CLOSE COMBAT TACTICAL TRAINER SAF ON A PC

Bob Burch, Peggy Hughley, Gene McCulley, Derrick Dietrich
Science Applications International Corporation
Orlando, Florida

Abstract

The current workstation designs used by SAFs, such as CCTT SAF and ModSAF, date back to a SIMNET legacy of the late 1980s. CCTT and other large simulation programs within Department of Defense are becoming the exception rather than the rule. The SAF Suite and Standalone configurations of CCTT SAF were created in part to provide a more cost-effective platform by combining application and workstation roles that normally would require no less than three host computers. The next logical step is to port SAF Suite and Standalone to the even more cost-effective Intel-based PC platform.

The porting of SAF Suite and Standalone to the PC requires a change in host computer platform, operating system (OS), compiler, and supporting software libraries. The current configuration is the IBM PowerPC/AIX OS UNIX workstation with the Powerada compiler from OC Systems, Inc. Powerada includes AdaMotif, a commercial Motif binding for Ada. The Intel-based PC configuration is a Pentium III/Linux OS PC workstation with the GNU Ada compiler, GNAT, and other public supporting software libraries.

Author Biographies

Bob Burch is a Chief Scientist at SAIC Orlando. Mr. Burch has over eighteen years of experience in Modeling and Simulation encompassing crew cabin simulations and Computer Generated Forces. Mr. Burch was the Chief Scientist for design on the Close Combat Tactical Trainer (CCTT) Semi-Automated Forces (SAF) system. At SAIC, Mr. Burch is a Chief Scientist in the Advanced Distributed Simulation Research Team and supports research in object-oriented software architecture and Computer Generated Forces. Mr. Burch received his Bachelors of Science in Computer Science from the University of Central Florida.

Peggy Hughley is a Software Engineer for SAIC. Ms. Hughley has worked for five years designing and developing Ada applications using Object-Oriented Methodologies. Ms. Hughley's experience is in real-time simulation systems. Her work includes development and testing of code in the Behaviors, Framework, and UCI (User Computer Interface) CSCs (Computer Software Components) of CCTT SAF. Ms. Hughley has also worked on the digitization of CCTT with the integration of FBCB2. Ms. Hughley received a Bachelors of Science in Computer Science with a minor in Mathematics from the University of Central Florida.

Gene McCulley is a Senior Software Engineer with SAIC Orlando. Mr. McCulley has over seven years of experience with CGF systems including UCF IST CGF, CCTT SAF, ModSAF, and the OneSAF Testbed (OTB). Mr. McCulley is a Debian GNU/Linux maintainer and previously maintained the Linux port of the GNAT Ada compiler. His current interests include software architecture and cellular automata.

Derrick Dietrich is a Senior Scientist at SAIC Orlando. Mr. Dietrich has seven years experience in Simulation and Training Systems. Mr. Dietrich was a primary developer of the Semi-Automated Forces (SAF) Workstation for the CCTT and UKCATT programs. Mr. Dietrich received his Bachelors of Science in Computer Engineering from the University of South Florida.

CLOSE COMBAT TACTICAL TRAINER SAF ON A PC

Bob Burch, Peggy Hughley, Gene McCulley, Derrick Dietrich
Science Applications International Corporation
Orlando, Florida

INTRODUCTION

The Close Combat Tactical Trainer (CCTT) is a large scale Distributed Interactive Simulation (DIS) training system (described in [1]). The CCTT Semi-Automated Forces (SAF) system provides doctrinally correct opposing and flanking forces in the CCTT virtual environment. CCTT SAF was developed as a sub-system of CCTT and as such is compliant with the system view promulgated by CCTT. One of the CCTT system decisions was the selection of IBM's AIX operating system on a Motorola PowerPC based platform. AIX is IBM's version of a Unix-flavored operating system. CCTT's reliance on AIX has proven to limit the portability of CCTT SAF code since AIX is present on a limited number of machines that are not popular mainstream devices. In addition, the cost of entry for both AIX and AIX-based machines is greater than other popular mainstream operating systems and hardware. This means that a typical site that might desire CCTT SAF does not have the required systems and the cost of obtaining these system is often prohibitive. In addition, the utility of these systems when not being used for CCTT SAF is limited. This paper documents efforts to evolve the CCTT SAF baseline to more mainstream hardware platforms and operating systems.

