Early Adapters' Lessons: How Other Government Sectors Implemented Open Source Software

Perry McDowell
MOVES Institute, Naval Postgraduate School
Monterey, California
mcdowell@nps.edu

John Scott
Mercury Federal Systems, Inc.
Arlington, Virginia
jscott@mc.com, johnmscott@mindspring.com

ABSTRACT

At I/ITSEC 2007, Dr. Paul Mayberry said during the Flag and General Officer Panel, "Proprietary solutions are short sighted… proprietary is essentially an anathema to those criteria of responsiveness… When I can only go back to one source and their sort-of dedicated workforce, it really is not getting at the fundamental criteria that we have for all of training transformation effects, and that is a notion of being agile, adaptive, responsive, timely and trainable." The US Navy has also been at the forefront of these changes: for example the Department of Navy Chief Information Officer released a policy on accepting the use of open source software (OSS) and Vice Adm. Mark Edwards reiterated it stating, "The days of proprietary technology must come to an end. We will no longer accept systems that couple hardware, software and data." Likewise, OSD in a recent report on Open Technologies has recently begun to see the light and understand that good technologies eventually become a commodity and there is a pronounced requirement to move toward open source software platforms to better respond to the needs of the warfighter.

Taken together, it is obvious that the modeling and simulation community is going to have to implement more open source solutions in order to meet the future needs of the Department of Defense. Like any major change in business practices, this will not always be an easy or popular transformation. However, examining how other sectors of government and industry are incorporating OSS into their products can be instructive to the modeling and simulation industry. The authors present case studies of other government open source implementations and how the lessons learned from these can be applied to modeling and simulation. Additionally, they discuss successful business models companies can use to profit from this new business model.

ABOUT THE AUTHORS

Perry McDowell is a Research Associate at MOVES Institute at the Naval Postgraduate School in Monterey and the Executive Director for the Delta3D Open Source Game Engine Project. He has been on the faculty of the Naval Postgraduate School since 2000, where he teaches and performs research. His research has been primarily focused on training in virtual environments with an emphasis on Naval applications. Currently, he is researching the best ways to implement game based learning into various military curricula. Mr. McDowell is a former Naval Nuclear Power – Surface Warfare Officer, whose sea tours include USS-VIRGINIA (CGN-38), USS ELROD (FFG-55), and USS ENTERPRISE (CVN-65). He graduated with a B.S. in Naval Architecture from the U.S. Naval Academy in 1988, and an M.S. (with honors) from the Naval Postgraduate School in 1995.

John Scott (employed at Mercury Federal Systems) is the project leader of DOD's Open Technology Development Initiative, which lays the groundwork for streamlined adoption of open source methodologies with DOD, which includes both the adoption (including testing) of private sector open source code and the formation of internal communities of interest around DOD systems, including classified systems. He has held senior positions in technology companies, including three start-ups, including management of software development teams and rapid ramp-up of consulting practices. John holds an MS in Systems Engineering from Virginia Tech and a BA in Mechanical Engineering from Lehigh University. He has been a repeat speaker at the O'Reilly Emerging Technology Conferences.

Early Adapters' Lessons: How Other Government Sectors Implemented Open Source Software

Perry McDowell
MOVES Institute, Naval Postgraduate School
Monterey, California
mcdowell@nps.edu

John Scott
Mercury Federal Systems, Inc.
Arlington, Virginia
jscott@mc.com, johnmscott@mindspring.com

INTRODUCTION

The use of open source software (OSS) in business is growing rapidly for a wide variety of reasons. IBM has morphed from being "Big Blue", a company that sold hardware and software solutions that were completely developed in house, to a company that derives the majority of its revenue by providing "solutions" using OSS. In an August 2006 report, the IDC Group wrote that "[open source is] the most significant all-encompassing and long-term trend that the software industry has seen since the early 1980s." (Heiman and Picardi, 2006, as quoted in Vass, 2007)

The military is beginning to see the value in OSS and leaders are beginning to urge the inclusion of OSS in military projects. Vice Admiral Mark Edwards, Deputy Chief of Naval Operations for Communications, said in a speech that

> The days of proprietary technology must come to an end. We will no longer accept systems that couple hardware, software and data… We can't accept the increasing costs of maintaining our present-day capabilities… In the civilian marketplace, it's just the opposite. Some private-sector concerns are cutting their costs by 90 percent while expanding their performance. (Buxbaum, 2008)

However, the military is not primarily concerned with costs – the most important consideration is the military's ability to counter any threat to the nation. Normally, we presume that our large technical infrastructure gives the U.S. an inherent advantage over its adversaries, but Brigadier General Nickolas Justice, U.S. Army Program Executive Officer for Command, Control, Communications Tactical presented a chilling challenge to this belief:

