GL Studio Brings Realism to Aircraft Cockpit Simulator Displays

Kim Stults
580th Software Maintenance Squadron

Testing operational flight programs (OFPs) for aircraft requires that the user be able to enter data and generate output via a graphical user interface (GUI). Desiring to enhance the realism for the test team and upgrade the outdated software, we began the search for a new tool. A new commercial off-the-shelf (COTS) product was selected and a success story unfolded. This article presents that story.

In 1995, after a decade of development, the Special Operation Forces (SOF) Extendable Integration Support Environment (EISE) was established at Robins Air Force Base in Warner Robins, Georgia. The purpose of the SOF EISE was to provide hardware and software support for seven selected avionics systems (see Figure 1). The support environment permits the modification and test of the OFPs running on the aircraft.

A key component of the SOF EISE is the crew interface (CIF) simulation. It is designed to allow real-time, simultaneous operations with the line replaceable unit and environment simulation portions of SOF EISE. The computer display touchscreens provide a simulation of the actual aircraft cockpits. The CIF stands as the key control element between the system user and both the real and simulated aircraft avionics. It provides a means by which the functional capability of simulated aircraft avionics, real aircraft avionics, and their associated OFPs can be evaluated against the required capability for these system components.

Problem

After almost 20 years, the architecture of the CIF segment was becoming unsupportable. A unique CIF simulation executed on a single-board computer (SBC) for each of the seven systems. The SBC code communicated with two silicon graphic computers over a network using a remote procedure call protocols (RPC). Three processes ran on one of the silicon graphics (client, server, and GUI), and two processes ran on the second (client and GUI). These processes communicated using COTS software library calls. The hardware was becoming unsupportable and the software solutions were outdated. The segment was incredibly complex and resulted in reliability problems as well as high maintenance costs.

Figure 1: *SOF EISE Supported Aircraft*

SOF Aircraft	Deployment Date
MH-53J PAVE LOW III (PL3)	1995
AC-130H Gunship (GS)	1997
MC-130H Combat Talon II (CT2)	1998
MC-130E Combat Talon I (CT1)	1999
MH-53M Pave LOW IV (PL4)	1999
EC-130H Compass Call (CC)	2003
HH-60G PAVE HAWK (PH)	2006

Potential Challenges

Even though a COTS product had been in place, high maintenance cost, poor customer support, and an unreliable development tool led to an investigation of new possibilities for today's technology for both hardware and software. Our requirements fell into basically two categories:

1. **Functional requirements.** These consisted primarily of requirements elicitation from our test group customer. The test group had written extensive procedures over the years that referred to specific graphical objects on the interface. The steps included *observe* actions that specified color, position, button presses, and other very specific details of the legacy graphics. This was a justifiable constraint given the level of effort that would be required to change thousands of pages of test procedures.
2. **Non-functional requirements.** These are essential to retarget the CIF GUI that resided on a silicon graphics machine to a Linux based system, better performance, and higher reliability.

COTS Considerations

After it was decided that the functional capabilities of the legacy system would determine the requirements for the upgrade, a development team met to discuss the vision for the upgrade. Discussion focused on previous problems with vendors, ease of debugging, and the future of the system. From this discussion, it was determined that we were looking for the following [1]:

- **A fully interactive 2-D or 3-D open graphic library-based development tool.** We needed a tool that would allow us to create custom widgets that looked exactly like the legacy widgets. Only tools with the ability to create 2-D objects could do that. We also wanted the capability to improve the appearance of those objects that did not impact test procedures. In the future, we will need to support new platforms. For those new platforms, we will not be restricted to legacy appearances. For these, we would like to create more photo-realistic panels. There is a growing desire for out-the-window views by the users. Should that ever become a requirement, we want to be able to meet it without having to change tools. A 3-D tool is required for that.
- **Non-proprietary, human-readable, object-oriented code.** Historically, there have been issues with the legacy tool that required us to examine the interim files that were generated. Because the interim files were written in a proprietary format, we were unable to analyze certain conditions while debugging. This significantly hampered our ability to quickly correct some defects, and we were not anxious to subject ourselves to that limitation again.
- **Lower development cost.** The legacy tool was old and its customer base was decreasing. Licensing fees were increasing. Expertise with the tool was limited. All these factors forced the cost of the development tool to rise. Newer, cheaper, and more capable tools existed. We would take a cursory look at a few of them and determine which of those deserve further evaluation.
- **Efficient object-oriented designs and code generation.** An object oriented approach to programming is accepted industry-wide. While it is not a function of the tool to provide the

approach, some tools make it easier than others. We wanted a tool that would support, even enforce, object oriented code.