OBJECTIVES

The objectives of this effort are to evolve the CCTT SAF baseline in such a manner that allows it to co-exist on a variety of platforms and operating systems. The following primary objectives were identified:

- Reduce cost of entry for small installations of CCTT SAF.
- Expand the list of supported platforms to include mainstream hardware and operating systems.
- Maintain the integrity of the legacy investments in data, coursework, hardware, and support structures.

For this effort, the desire was to fully duplicate the functionality of the existing legacy platform while obtaining these objectives. During our efforts, many choices could have been made that would make the porting effort easier but would compromise one of these objectives. In each case, we strived to identify a solution that balanced these objectives while enabling the greater objective of portability of CCTT SAF. The optimal solution would allow for the evolution of the baseline to support other platforms while maintaining a backward compatibility with the legacy system. In this manner, a portable version of CCTT SAF could be slowly instituted into the existing baseline and minimally perturb the baseline support that now exists.

CCTT SAF

It is useful to understand the legacy CCTT SAF configuration and its evolution in order to understand the motivations and issues of porting CCTT SAF to Linux. CCTT SAF is a sub-system in the CCTT system. CCTT SAF is a layered architecture consisting of about 850,000 physical lines of Ada 83 code (later converted to be compliable under Ada 95). The CCTT system software architecture relies heavily on reused code with many layers or functions shared by many of the CCTT applications (of which CCTT SAF is one of about nine distinct applications, see [1] for more detail).

CCTT SAF Application Architecture

To fully understand the scope of the Linux porting effort, one must also understand the application architecture. CCTT SAF actually consists of multiple distinct applications designed to run on separate computer nodes on the network. The CCTT SAF application architecture is client server. The servers for SAF are the Computer Generated Forces (CGF) application. The clients are the SAF Workstations (SAF WS) applications. The CGFs are responsible for the simulation of the virtual world and "serves" the models of vehicles, organizations, and environment. The SAF WSs are the operator interface mechanisms that control and view the status of the simulation on the CGFs. In addition, the DIS Visual Interface (DVI) tool and the Environment Manager applications manage the exercise assets and the environment simulation respectively.

Each CCTT SAF application is a collection of Unix processes that communicate across shared memory. There is a set of low-level processes that exists on all nodes in the distribution and serve as the "system services" layer of the system architecture. Specifically, these common processes include the network interface process, entity database process, and priority manager process. The common network interface provides the ability to distribute the applications across the network and communicate DIS PDUs (Protocol Data Unit). The common entity database maintains a database of all physical world entities for each application. The priority manager helps manage the real-time priorities of the applications.

The size of a CCTT SAF system is controlled by how many SAF WSs and CGFs are connected together. CCTT SAF was designed to allow the user to decide how many workstations and CGFs would be allocated to an exercise. For example, the training site has 15 CGFs and 10 workstations. It can support 5 exercises running concurrently in different virtual worlds. Because of this use case, the SAF applications allow for any number of the "farm" of CGFs and workstations to be allocated to an exercise. Since each CGF simulates about 100 entities (per the current baseline, however this number will soon approach 300), a 400 entity exercise would require 4 CGFs (realizing that tank platoon is nominally 4 entities). In addition, the workstations are designed to control up to 120 entities without overload and can control any number up to the operator's limits.

Motivations

Any porting effort requires a use case that supports the effort of porting. Originally, CCTT SAF was unable to be executed independently and thus would not support a use case that would justify Linux. However, CCTT SAF has evolved over time to include support for use cases that are not well known. This section discusses the evolution of CCTT SAF which ultimately provides the motivation and justification for porting to Linux.

