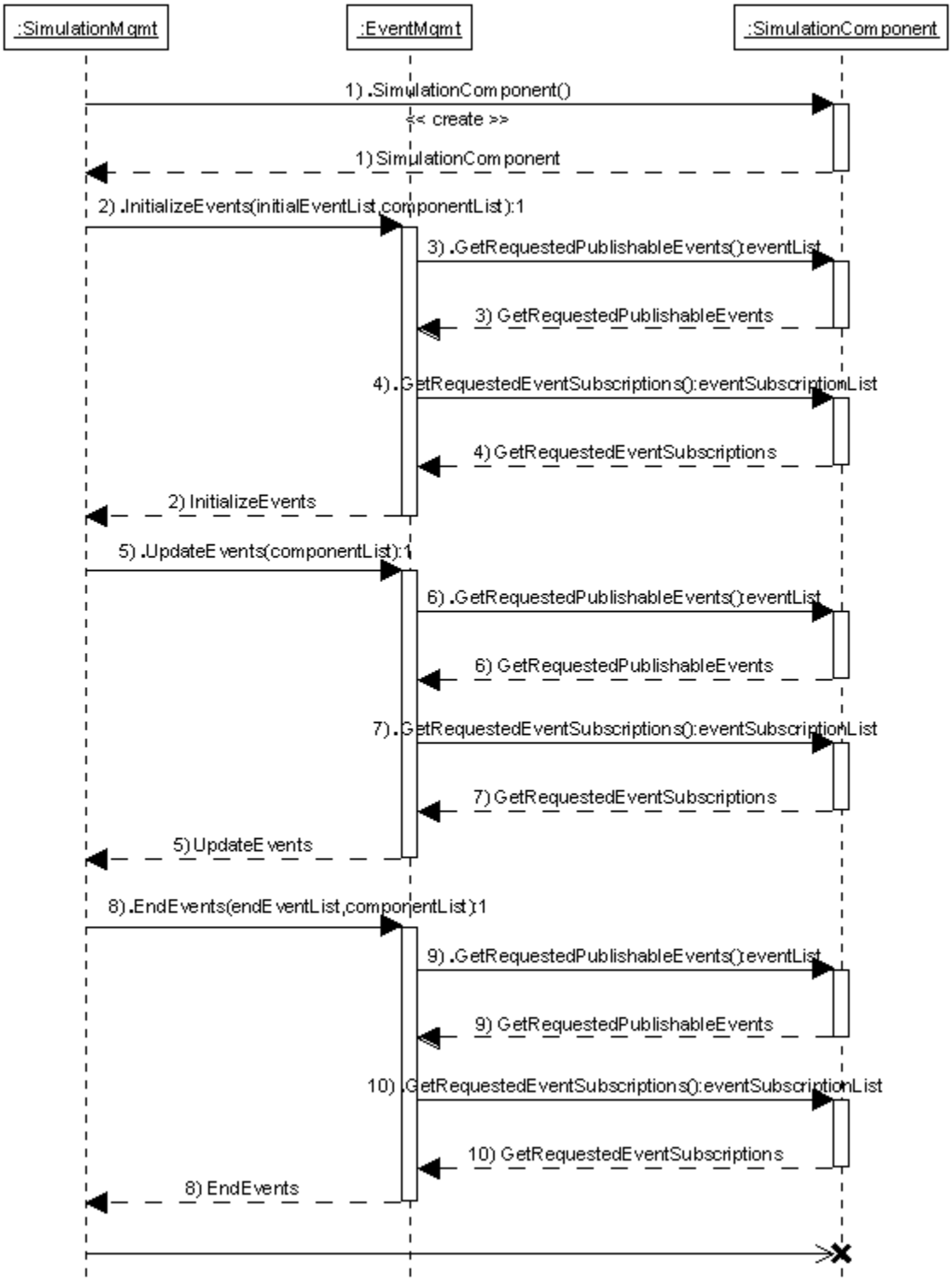

Figure 15: Simulation Component Lifecycle Diagram

The following describes a simulation component's lifecycle:

- *Creation:* Simulation components are instantiated by SimMgmt based on the components required by the current scenario.
- *Initialization:* Each component's state is initialized by EventMgmt through a set of events defined during registration and configuration.
- *Simulation Loop:* During the main simulation loop EventMgmt gets events published by each component and ensures the events are publishable by the component based on each component's registration. EventMgmt then provides callback service for all events published to which the component has subscribed.
- *End:* Each component's state is finalized by EventMgmt through a set of events defined during registration and configuration.
- *Destruction:* Simulation components are destroyed by SimMgmt at the end of the scenario.