> On the battlefield, will Open Technology allow flexible adversaries to adapt their strategy in less time than it takes to develop our plan? Our "new" adversaries are not tied to large investments in technology, systems, or acquisition processes - We must rapidly adapt! (Justice, 2007)

Just as many in the civilian sector who have transitioned to OSS have discovered, OSS offers many advantages, and one of them is being well suited for agile development. At the 2007 I/ITSEC Flag and General Officer Panel, then Deputy Under Secretary of Defense for Readiness, Paul Mayberry, said,

> Proprietary solutions are short sighted… proprietary is essentially an anathema to those criteria of responsiveness… But the issue is right now, again, with proprietary is how you quickly turn to new, emerging, and changing problem sets. When I can only go back to one source and their sort-of dedicated workforce, it really is not getting at the fundamental criteria that we have for all of training transformation effects, and that is a notion of being agile, adaptive, responsive, timely and trainable." (Williams, 2007)

Much of this impetus for this interest in OSS was created by the publication of the DoD's Open Technology Development Roadmap (Herz, Lucas, and Scott, 2006). This study, created for the Deputy Under Secretary of Defense for Advanced Systems & Concepts, defined three goals to drive adaptation within the DoD:

1. Leverage open source infrastructure and technologies
2. Apply open source collaborative technologies
3. Change the default acquisitions and development behavior to default to technology services vs. products

There is obviously a big push for multiple reasons within the DoD to consider a new alternative to the current method of software procurement. This is not just a minor modification, but rather a complete change to the mindset of those responsible for procuring software systems.

OSS Primer

OSS has an undeserved reputation for being a counter-culture movement, consisting of anarchists, aging hippies or socialist graduate students working at night who believe that all software code should be free. While there are certainly people in the movement who might fit these descriptions, they are a small minority of those working with open source, especially in business. The average OSS developer is thirty years old, with eleven years of programming experience. And those that actually use OSS don't come close to that characterization – 87% of all US businesses use OSS in one form or another. (Vass, 2007) Generally, those who use OSS are people who feel that being open source makes the software provide the best solution for their problems. Two of the most interesting and appealing aspects about OSS is that the cost to learn is essentially zero and the knowledge about how and why the code developed is freely accessible. A recent report out of Europe (Ghosh, 2006) showed that joining an OSS group was one of the best ways for programmers to learn and train around a software language or product.

The definition of open source is rather long, with ten criteria required to be considered open source according to the Open Source Initiative (OSI). However, many of these are minor or legalities. The three major criteria that software must meet to be considered open source are:

1. Free redistribution -the license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source code must be distributed - The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge…

3. Derived works - the license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software. (Coar, 2006)

There are many licenses which meet the definition of open source by the OSI, but by far the most popular is the Gnu Public License (GPL), with over 60% of open source projects licensed under the GPL. The next two most popular licenses, the Lesser Gnu Public License and the Berkeley Software Distribution have between 6 and 7% each, so it's easy to see that the GPL is by far the dominant license. (Freshmeat, 2008) Open source licenses list the requirements of using the software and present the rights and requirements of the parties using the software. David A. Wheeler, a software expert at the Institute of Defense Analysis, argues strongly that anyone creating an open source project should use the GPL (or a compatible license) and not try to invent their own license, as meeting these criteria make it much easier for the new project to be used with other projects. (Wheeler, 2008) The topic of open source licenses is too vast for this paper to give more than a cursory mention; for more information please see either (Wheeler, 2008) or (Rosen, 2005).

However, the one item a potential user must remember when considering using a piece of OSS as part of a larger software project is to ensure its license is acceptable. By doing this, the developer can ensure that the other components of the project are compatible with the open source code – some licenses limit what types of software the open source software can be combined with. Additionally, developers must ensure that the license does not contain any unacceptable requirements. For example, it is common in licenses that all derivative works must be released as open source. While this is not inherently a bad thing (it is what keeps the open source movement strong), the developer should purposefully make that decision rather than be surprised by it after expending significant resources combining proprietary software he doesn't want to release with an open source library whose license requires it.

OSS ADVANTAGES

Cost

Generally, when people not familiar with OSS think of its advantages, the first one to enter their minds is cost, generally considered as "free." While open source licenses allow a person or company to download OSS for free, unless it is a lone technical person downloading an application for personal use, there will be costs associated with using the software. While the initial costs for OSS might be lower (or even free), businesses and program managers are generally more concerned with the total cost of ownership (TCO). This is the cost of acquiring, installing, using, updating, and all the other myriad tasks required to create a capability that requires software. While there have been many studies done comparing TCO for proprietary software to OSS, unsurprisingly the best predictor over which model the study was going was going to find cheaper was who was doing (or had sponsored) the study. Microsoft's studies show convincingly that proprietary was much cheaper, while IBM's or Sun's show equally compelling evidence that OSS is the true bargain.