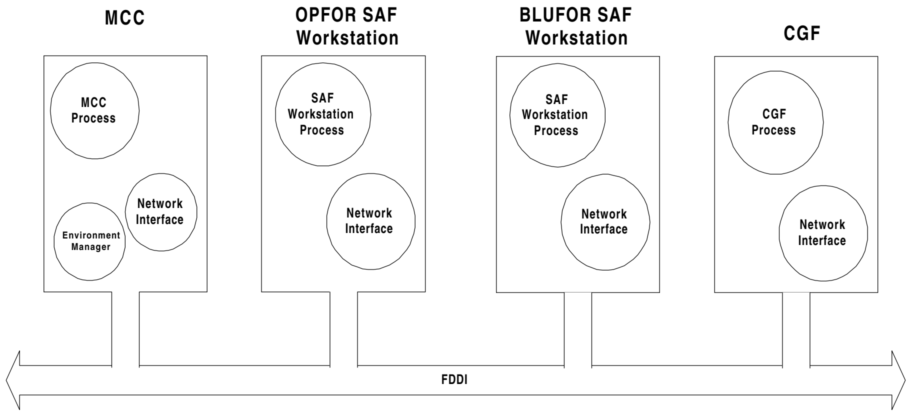

Figure 1: CCTT SAF Configuration

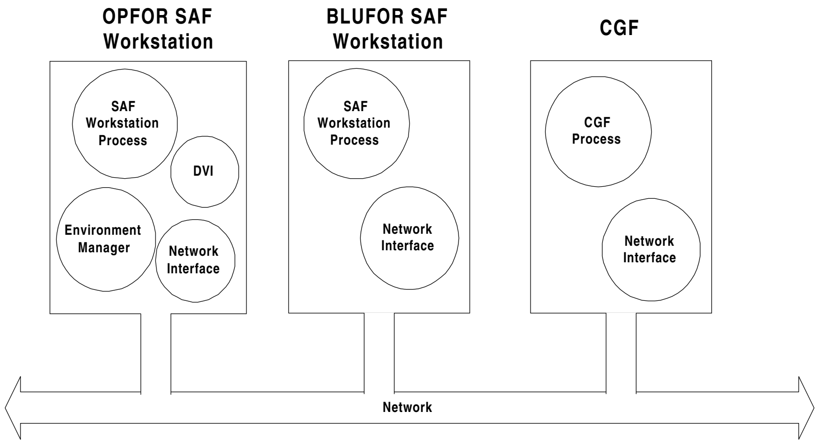

Figure 2: Standalone SAF Configuration

Since CCTT SAF was developed as part of the CCTT system, any use of CCTT SAF separate from the CCTT Training System required the Master Control Console (MCC) to start an exercise. This increased the resources needed with very little payback. This configuration is shown in Figure 1. However, there is nothing systemic to CCTT SAF that required an MCC. The real requirement for starting CCTT SAF is a series of simulation management Protocol Data Units (PDUs). SAIC developed new configurations and tools for CCTT SAF that replaced the MCC and provided a standalone environment (separate from the training site). This configuration, referred to as Standalone SAF, immediately increased the set of viable use cases for CCTT SAF.

Standalone CCTT SAF

Standalone CCTT SAF provides more flexibility for using CCTT SAF and thus increases its user base. These new configurations provide minimal resource requirements and helped scope the CCTT SAF architecture and configuration appropriate to the problem to be solved. Increasing the cost-effectiveness of CCTT SAF promotes research and analysis with the accredited equipment and behavioral models in CCTT.

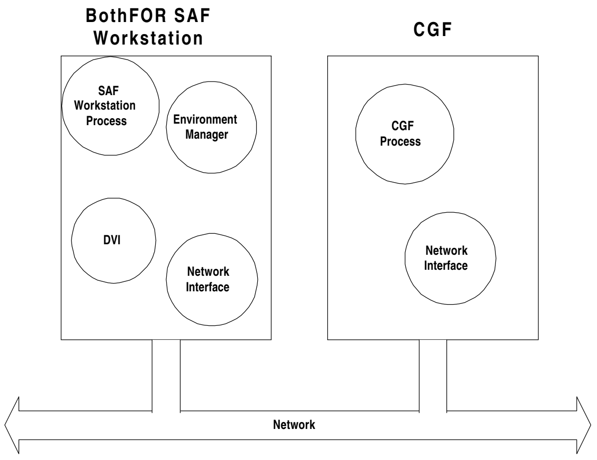

Figure 3: Standalone CCTT SAF Configuration with BothFOR

While Standalone CCTT SAF resulted in smaller configurations, they still support the concept of a "farm" or collection of CGFs and workstations sized appropriately for the types of exercises and scenarios desired. For example, a site that runs only Company level exercises may need one CGF while a site that wants to run multiple Battalions may use 5 to 6 CGFs. At each site, the user would choose how many workstations would comfortably control the types of exercises they desired.