Event Classes

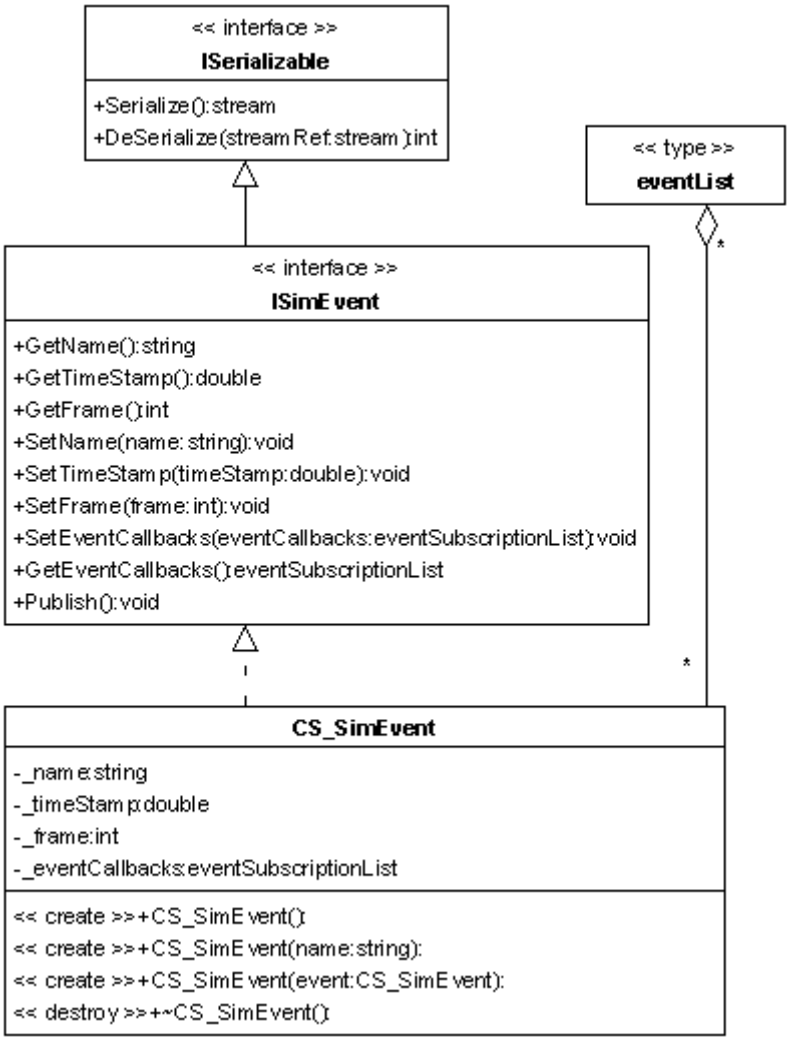

Figure 16: Event Class Diagram

Every event used in the simulation must subclass SimEvent and therefore must implement the interfaces *ISerializable* and *ISimEvent*. When created a SimEvent must at a minimum contain its identifying event name. The creator may also add any data desired to the sub-classed event. The EventMgmt infrastructure component will add the frame and timestamp when the event is published. Every event is serializable so the

infrastructure (and possibly other simulation components) can use the event without knowledge of the data types or data contained in the event.