The problem with TCO studies is that the assumptions used by the researchers greatly affect the results. In the case of an operating system for a network, for example, the amount of time that system administrator will have to spend to perform maintenance, updates, etc. for each of the operating systems being evaluated can radically swing the results of the TCO study. In reality, unless the manager responsible for making the decision can find a TCO report by an impartial observer looking at a situation that closely matches the manager's requirements, the manager's best option is to attempt to estimate the TCO of his/her software options for his/her specific computing need.

In concluding the discussion on cost, the only thing to say is there is no clear and compelling evidence for the proposition that either proprietary or OSS is significantly cheaper. As more OSS enters common usage, the answer to this question will become clearer.

"Free as in freedom, not free as in beer"

However, there are many reasons why OSS is popular besides cost; the old saw "Come for the cost, stay for the quality" certainly applies to OSS. One of the most popular reasons people become advocates of OSS is the freedom that it provides, which manifests itself in many different ways, all of which generally lead to a higher quality of software. People have found that there are many things that they can do using OSS that they could not do using proprietary software, and that these increase the value of OSS.

A good example of this is when a small simulation company, Engineering and Computer Simulations (ECS), created an application that allowed a web browser to open a training simulation on the user's computer and report the results of the user's session. The user, on a browser, was logged in to a learning management system (LMS) which was taking the user through a course of instruction. When the course required the user to user to demonstrate competence in a training simulation, the LMS would launch the simulation. As the user completed objectives in the simulation, this was reported back to the LMS, which kept track of the user's progress. Although ideally, any SCORM conformant LMS should be able to take the feedback as input, in reality it is much more difficult. Many groups have tried to do this using proprietary LMS's without success. However, ECS was able to do this using the open source LMS Moodle project, mainly because the ECS programmers could examine Moodle's source code and determine the data it was expecting. Developers consider being able to make the software do whatever they desire a huge advantage of OSS; a common saying among OSS advocates is "How much is it worth to have software that does *exactly* what you want it to do?" Generally, the costs of having to adapt other programs, or worse, personnel, to the way a unchangeable piece of software operates is huge.

Another good example of the freedom provided by OSS is the ability to have any programmer or company work on the product. This gives several advantages to the user. The first is that the user can never be locked in to a particular vender for work on the application. For example, if the DoD pays a company to build an application using their proprietary software, and later the DoD wants to upgrade the application, the only company which can do the work is that which did the initial work. However, if the application is built using OSS, any company can bid on the work, and multiple bidders means that the DoD is much more likely to get a lower price.

A similar effect is lock-out. While this is not as well understood as lock in, it is another common problem. It occurs when a company no longer supports applications it built previously; this can be due to the company releasing a newer version of its product and deciding to no longer support the old one, or a company gets acquired by a competitor who no longer supports the product, or a company simply goes out of business. When this happens, the product is dead, and there is nothing that anyone using it can do because they do not have access to the code. However, with OSS, anyone wanting to use the product has access to the source code, and can modify it as necessary to meet their needs.

Security

Comparing the number of viruses, worms, Trojan horses, etc. that infects Microsoft Windows and its applications compared to either Linux or Apple OS-X would indicate that open source products are significantly more secure. However, it is not correct to say that OSS is always more secure than closed software: it depends on the domains and the software (either closed or open) that is available. OSS does have one major advantage over closed systems: because the source code is visible to all, security flaws are more apt to be discovered and corrected than if the code cannot be viewed. While this might seem counterintuitive to many, believing that the visibility would lead people to be able to discover and exploit and security flaws, the fact is that in large software projects it is almost impossible for the author to ensure that the code is completely secure. Also once a code flaw or bug is found an individual can make that change immediately versus waiting for a vendor to create a patch. Review of code by the largest number of competent programmers is the only way to find errors, and no matter how good a company reviews their closed software, their review will not have nearly the number of competent personnel reviewing the code. (Saltzer and Schroeder, 1975) said that computer security must not depend upon the ignorance of the attacker, for a determined hacker is not likely to remain ignorant long.

CURRENT GOVERNMENT USE OF OPEN SOURCE SOFTWARE

One of the earliest and most enthusiastic adapters of open source in government has been NASA. Again showing that cost is often times a minor factor in deciding to either use open source, or release code as open source, NASA did this for many reasons besides cost:

- To increase NASA software quality via community peer review
- To accelerate software development via community contributions
- To maximize the awareness and impact of NASA research
- To increase dissemination of NASA software in support of NASA's education mission

NASA lists twenty open source projects on its web site which can provide a wealth of functionality to developers; they include:

- CODE: a software framework for control and observation in distributed environments. The basic functionality of the framework allows a user to observe a distributed set of resources, services, and applications. A user can also use the framework to manage distributed resources, services, and applications. The components that allow measurements to be made and actions to be performed are modular and a small set are provided with the framework. Users can also write their own measurement and action components or use ones that are provide by 3rd parties.