Standalone SAF provides an expandable distributed set of CGF Simulators allowing the user to scope resources appropriately for the density of the exercises to be performed, but minimizes the computational resources required. Figure 2 shows a typical configuration of systems and processes. In this new configuration, the tools resident on the SAF Workstation assume the role of the MCC. Also, this configuration supports distribution across different types of network connections while the original CCTT SAF configuration only supported FDDI (Fiber Distributed Data Interface). The normal Standalone SAF configuration now can use Ethernet.

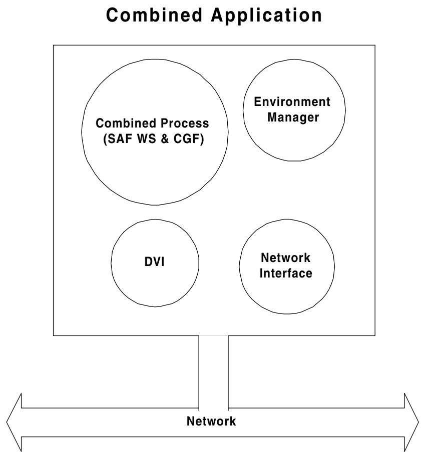

Figure 4: Combined Configuration

The original SAF Workstation (which was also part of the Standalone configuration) controlled only one side (OPFOR or BLUFOR) at a time. Because of this, two SAF Workstations were needed to support opposing and friendly forces. The CCTT SAF configuration requirements were then reduced further with the development of the SAF Suite. The SAF Suite combined the functionality of the OPFOR and BLUFOR SAF Workstations into a BothFOR SAF Workstation (Figure 3). BothFOR allows the SAF operator to control both forces on one workstation. This reduced the footprint of CCTT SAF again to two computers. Finally, the combined application was created which joined the capabilities of the BothFOR Workstation and CGF into one application (Figure 4). The footprint for CCTT SAF is now only one processor for small exercises.

These evolutions have developed two distinct configurations for Standalone CCTT SAF. The first is a distributed system with any number of SAF Workstations and CGFs. This configuration can be scaled by adding processors to support large exercises. The second configuration is the Combined configuration where small exercises can be executed on a single computer. These configurations facilitated the use of CCTT SAF structurally, but the cost of entry was still high using AIX and the PowerPC. What was needed was a configuration that minimized the cost of entry and took advantage of lower cost mainstream platforms.

PLATFORM CONTRASTS

The legacy CCTT SAF platform is almost as different as is possible from the target platform. Because of this vast difference, nearly every component that depended on the machine and operating system implementations had to be replaced. Table 1 illustrates some of the differences between the legacy (Power PC using AIX) and the target system (x86 running Linux).

Table 1: Platform Characteristic Contrasts

Catagory	Legacy	Target
Ada Compiler	OCSystems PowerAda	GNAT
C compiler	AIX C	GCC
Instruction set	PowerPC	IA32 (Intel x86)
Operating System	AIX	Linux
X libraries	X11R5	X11R6.4
OS bindings	OCSystems POSIX	Florist POSIX
UI bindings	SERC-derived AdaMotif	Custom built X/Motif
Endianness	big	little
Memory model	segmented	flat
Stack direction	up	down

The only real similarity between these two platforms is that they are Unix-based systems that provide standard Unix system services (e.g., shared memory, message queues) and an X/Motif user interface. The only problem not encountered was one of alignment. The IA32 model has less restrictive alignment requirements than the PowerPC model. This is fortunate because a more restrictive alignment requirement would mean extensive changes to the data model of some components.

Some of the differences listed above (e.g., Instruction set and stack direction) had minimal to no impact on the porting effort but are intended to illustrate the extent to which the platforms differ.