Event Lifecycle

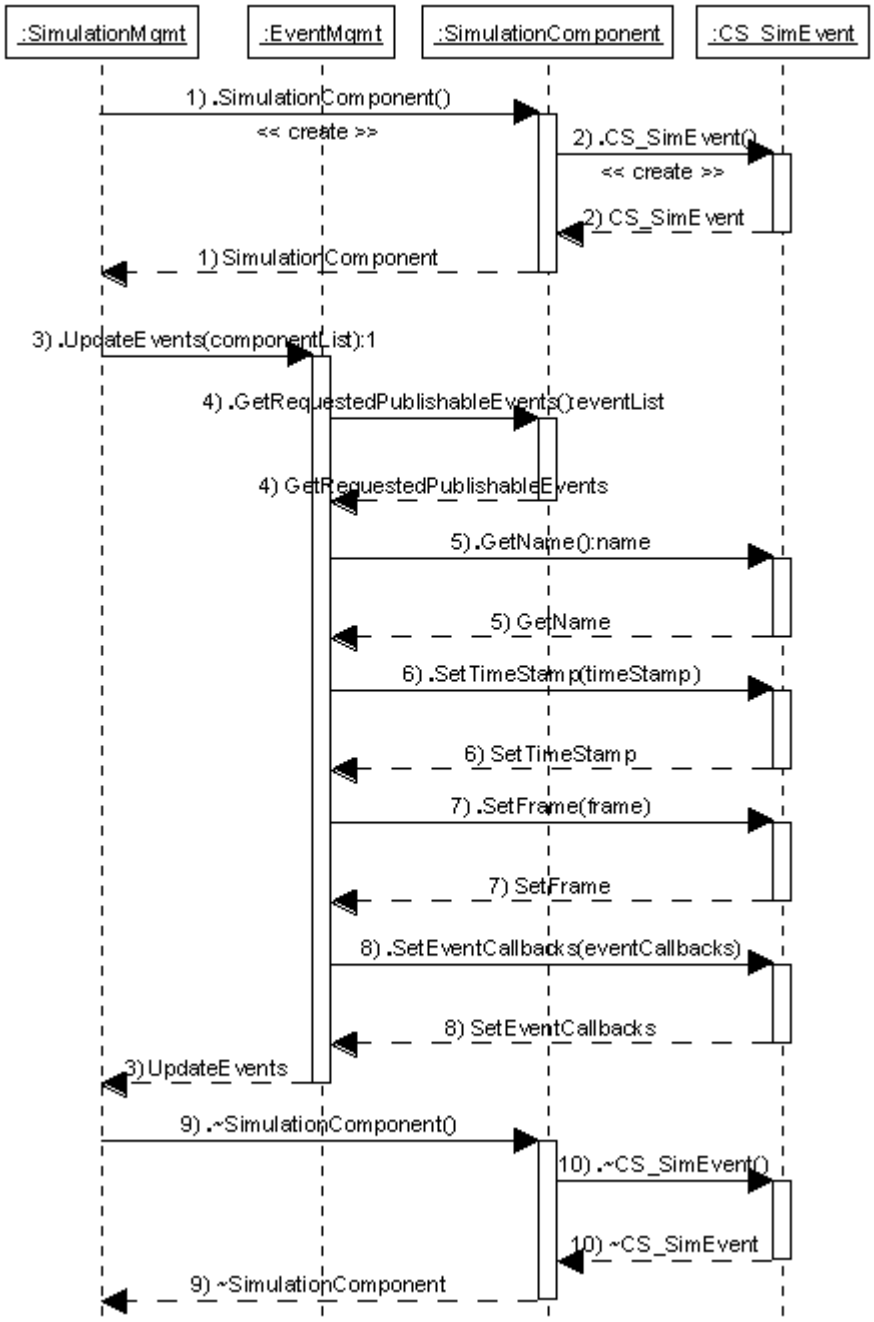

Figure 17: Event Lifecycle Diagram

When a simulation component chooses to publish an event, it creates the event with the appropriate event name and adds the appropriate data to its event subclass. It returns the list of published events to EventMgmt which copies, stores, and timestamps the event. EventMgmt then fires the associated event callbacks of the event subscribers. The simulation component can then destroy the event on the next frame. EventMgmt retains its copy of the event until the end of the scenario.