- Growler: a C++-based distributed object and event architecture. It is written in C++, and supports serialization of C++ objects as part of its Remote Method Invocation, Event Channels, and in its Interface Definition Language. Its primary application has been in support of interactive, distributed visualization, computational steering, and concurrent visualization, but it is a general purpose system for distributed programming.

- JavaGenes: a general purpose evolutionary software system written in Java. It implements several versions of the genetic algorithm, simulated annealing, stochastic hill climbing and other search techniques.

- Livingstone2: a reusable artificial intelligence (AI) software system designed to assist spacecraft, life support systems, chemical plants or other complex systems in operating robustly with minimal human supervision, even in the face of hardware failures or unexpected events. Livingstone2 diagnoses the current state of the spacecraft or other system and recommends commands or repair actions that will allow the system to continue operations.

- Vision Workbench: a modular, extensible, cross-platform computer vision software framework written in C++. It was designed to support a variety of space exploration tasks, including automated science and engineering analysis, robot perception, and 2D/3D environment reconstruction, though it can also serve as a general-purpose image processing and machine vision framework in other contexts as well. It leverages the Intelligent Robotics Group's (IRG) extensive experience developing surface reconstruction and tools for planetary exploration---e.g. the Mars Pathfinder and Mars Exploration Rover missions---and rover autonomy.

- World Wind: an application which allows any user to zoom from satellite altitude into any place on Earth, leveraging high resolution LandSat imagery and SRTM elevation data to experience Earth in visually rich 3D, just as if they were really there. Particular focus was put into the ease of usability so people of all ages can enjoy World Wind. All one needs to control World Wind is a two button mouse. Additional guides and features can be accessed though a simplified menu. Navigation is automated with single clicks of a mouse as well as the ability to type in any location and automatically zoom into it. One of the features of World Wind is a set of visually intense animations that demonstrate a variety of subjects such as hurricane dynamics and seasonal changes across the globe as shown in Figure 1. (NASA)

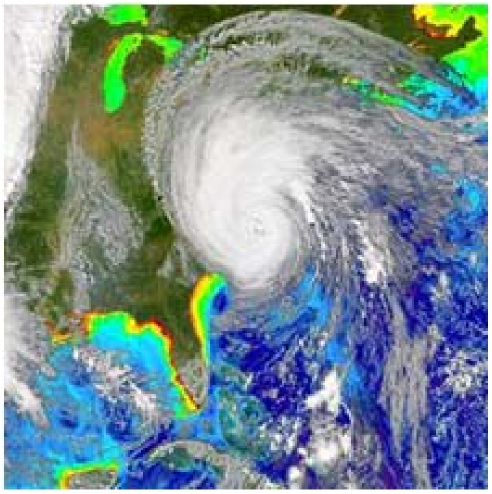

Figure 1 – A World Wind application by the Goddard Space Flight Center demonstrating weather dynamics

Another open source software that the government is using is the Globus Toolkit. The Globus Alliance, of which the Department of Energy's Argonne National Laboratory at the University of Chicago is an active member is developing the toolkit.

The Globus Toolkit is designed to make it easier to create a computing grid, which can then be used to solve problems. A computing grid lets people share computing power, databases, instruments, and other on-line tools securely across corporate, institutional, and geographic boundaries without sacrificing local autonomy. An example of what can be done using a grid of this type is Figure 2, which is a representation of the gravity wave resulting from a collision of black holes. To produce the data used to produce this image, multiple supercomputers were connected using the Globus Toolkit. This research was awarded a 2001Gordon Bell prize in high-performance computing in a special category emphasizing high-quality algorithms and software libraries. (Globus)

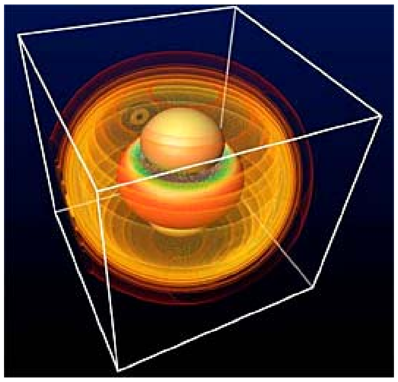

Figure 2 – Graphical Representation of black hole collision rendered by computers connected via the Globus Toolkit (Globus)

OSS has been widely used in the geospatial field as well. Military offices such as the Naval Research Lab, National Geospatial Agency and National Reconnaissance Office have partly funded the OSSIM (Open Source Software for Imaging and Mapping) geospatial tool set.