- **A compact runtime library.** Using our legacy tool, the libraries had become nearly unmanageable due to increasing size. While the increases were mandated by increased capability, they were still consuming disk space.
- **Flexible licensing options.** As the number of supported platforms grows, we needed to be able to adjust our licensing agreements. We needed to be able to establish a fixed number of development licenses and a different number of runtime licenses. We needed the option of a site license, in the event the requirements dictated. That is, in the event the number of platforms we were required to support grew to the point that it was economically feasible to get a site license instead of individual run-time license.
- **Proven COTS product with some demonstrated level of maturity.** There are a plethora of tools on the market that meet our needs. Many of them are excellent, some are only good. We had neither the time, nor the engineering resources to evaluate all, or even most of them in depth. We elected to rely on the tool's customer base to do that for us. Mature tools have a large customer base. We wanted an alternative pool for advice, customers who were already using the tool.
- **COTS vendor with good technical support.** Every new tool comes with the promise of technical support. We wanted to be sure that we were getting more than promises. The vendor we eventually selected provided excellent technical support during the evaluation period. We described several unique problems and they promptly provided explanations or solutions.

Certainly there are a lot of graphics products on the market. We looked at several and had vendors come in to demonstrate their products. We obtained references from those companies and contacted their customers. One can usually learn more from a vendor's customer than a vendor representative. For example, you can find out what to expect in the company's training courses, the quality of the technical support, the tool's ease of use, etc. After making an initial company selection for prototyping, we arranged for a week-long training session at a facility near our site. The instructor was technically competent and a veteran in front of the classroom. He provided satisfactory answers and examples for all of our questions. Most of the questions were drawn from what we thought would be difficulties with using his company's tool. In every case, we were more than satisfied with his response.

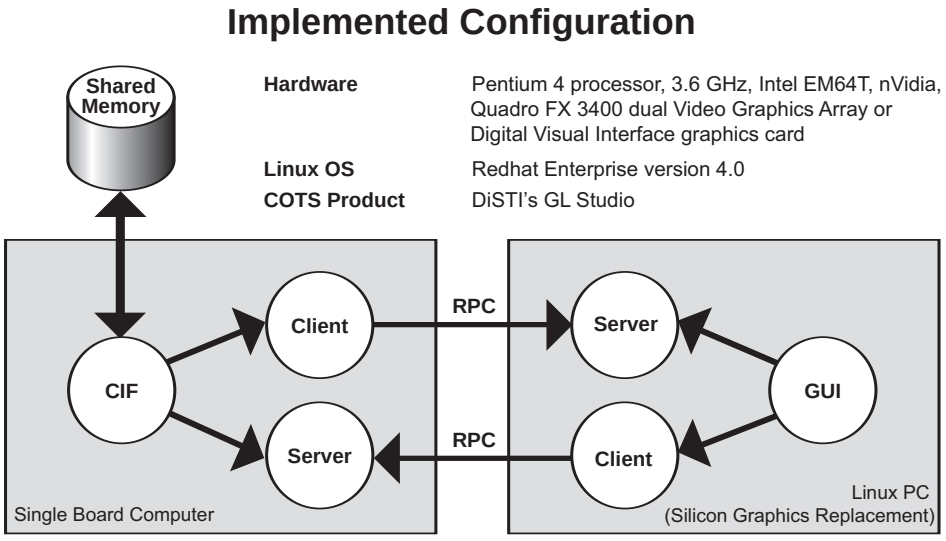

Figure 2: *Selected Architecture for Configuration*

Obstacles

Change is not always welcome. There were concerns in moving from the familiar to the unfamiliar. We had previously faced difficulties with customer support, so a good working relationship with the vendor was crucial. There were concerns from the test team that the look and feel of the test station not be changed for fear of impact to the current test procedures.

The other obstacles we encountered were typically hardware-related or self imposed. The original Linux personal computer we were given for development had an incompatible operating system (OS) and graphics card for the tool we were evaluating. Our first step was to acquire a compatible OS. At this point though, the hardware acquisition still lagged behind the software we were developing. The integrated product would be driving two monitors. We did our early development on systems that only had one. We were concerned that mouse clicks on the second monitor would not return the screen coordinates we were expecting. We tasked one of our developers to examine the possibility of intercepting the universal serial bus event stream and modifying it. This was a risk mitigation step. Our next step was to choose a multi-headed graphics card to work with our selected OS. Finally, we had to find a touch screen that had existing drivers for our chosen Linux OS. Once all of our hardware was in place, we knew that the (x,y) desired coordinates were being returned from mouse events and the dual screen implementation was a non-issue. The vendor had assured us that this would not be a problem, but we had no way to prove it. Therefore, a lot of time had been spent on risk mitigation that was completely unnecessary.