Event Callback Class

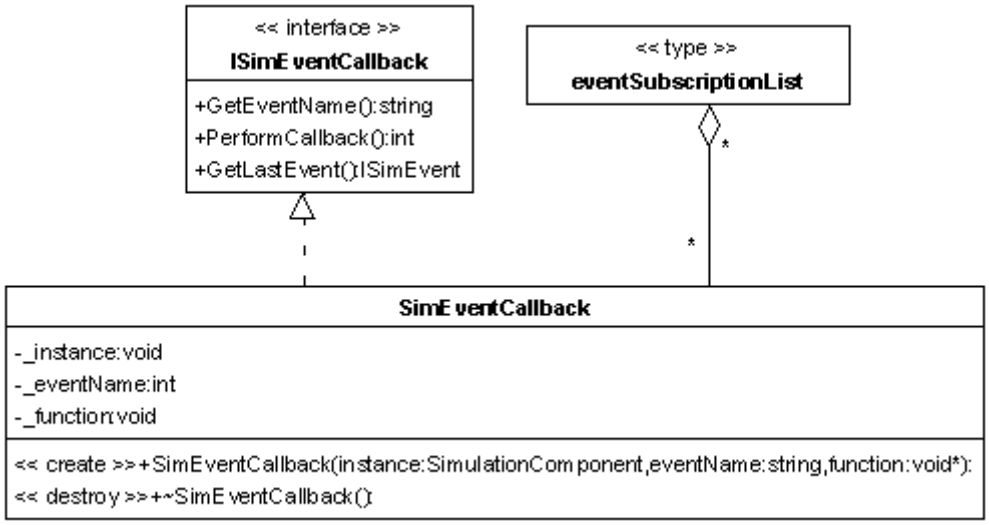

Figure 18: Event Callback Class Diagram

Every event callback is of type SimEventCallback which implements the ISimEventCallback interface. The EventMgmt infrastructure component associates each event with a series of registered callbacks contained in simulation components and maintains the association through the use of the event's name.