OSSIM is a high performance software system for remote sensing, image processing, geographical information systems and photogrammetry. It has been under active development since 1996. OSSIM has been funded by several US government agencies in the intelligence and defense community and the technology is currently deployed in research and operational sites. Designed as a series of high performance software libraries it is written in C++ employing the latest techniques in object oriented software design. A number of command line utilities, GUI tools and applications, and integrated systems have been implemented with the baseline. Many of those tools and applications are included with the software releases. (OSSIM)

One exciting development in the same field is that Ball Aerospace recently released its multiplatform application for processing & viewing remote sensing data (called Opticks) for use as OSS as well. Bell came to the conclusion that it was a better business model to release as open source a project that they had developed for the government than to try and add proprietary barriers around it for others to use. They decided that making Opticks open source would greatly increase the number of development teams using it, and by being an expert on Opticks, Bell would be in a great position to capitalize upon the additional use. (Saunders, 2007)

The government has two primary methods to use open source software. The first covers most of the projects discussed so far, which is either releasing software built in-house as open source or working with others to build OSS. The other is not developing the OSS or using existing OSS to develop other software applications, but rather to use existing OSS just as it comes. This is similar to the way that the vast majority of users employ software, just as people use MS Word or the Firefox browser.

One of the most successful government implementation of a system based upon OSS not created within the government is the Department of Veteran's Affairs Veterans Health Information Systems and Technology Architecture (VISTA). VISTA is a health care management system that allows health care workers to input and reference a patient's entire medical record. VISTA has adapted by both public and private health care organizations.

Another OSS that the government uses is Pentaho, business intelligence software. Pentaho's Open BI Suite provides the user with all the features of a proprietary BI product and can interact with many third party applications. It is one of the most popular BI software, and it is currently in use the Navy, Air Force, Food and Drug Administration, Department of the Interior and NAVAIR Systems Command.

Additionally, Northrop Grumman is using OSS as the basis for their Future Ship Architecture. They made the decision because of the unrestricted access to the code, allowing them to build loosely coupled modules. Additionally, these are vendor-independent and follow standards. This will allow them to quickly and easily produce the software to drive the Navy of the future.

CURRENT OPEN SOURCE USAGE IN M & S

Although currently very few government modeling and simulation projects are built entirely, or even largely, using OS technology, there are several OS projects that the DoD has either created or used for multiple projects.

Delta3D

Delta3D is an open source game and simulation created at the MOVES Institute at the Naval Postgraduate School in Monterey, California. Delta3D was created when researchers there became frustrated after a successful project could not be distributed throughout the Marine Corps due to the high cost of the run times licenses of the underlying proprietary software. Rudy Darken, as Associate Professor at MOVES, and Erik Johnson, who would become the technical lead of Delta3D, began searching for the technology to build visual simulations which would not require per-seat run time licenses. Darken and Johnson viewed per-seat licenses as a remnant of the time when doing visual simulations required machines costing $50,000 and up, meaning that these simulations were done only on a small number of computers. Because the advent of the graphics card and cheap, high capability PC's had meant it was possible to run visual simulations on computers costing as little as $2,000, it was now practical and desirable to run simulations on as many computers as possible, and per-seat licenses did not scale well to this new paradigm.

The goal was not to find an open source solution, but merely one which did not require per-seat licenses. As Johnson searched for a product which he and the MOVES students could build visual simulations atop, he realized that there were many high quality open source projects which performed one of the myriad of tasks that the software underlying a visual simulation is required to perform. For example, OpenSceneGraph provided excellent visual rendering; OpenDynamicsEngine provided rigid body physics; OpenAL provided audio. Johnson also discovered that many groups had built products by combining two or more of these libraries to get the functionality they needed.