The difference in the memory model affected only some low-level debugging code that took advantage of the compiler and runtime library's use of the segments. This was easily worked around by not using that code in the flat model.

Because the AdaMotif bindings were really part of the legacy runtime and were incompatible with the GNAT compiler, a substitute had to be custom built for the porting effort as no suitable product currently exists.

The difference in X libraries was only an issue where older deprecated interfaces were being used that had been phased out in X11R6. While this took some effort, it was not a major hurdle and was one that needed to be addressed anyway for long-term maintenance of the baseline.

The OS bindings complied with the Ada POSIX standard and were thus not much of a problem. For the port, we used the free Florist bindings to replace the functionality provided by the legacy POSIX bindings. The POSIX bindings provided with the legacy compiler were incompatible with the GNAT compiler.

Differing semantics in the actual operating system however, were a problem. For example, various components relied on doing a `shmat()` system call to attach a shared memory segment to a file. This approach is now deprecated in more modern versions of Unix in favor of the more general `mmap()` system call which allows memory mapping of files.

The endianness difference was a big issue as it affected various components that expressed their data model in representation clauses in the code. This topic is discussed in more detail below.

Figure 5 illustrates the dependencies that the components of the SAF have on the platform.

MAJOR PORTING ISSUES

For the purposes of this paper, the major issues related to the porting effort of the CCTT SAF code can be divided into three categories: infrastructure, endianness of existing data, and the Graphical User Interface (GUI). The following sections discuss each of these in more detail.

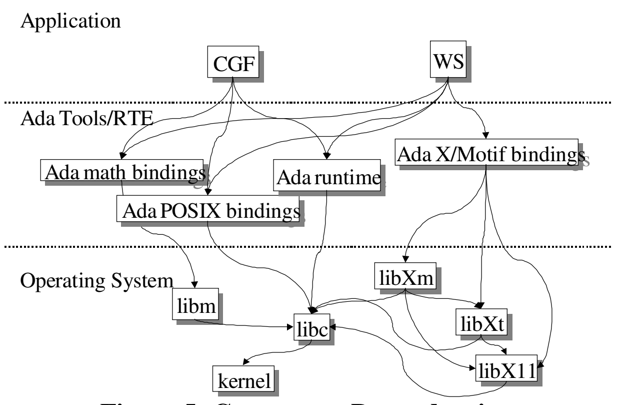

Figure 5: Component Dependencies

Infrastructure

Infrastructure issues are related to the difference in use or specification of the services, devices, and interfaces of the machine. This primarily includes the lower level software components that provide an interface to the machine resources. However, there are resource issues that are related to the GNAT compiler.

Most of the issues related to the system infrastructure are subtle implementation differences in the compiler and operating system services. Specifically, the infrastructure code for CCTT SAF is amazingly complex, incorporating many layers of Abstract Data Objects (ADOs), Abstract Data Types (ADTs), and Ada generics. One of the most insidious differences is the policy on the initialization of static memory. The AIX Ada compiler initialized static memory to zero while the GNAT compiler does not. One of the porting efforts has been to find the locations where lack of initialization affects the execution of the system. It is interesting to note that the style guidelines of the software development stated that the developer was responsible for initializing memory [2, Section 20.4.4.7]. Specifically, one would expect that a static variable would be initialized where the algorithm or data structure required it rather than relying on the compiler.

A specific example of an infrastructure issue is how the Planned View Display (PVD) mapped its terrain representation. The PVD is a process that displays an overhead map view of the battle area replete with overlay graphics and vehicle locations. The PVD map view is a faithful representation of a military map and is based on a rather large database on disk. To facilitate the strict performance requirements on the PVD, the code *memory maps* the database files. On AIX, this was accomplished by using the `shmat()` operating system call. However, Linux required the software to call `mmap()`, which allows for the memory mapping of files to shared memory. AIX allowed the `shmat()` routine to perform this operation while other systems do not.

Data Endianness

Data endianness issues are related to how information is retrieved from storage or communicated to other systems in a distribution. It would be useful to define endianness for this discussion.