Event Callback Lifecycle

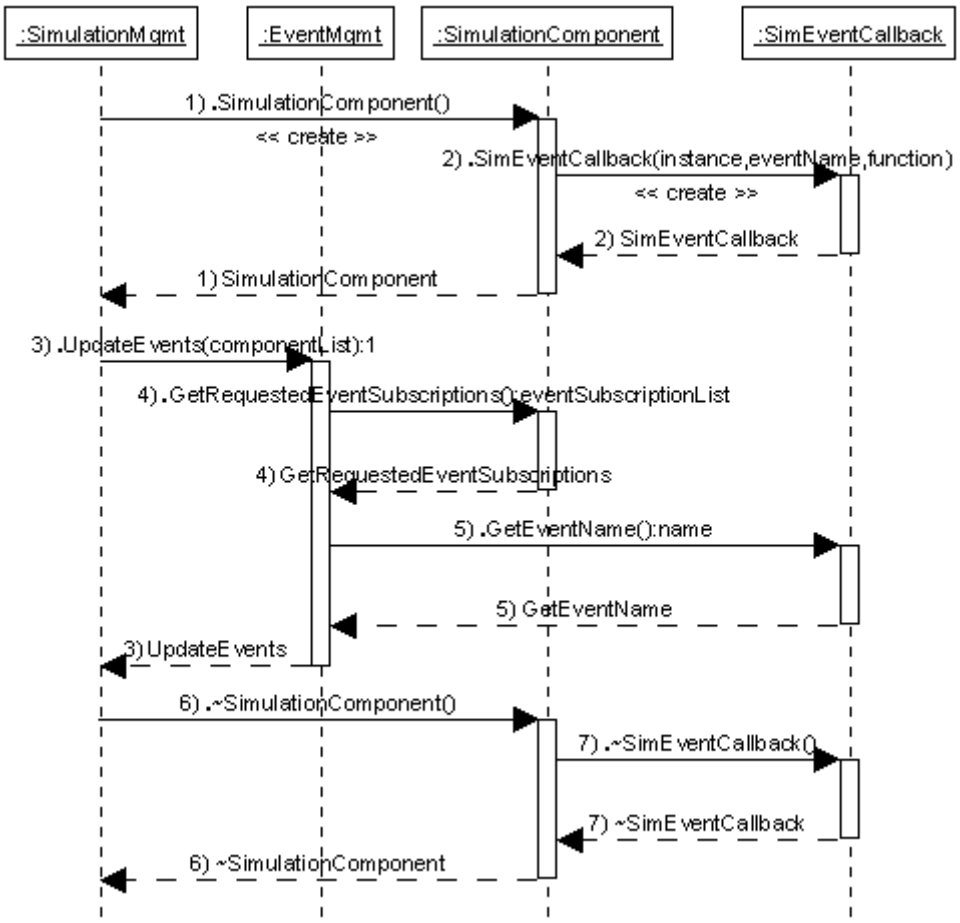

Figure 19: Event Callback Lifecycle Diagram

The event callback is created for each event a simulation component would like to listen to. During callback creation, the simulation component identifies its own instance, the name of the event to listen for, and the pointer to the function that will receive the callback. The list of callbacks for each simulation component is passed back to EventMgmt which associates the callback with a particular event through the event name. When the event is published EventMgmt makes a call to the appropriate callback function.

APPENDIX E: GLOSSARY OF ARCHITECTURE TERMS

The following terms are used in the documentation of the architecture and are defined here.

- **Component -** A component is a uniquely identified and versioned, independently deployable, licensed subset of simulation functionality.
- **Configuration** – A configuration is a list that references a set of simulation component registrations and a set of initial and end events.
- **Configuration Management** – Configuration management represents an infrastructure component that is responsible for providing initial states for all components and maintaining entity and scenario configurations.
- **Connector** – A connector is a mechanism that allows components to communicate with each other.
- **Content Component** – A content component is a simulation component that is configured and run per entity in a scenario and whose lifecycle corresponds to the entity to which it is attached.
- **Content Layer** – The content layer consists of all of the simulation's content components. It is responsible for defining entity behaviors.
- **Entity** – An entity is a configuration that represents an aggregation of the behavior and functionality of a set of content components.
- **Event** – See persistent event.
- **Event Callback** – An event callback represents a simulation component's subscription to an event. It specifies a function that is called when that event is published.

- **Event Management** – Event management represents an infrastructure component that is responsible for controlling the persistent event connector that provides a communication mechanism for all simulation components.
- **Execution Layer** – The execution layer establishes the run-time environment for the component-based simulation. It is responsible for initialization, window and rendering context setup, execution context and environment, and termination. It has a dependency on the infrastructure layer.
- **Framework Component** – A framework component is a simulation component that is configured and run as singleton in a scenario and whose lifecycle is the same as that of the scenario.
- **Framework Layer** – The framework layer consists of all of the simulation's framework components. It is responsible for defining behavior of the simulation's virtual environment and tools. It has a dependency on the content layer.
- **Infrastructure Component** – An infrastructure component is a component that provides core, underlying functionality to the simulation. There are six infrastructure components that are responsible for component licensing, component registration, component configuration, event management, time management, and simulation management.
- **Infrastructure Component Reference** – An infrastructure component reference represents a connector that allows peer-to-peer infrastructure component communication.

- **Infrastructure Layer** – The infrastructure layer consists of infrastructure components which handle simulation component loading, registration, licensing, configuration, and execution, as well as management of the simulation lifecycle. It has a dependency on the framework and content layers.
- **License Management** – License management represents an infrastructure component that is responsible to ensure each component that will run has a valid license.
- **Persistent Event** – A persistent event is a connector that allows peer-to-peer simulation component communication. It is persistent because its data is maintained persistently until superseded by a newer persistent event of the same name.
- **Registration** – A registration is a set of information about a component that includes its name, location, and its set of publishable and subscribable events.
- **Registration Management** – Registration management represents an infrastructure component that is responsible for registering each component, maintaining and updating that information, and providing component registration information to other infrastructure components. Information gathered by registration management can be used to resolve component identification, communication, and security issues.
- **Scenario** – A scenario is a configuration that represents an aggregation of the behavior and functionality of a set of framework components and a set of entities.