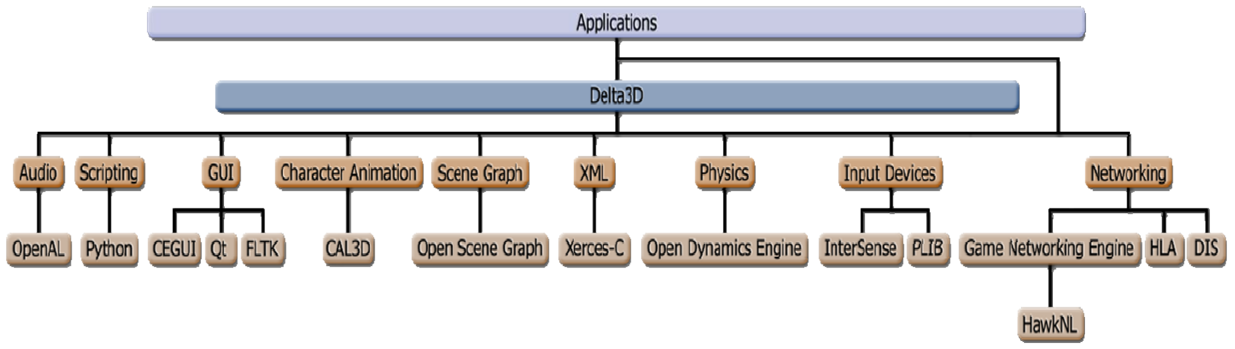

Figure 3. Delta3D Architecture

He realized that if these many libraries were combined under one application programming interface (API), that the resulting game engine would be quite powerful. Moreover, by reusing the existing libraries, the engine could be built without having to build everything from scratch. Darken and Johnson decided to create a game engine using these libraries as the basis. The architecture for the latest version of Delta3D is shown in Figure 3. A scene from a Delta3D application is show in Figure 4.

Figure 4. Screenshot from Delta3D application for JIEDDO

OpenEaagles

Open Extensible Architecture for the Analysis and Generation of Linked Simulations. (OpenEaagles) is a multi-platform, open source, simulation framework designed to produce virtual or constructive simulations rapidly. It was originally created by the Air Force and then later released as an open source project to gain some of the benefits of being open source. It has been used in a wide number of applications, including training, human factor studies, and stand-alone and distributed constructive applications to analyze systems. It is a mature software framework as it has been in active development for over a decade.

OpenEaagles is not a simulation, but rather serves as a “design pattern” for simulations, providing a structure for building simulations. Figure 5 shows the OpenEaagles framework, while figure 6 shows the package hierarchy in an OpenEaagles application. By providing abstract representations of system components which normally communicating using the Distributed Interactive Simulation (DIS), models of different levels of fidelity can be easily intermixed and selected for optimal runtime performance within the same simulation.. For example, in a simulation where a precise representation of enemy artificial intelligence (AI) is required, exceptionally realistic AI software, which might be slow, can be used. If that simulation does not run quickly enough and the accuracy of physics calculations is not critical to the application’s success, a quicker but less precise physics model can be used. (OpenEaagles).

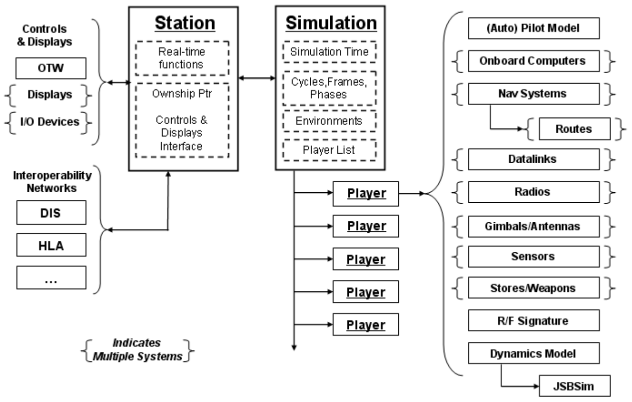

Figure 5. OpenEaagles Framework (from OpenEaagles)

SubrScene

Like OpenEaagles, SubrScene was created by the Air Force for internal use and later released as open source. SubrScene is designed to aid in the visualization of first person immersive simulations. Originally built as an image generation application, later the rendering component was broken out into its own framework which embodies the entire core set of image generation features. It is especially useful for simulations consisting of multiple viewpoints, such as a CAVE or multi-display simulator.

SubrScene is designed to abstract the layer of graphics from the user application, thus allowing the developer to not worry about the rendering of his/her application. It is designed to be simple to use, with the user just linking to framework library. Additionally, if the user desires, the framework can spawn a graphics context and window, further reducing the display requirements of the developer. The framework basic functionality support includes player placement, articulation, special effects, data base intersection, shader management, dynamic viewing channel control, environment control, and masking functions.

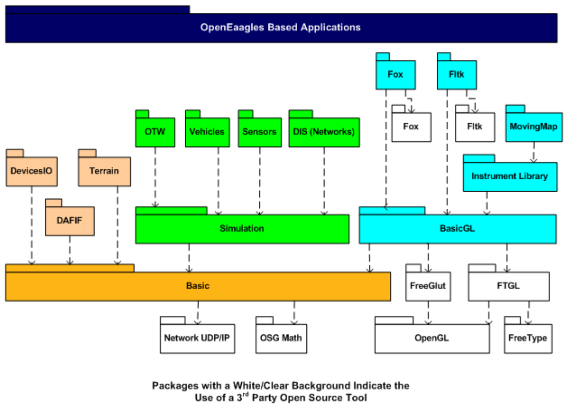

Figure 6. OpenEaagles Package Hierarchy (from OpenEaagles)