The Solution

The ultimate architecture for our solution was a combination of procedural programming in the system simulation and object-oriented programming in the GUI. Figure 2 depicts the selected architecture for our solution. DiSTI GL Studio was a perfect fit for our COTS product requirements. GL Studio is a premier Human Machine Interface (HMI) development toolkit that allows for the creation of end-to-end safety critical displays from prototype to delivery. GL Studio has no proprietary formats and flexible licensing options, and it produces reliable, safe, efficient, reusable applications in a rapid and easy fashion. The GL Studio development package was a fraction of the cost of our previous COTS product. The licensing fees alone will save approximately $120,000 over the next five years. The PCs will save an additional $42,000 over the same time period.

Keys to Successful Integration

Our COTS integration was completed ahead of schedule and within budget. The integration effort was a success story for many reasons.

Partnering With the Vendor

We received and continue to receive excellent technical support from DiSTI. Some vendors have a take-it-or-leave-it approach. We were a team and they wanted us to succeed. We were not the only

ones working weekends when we had a problem to solve. Particularly, they worked one weekend to develop a drop-in keypad class for us. In the beginning, we had issues with the particular OS we had chosen. It was not supported by GL Studio, but they helped us get it working anyway. They also had many customers running multi-headed touch screen applications in Windows, but we were the first to attempt it with Linux. Thanks to DiSTI's support, we were able to implement our desired design.

Networking With Similar Users

DiSTI networked us together with the C-130 Self-Contained Navigation System (SCNS). ARINC had previously designed photo-realistic pilot panels for the SCNS project. We were able to obtain the reusable software objects for several pilot instruments and implement them into our architecture seamlessly. This saved a tremendous amount of time and money. For our initial delivery, we did not have time in the schedule to photograph our own instruments and code the displays. Having a product that was already being used with displays that we needed was a life saver.

Selecting the Hardware

Implementing the design of one multi-head PC with two monitors – on each of the five platforms – reduced the 10 required host computers to five. This single PC architecture, along with utilizing a shared memory segment to replace the outdated communications library, greatly simplified our code – eliminating approximately 25,000 lines of code.

Assessment of Results

The overall results for our upgrade have been outstanding. At this time, all of the platforms have been successfully ported to the new architecture and have been in use for approximately six months. Not only has the integration been a success for the development team, but for the end users as well.

Customer Satisfaction

Even though our customer, OFP developers, and testers were leery of a change in the beginning, they have been very pleased with the improvements in our architecture, the most notable improvement being the photo-realistic displays for the pilot instruments (Figure 3, a and b). This, of course, enhances the realism of flight simulation and test.

> *"Use of COTS products is a viable way to upgrade existing systems."*

Sharing Across Multiple Platforms

A huge success of the EISE in general is the ability to share software across multiple platforms. The object-oriented design approach saved us a substantial amount of time by providing the capability to reuse components instead of redeveloping individual instruments or instruments parts. The new COTS software also supports multiple OSs, leaving the future possibility for further upgrades and configuration changes completely open and easy to maintain.

Conclusion

Use of COTS products is a viable way to upgrade existing systems. As part of our development plan, the port was completed in incremental steps. In order to minimize risk, our first step was to port only the code that ran on the silicon graphics machine and to maintain the original interfaces between the SBC code and the silicon graphics. Once the new graphics were in place, the second phase was to restructure the SBC code and make minimal interface changes. This approach worked very well for our initial delivery.

Now that we have transitioned to the new architecture, we have many options available to pursue. The most important option is the addition of more photo-realistic displays as time and schedule permits. The Linux solution was a good fit with the planned migration to a real-time Linux architecture in the lab. In general, the COTS HMI development tool was able to help us save a substantial amount of time and budget, allowing us to complete our objective well within our required deadline. The new architecture of the instrumentation lends itself for use in many other applications for the future as well, further saving time and budget for the SOF EISE lab as well as any other entity that may receive this source as government-furnished information for other similar programs.◆

Figure 3 a and b: *CT1 Pilot Display*

References

1. "Generating Human Machine Interfaces." Orlando: DiSTI, 2006.

About the Author

Kim Stults is a member of the technical team for the 580th Software Maintenance Squadron (580 SMXS) of the 402d Software Maintenance Group at Warner Robins Air Logistics Center, Robins Air Force Base, GA. The 580 SMXS performs software maintenance changes for the SOF weapon systems. She is responsible for the crew interface segment of the EISE. Stults has 17 years experience in software engineering, 11 in private industry, and six with the Air Force. She has bachelor's degrees in mathematics and computer science.

420 Richard Ray BLVD
STE 100
Robins AFB, GA 31098
Phone: (478) 926-0718
Fax: (478) 926-0226
E-mail: kimberly.stults
@robins.af.mil