> “**big-endian** *adj.* [common; From Swift's "Gulliver's Travels" via the famous paper "On Holy Wars and a Plea for Peace" by Danny Cohen, USC/ISI IEN 137, dated April 1, 1980] 1. Describes a computer architecture in which, within a given multi-byte numeric representation, the most significant byte has the lowest address (the word is stored `big-end-first'). Most processors, including the IBM 370 family, the PDP-10, the Motorola microprocessor families, and most of the various RISC designs are big-endian. Big-endian byte order is also sometimes called `network order'…” [3]

> **“little-endian** *adj.* Describes a computer architecture in which, within a given 16- or 32-bit word, bytes at lower addresses have lower significance (the word is stored `little-end-first'). The PDP-11 and VAX families of computers and Intel microprocessors and a lot of communications and networking hardware are little-endian.”[3]

CCTT relies on massive amounts of stored data. For example, one terrain database describes over 12 million trees and 30 thousand buildings. In addition, the characteristics of each vehicle component are described by data files. Each of these databases must be read and interpreted in real-time. Because of this, CCTT employs binary representations of data on disk. Using binary representations on disk allowed for pre-compilation of data from the source thereby saving processing time. The legacy system stored this information in big-endian format. The target Intel x86 hardware is little-endian. Thus, the Intel hardware cannot directly read the databases using the existing code.

Two methods used here to deal with endianness are native conversion and convert on communication. Native conversion is the conversion of data off-line to produce binary data files in the native format of the platform. This approach was used for the model and system characteristic data. This approach was chosen since the information was a memory image that was compiled from an ASCII representation. Using this approach, the binary files were replaced with re-compiled binary files from source data.

The second approach is the conversion during communication. This approach was used for the terrain data, PVD data, exercise data, and network communications. Implementing this approach proved to be more difficult since specific code needs to be generated for each attribute in each data model to be swapped. The following sections discuss each of these data files. The conversion during communication differs from the off-line conversion in that the data remains in its original state and is only converted as needed. For data files, this means that the software reads the original big-endian file and converts it to little-endian. For communication, the data may be stored internally as little-endian but converted to big-endian when communicated (and converted back to little-endian when received).

Terrain Data

The terrain data represents the definition of the virtual environment for CCTT SAF. A top-level description of the terrain data can be found in [4]. The terrain data model consists of many different individual record types representing data headers, data pages, data patches, elevation information, and features. Because of this, the data model is non-trivial. Also, the size of the databases are also non-trivial consuming up to 80 Megabytes of off-line storage. The target system reads the big-endian version of the file into its internal cache converting it during the read to a little-endian format. Since the compilation of the terrain data is difficult and the size of the data would make storage and management difficult, it is converted on use rather than off-line. Since this data is not read often, the effect on performance is negligible.

PVD Data

The PVD data represents the map view data of the virtual environment. It is similar to the terrain data in that it consists of large data files. The PVD has stringent performance requirements on how fast features and screens must be redrawn. When these requirements were contrasted to the density of the virtual world (millions of features), the decision was made to pre-compute many of the map features. So for this reason, the majority of map features including buildings, trees, contour lines, roads, and water are pre-computed into a large set of data files. For reasons similar to the terrain data solution, the PVD data is converted to little-endian when it is read and thus remains stored in the original big-endian format on disk.

Exercise Data

The exercise data is a series of files that store information about the vehicles, military organizations (platoons or companies), objects (wire, abates), orders, and environmental aspects of a particular scenario. These files express the types of military entities present in the scenario and the state of the virtual environment. This information is stored in an object database that was developed as part of the CCTT SAF development (rather than using off-the-shelf solutions). These objects are stored off-line as memory images of the objects in the object database. These binary images are stored as big-endian on the legacy system.

Exercise files are developed by the Army user to train specific tasks for student crews. As would be expected, the user has developed many exercise scenarios to teach and/or enforce coursework. There has been a large investment, by the user, in these files. Therefore, any successful port of the CCTT SAF system must be able to use these scenarios without modification. Because of this, the exercise files are read as big-endian and converted on read to little-endian before storage in the object database.