Once complete, the SubrScene IGS consists of three run time applications. One is used for network execution for the IGS, another is the remote rendering interface application that is launched by the network manager, and the third controls the entire application. This is shown in figure 7. (SubrScene)

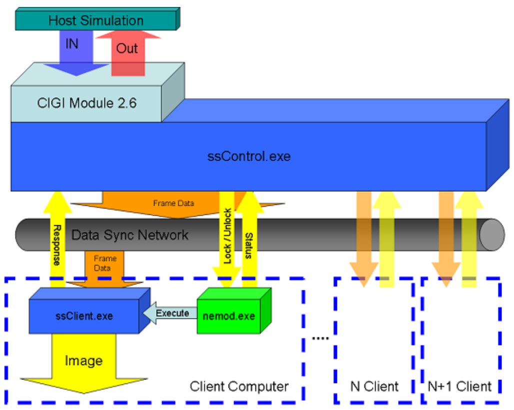

Figure 7. SubrScene Executable Framework (from SubrScene)

BRL-CAD

Another important government tool that was developed within the government and then released as open source is BRL-CAD. This is a powerful computer-aided design (CAD) tool that was begun at the Army Ballistic Research Laboratory (now the Army Research Laboratory) in 1979 to assist with simulation and engineering analysis of combat vehicle systems and environments. It has been used at over 2000 different sites throughout the world and now consists of over one million lines of code in the C programming language.

BRL-CAD is a cross-platform solid modeling system that includes interactive geometry editing, high-performance ray-tracing for rendering and geometric analysis, image and signal-processing tools, a system performance analysis benchmark suite, and libraries for robust geometric representation. Its models are not intended to be used in real-time rendering applications, such as games, but rather in more in-depth engineering simulations.

BRL-CAD uses constructive solid geometry (CSG) to build and analyze realistic models of objects by modifying just a few primitive shapes. Even though this might lead some to believe that the resulting models will be cartoonish or poor in quality, this is not the case – Figure 8 shows one of the highly detailed models created using BRL-CAD. By using CSG, as compared to more traditional modeling methods where just the outer surfaces are modeled, BRL-CAD has the capability to build objects with real-world materials, densities, and thicknesses so that analysts can study physical phenomena such as ballistic penetration and thermal, radiative, neutron, and other types of transport. Figure 9 shows where

BRL-CAD can be used in the engineering process. (BRL-CAD)

Figure 8. Bradley fighting vehicle created using BRL-CAD. (from BRL-CAD)

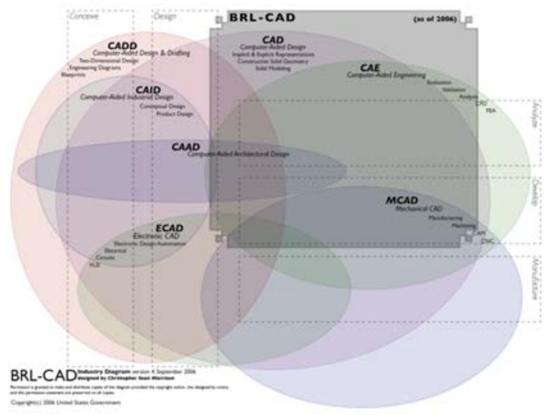
Conceive
Design
BRL-CAD
(as of 2006)
CAD
Computer-Aided Design
Implicit & Explicit Representations
Constructive Solid Geometry
Solid Modeling
CADD
Computer-Aided Design & Drafting
Two-Dimensional Design
Engineering Diagrams
Blueprints
CAE
Computer-Aided Engineering
Evaluation
Validation
Analysis
CFD
FEA
CAID
Computer-Aided Industrial Design
Conceptual Design
Product Design
CAAD
Computer-Aided Architectural Design
Analyze
Develop
MCAD
Mechanical CAD
Manufacturing
Machining
CAM
CNC
ECAD
Electronic CAD
Electronic Design Automation
Electrical
Circuits
VLSI
Manufacture
BRL-CAD Industry Diagram version 4, September 2006
designed by Christopher Sean Morrison
Copyright (c) 2006 United States Government

Figure 9. BRL-CAD's functionality (from BRL-CAD)

CONCLUSION

The government has been using OSS in a wide variety of applications for quite a while, and has seen a myriad of benefits from doing so. However, despite the fact that the government has been using open source for some time, there is still much work to be done. Moving the DoD's acquisitions infrastructure to use OSS whenever it makes sense will not be an easy task. For one thing, there are many organizations, which are locked in to a way of doing business or worse yet tied to one specific software vendor. The best that should be hoped for is that acquisitions project managers and companies should be required to at least review what OSS available that could meet a desired capability.