- **Simulation Component** – A simulation component is a component that provides simulation-driven functionality or behavior. The two types of simulation components are framework components and content components.
- **Simulation Management** – Simulation management represents an infrastructure component that is responsible for initializing, running, and terminating each scenario through the instantiation and destruction of all simulation components and the control of the time management and event management infrastructure components.
- **Time Management** – Time management represents an infrastructure component that is responsible for maintaining current simulation time.

REFERENCES

Aksit, Mehmet. Software Architectures and Component Technologies. Kluwer Academic Publishers, 2002.

Alexa, Stefan. *America's Army Game: Its (Virtual) Reality Representation and Cocaine*. 2004 International Conference on Cyberworlds. IEEE, 2004.

Baracos, P. et. al. *Enabling PC-Based HIL Simulation for Automotive Applications*. IEEE, 2001.

Batista, H., Costa, V., Pereira, J. *Games of War and Peace: Large Scale Simulation Over the Internet*. Proceedings of the 7th International Conference on Virtual Systems and Multimedia. IEEE, 2001.

Clements, P. *Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000*, Software Architecture Technology Initiative, 2005.

Clements, Paul; Kazman, Rick; Klein, Mark. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, 2002.

Computer Science and Telecommunications Board, National Research Council. Modeling and Simulation: Linking Entertainment and Defense. National Academic Press, 1997.

Erwin, S. *Video Games Gaining Clout as Military Training Tools*, National Defense Magazine, Nov. 2000.

Fong, Gwenda. *Adapting COTS Games for Military Simulation*. Association for Computing Machinery, 2004.

Gamespy. *Take Off and Stay High*, October 2004. Retrieved August 6, 2005 from http://www.gamespy.com/articles/556/556463p1.html.

Gomaa, H. Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley, 2000.

Hamilton, John A., Pooch Udo W. *An Open Simulation Architecture for Force XXI*. Proceedings of the 1995 Winter Simulation Conference, 1296-1303. IEEE, 1995.

Hofmeister, C., Nord, R., Soni, D. Applied Software Architecture. Addison-Wesley, 2000.

Korris, James H. *Full Spectrum Warrior: How the Institute for Creative Technologies Built a Cognitive Training Tool for the Xbox*. 24th Army Science Conference, 2004.

Lewis, M., Jacobsen, J. *Game Engines in Scientific Research*. Communications of the ACM, Vol. 45, No. 1, January 2002.

Lindheim, R., Swartout, W. *Forging a New Simulation Technology at the ICT*. IEEE Computer, 2001.

Macedonia, Michael. *Entertainment Technology and Virtual Environments for Military Training and Education*. Forum Futures 2001, Forum for the Future of Higher Education, 2001.

Macedonia, Michael. *Games Soldiers Play*. IEEE Spectrum. March 2002.

Morris, C., Tarr, R. *Templates for Selecting PC-Based Synthetic Environments for Application to Human Performance Enhancement and Training*. Proceedings of the IEEE Virtual Reality 2002.

Pace, J. et. al. *Accomplishing Adaptability in Simulation Frameworks: the Bubble Approach*. IEEE, 2001.

Prensky, M., *True Believers: Digital Game-Based Learing in the Military,* Digital Game-Based Learning, McGraw-Hill, 2001.

Samara, Shaul. Interview with Shaul Samara, VP of Development for Simigon, on September 9, 2005.

Simigon, Inc. *Company vision and product descriptions of Knowbook and Airbook*. Retrieved August 20, 2005 from www.simigon.com.

Szyperski, C. Component Software: Beyond Object-Oriented Programming. ACM Press, 1998.

Szyperski, C., Messerschmitt, D. Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, 2005.

Van Lent, M. *Integrating Architecture.* Presentation for the SOAR Workshop, 2004.

Zyda, M. et. al. *Entertainment R&D for Defense*. IEEE Computer Graphics and Applications. January/February 2003.