Network Communications

Since CCTT is a distributed simulation system, applications interoperate with other systems operating on nodes on the network. For CCTT SAF, there are two major protocols in use: DIS and CGF Protocol. The DIS protocol is an open standard network protocol developed by the modeling and simulation community to communicate information about things in virtual environments and characteristics of virtual environments. A more detailed description of the CCTT DIS dialect can be found in [5]. The CGF Protocol is a network protocol developed for CCTT SAF that maintains the consistency of CCTT SAF's shared object database across distributed computational systems[6]. The CGF Protocol is actually implemented as experimental Protocol Data Units (PDUs) under the DIS protocol. The DIS standard is to communicate on the network using the big-endian format (also known as network order). Because of this, the information slated for transmission across the network must be converted to big-endian format.

Since the network information is both read and written, the conversion for these data models must be supported both ways. For this reason, the software must convert these objects from little to big and from big to little-endian. The terrain data and PVD data do not require this two-way conversion since they are read-only sources of data.

Graphical User Interface

Graphical User Interface (GUI) issues are related to differences in the implementation of primarily X and Motif in the two systems.

X/Motif Ada bindings

The legacy CCTT system utilized proprietary Ada bindings to X/Motif that was part of the legacy compilation system. In order to de-couple CCTT from the PowerAda compiler, it was necessary to replace these bindings with open source bindings. This was done for the X11, Xt, and Xm layers. However, no open source bindings could be found for the Mrm (Motif Resource Manager) layer, so it was necessary to implement our own Ada bindings to Mrm.

Initialization Calls

The Graphical User Interface (GUI) applications in CCTT were designed and implemented under X11R3. In porting to Linux, the applications now run using X11R6. While the procedures used in the legacy initialization sequence were still available in X11R6, this sequence produced errors when run under the Linux implementation of X11R6. The preferred method for initializing an X/Motif application was changed from X11R3 to X11R6, therefore the X/Motif initialization sequence was updated for all of the GUI applications to the new sequence.Additionally, all calls to `MrmOpenHierarchy()` for UID files were replaced with `MrmOpenHierarchyPerDisplay()` , which is now the standard method for reading UID files.

Colors

Many of the calls to create pixmaps in the legacy code contained hard-coded color depths of 8. Because the CCTT system specified 256 colors for the displays of all workstations, this was not a problem. However, in order to support different display settings it was necessary to remove the hard-coded color values and replace them with the actual color depth of the screen obtained from a standard X11 function. Another problem that was found was that the standard call to `XDefaultColormap()` did not function properly under the Linux implementation of X11R6. While the call itself did not fail, all subsequent calls using the colormap would fail. To correct this problem, we found it necessary to obtain all colormaps from the top-level widget.

RESULTS

The results for this effort have been positive. We have successfully ported each of the applications that compose CCTT SAF at some level of minimum functionality. The major issues from this point on are stability issues as well as functional and capacity testing.

Developmental Results

One pleasing result is that we also ported the *process* of development used for CCTT SAF with quite positive results. CCTT is a large training system that commanded its own building. As would be expected, the development solution was "big-iron" as well. The typical development environment was distributed X-stations connected to centralized computational resources. The original CCTT development team consisted of over 100 engineers contrasted with about 15 support computers and 3 compilation computer systems. It was not unusual for up to 25 engineers to be using the computational resources at any given time. In developing the Linux port, the engineers are using de-centralized computational resources with each engineer capable of compilation on a singly-owned computer. Owing to the laws of processor improvements, the processors on each developer's desk for this effort are larger than the processors shared by 25 engineers on CCTT. This project used de-centralized, mostly free-ware tools. Each engineer then developed and ran the system on his/her own private computer.

Functional Results

We have also been pleased with the functional results. We have been able to port stable applications and are observing them interoperate and communicate across the distribution. Currently, we are performing system integration testing of these newly ported applications. Given the complexity of CCTT SAF and the vast range of functionality, we fully expect to see more issues associated with porting. One very pleasing result was the successful execution of CCTT SAF applications on a standard off-the-shelf laptop computer system. Historically, CCTT SAF has been viewed as tied to the PowerPC and the training site. Our efforts have proven our beliefs that CCTT SAF has the capability to be portable and accessible.