Perhaps the best advice on changing the mentality of those responsible for procuring software for the DoD comes from David Carney, the chairman of the Software Engineering Institute at Carnegie Mellon University. Carney was writing not about the movement to OSS, but another change in software procurement that the DoD went through in the 1990's: moving from specialty military computer equipment to commercial off the shelf (COTS) equipment. Just as with OSS, the movement started slowly, with great doubts and uncertainties. But eventually, it picked up momentum, and has saved the DoD huge amounts of funds. Carney writes in the forward:

> "The frustration I noted stems from my sense of two things. First, I perceive that many of us are paying phenomenal lip service to the new directives and mandates, to the things that must be accomplished to meet these changed circumstances. 'We've all got to start doing business differently' we are saying, loud and clear. And second, most of us—upper-level policy managers to low-level analysts, civilian contractors and DoD colonels—are confidently doing just what we've always done, thinking that somehow the need for change applies to everyone except us, and waiting for the guy in the next office to mend his ways...
>
> And to repeat the initial point: each individual in the community has to change his or her habits, sometimes in deep and profound ways. The experience that comes from an entire career may now be a doubtful, or even an untrue guide for the new acquisition paradigm. This is a painful prospect to all of us. But it is the reality that we face. The quicker we face it, the better off we'll all be." (Carney, 1998)

REFERENCES

Bower, J. L. and Christensen, C. M. (1995). Disruptive Technologies: Catching the Wave, *Harvard Business Review*, Jan.-Feb., pp. 43-53. doi: 10.1225/3510.

BRL-CAD, (N.D). Retrieved July 2, 2008 from http://brlcad.org.

Buxbaum, P. (2008) Navy to focus only on open systems, *Federal Computer Weekly*, Published on March 6, 2008. Retrieved March 10, 2008 from http://www.fcw.com/online/news/151858-1.html.

Carney, D. J. (1998). *Quotations from chairman David (A little red book of truths to enlighten and guide on the long march toward the COTS revolution).* Pittsburg, Pa. Carnegie Mellon University.

Coar, K. (2006). The open source definition. Retrieved June 23, 2008 from http://www.opensource.org/docs/osd.

Freshmeat (2008). License breakdown. Retrieved July 2, 2008 from http://freshmeat.net/stats/#license.

Ghosh, R. A. (2006). *Study on the: Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU.* Maastricht, The Netherlands : United Nations University-MERIT.

Globus Alliance (N.D.) Retrieved July 2, 2008 from http://www.globus.org/.

Hardy, M. (2005, April 4). Open wide: Linux appeal grows as applications flourish, *Federal Computer Weekly*, Retrieved June 23, 2008 from http://www.fcw.com/print/11_13/news/88465-1.html.

Heiman, R. V., and Picardi, A.C. (2006). Worldwide software 2006–2010 forecast summary, Worcester, Massachusetts: IDC Group.

Herz, J. C., Lucas, M. and Scott, J. (2006). Open Technology Development. Washington D.C.: U.S. Government Printing Office.

Justice, N. (2007). “Deploying Open Technologies and Architectures within Military Systems.” Third DoD Open Conference, Association for Enterprise Integration, Vienna, Va., December 11 2007.

NASA (N.D.). Retrieved July 2, 2008 from http://opensource.arc.nasa.gov/.

OpenEaagles web site (N.D.). Retrieved June 23, 2008 from http://www.openeaagles.org/.

OSSIM, (N.D.). Retrieved July 2, 2008 from http://www.ossim.org/

Rosen, L. (2005). *Open source licensing: Software freedom and intellectual property law*. Upper Saddle River, New Jersey : Prentice Hall Professional Technical reference.

Saltzer, J. and Schroeder, M. (1975). The protection of information in computer systems,” *Proceedings of the IEE*, v63 n9.

Saunders, V. M. (2007). “Opticks open source project: Taking a government developed application to the open source community.” Third DoD Open Conference, Association for Enterprise Integration, Vienna, Va., December 12 2007.

SubrScene Introduction Page (N.D.). Retrieved June 23, 2008 from http://www.subrscene.org/intro.html.

Vass, B. (2007). “Migrating to open source: have no fear.” Third DoD Open Conference, Association for Enterprise Integration, Vienna, Va, December 11 2007.

Wheeler, D. A. (2008). Make your open source software GPL-compatible. Or else. Updated May 31, 2008, retrieved July 2, 2008 from http://www.dwheeler.com/essays/gpl-compatible.html.

Williams, J. S. (Ed.) (2007, November 28). Mayberry calls for further integration. I/ITSEC Show Daily, p. 4 Retrieved from http://iitsec.org/documents/Day3_2007.pdf