FUTURE WORK

Even though the system works, there are issues that should be addressed for longer term benefit. Issues of alignment of fields within the various data structures will have to be resolved in order to work on architectures with stricter alignment. Also, there are many places in the bindings that make gross assumptions, for convenience, about pointer sizes. While these two issues may seem esoteric, they must be resolved in order for the system to work on any of the available 64-bit platforms in a 64-bit clean mode (e.g., Sun UltraSPARC, DEC Alpha, Intel Itanium).

The components of the system that are implemented as distinct processes (network, entity database, cgf application) communicate through shared memory and message queues. It would be preferable to re-architect these components such that they can be instantiated as POSIX compliant Ada tasks within the same program. This would improve performance, manageability, and the ability to isolate instantiations on a system (referred to as being in a "sandbox") in the standalone case.

There are some places in the code that awkwardly deal with capabilities in the only way possible in Ada83. Examples include the definition of bitwise types, calls through function pointers, and interfaces to non-Ada code. It would be advantageous to port these capabilities to Ada95 to take advantage of greater portability, performance, and maintainability.

CONCLUSIONS

In this paper, we have presented the major issues encountered while converting a complex system. CCTT SAF is a complex and large modeling and simulation system that was developed and fielded under a centralized computational system. We have been successful with our effort in porting to Linux and a de-centralized development methodology. Most importantly, there are many benefits that can be realized from a successful port to a IA32/Linux platform.

The benefits provided by porting CCTT to a PC platform will be felt by both the system user and developer. Developers will be able to work more efficiently, and, therefore, more quickly. Baseline CCTT requires a set of workstations linked to a server for development. A separate set of processors is required for testing. Developers must access the server to write and compile the code. The developer must then wait until the resources required to run the test are available. The executables generated from the code change must then be copied to the test machine and run. After the test is completed, the test logs and any other necessary documentation must be generated. The developer must then take the new information back to his desk and start the process again. Under Linux, the developer could use the same machine to develop and test. Since he would be the only user of the machine, the wait time would be eliminated. This may also encourage developers to test more often since the resources are more readily available.

CCTT original use case was for deployment on a large scale similar to the Ft. Hood facility. This facility houses 10 SAF Workstations, 15 CGFs, and more then 30 manned modules. The soldiers being trained were those present in the manned modules. The SAF Operators were facilitators to the training of soldier vehicle crews. After the Linux port, CCTT SAF is more accessible for new use cases. It can be distributed without the manned modules and other system processors which minimizes the cost of the system. Also, since the system will run on PCs, the hardware required usually exists on site and the software is already owned by the user thus the cost to install is minimal. Due to the low cost, CCTT can be used in classroom settings to train higher echelon leaders. The role of the SAF Operator is similar to that of the higher echelon leaders. They can issue orders to their subordinates and observe the outcome thus providing them with a real life training perspective.

Another benefit of porting CCTT SAF to Linux is the increased possibilities for alternative form factors. CCTT SAF now can be more easily ported to other forms since Linux is supported by a variety of forms from net appliances to large scale systems.

ACKNOWLEDGMENTS

This work was done under SAIC internal research funding.

REFERENCES

[1] APM, Close Combat Tactical Trainer, http://www.stricom.army.mil/STRICOM/PM-CATT/APM-CCTT/

[2] Close Combat Tactical Trainer Software Development Plan – Revision B, 28 March 1995, Contract N61339-93-C-004, CDRL A002, Report Number 94-CCTT-LFS-00166

[3] The Jargon Dictionary [netmeg.net], http://www.netmeg.net/jargon/

[4] Watkins, J., Provost, M. "Design of Terrain Reasoning Database for CCTT", *Proc. Of the Fifth Conference on AI, Simulation, and Planning in High-Autonomy Systems*, December 1994

[5] CCTT Interoperability Description Document, http://www.stricom.army.mil/STRICOM/PM-CATT/APM-CCTT/CCTT/IOP/

[6] Shen, D., "CGF Protocol Support the CCTT SAF", *14th Workshop on Standards for the Interoperability of Distributed Simulations*, March 1996