OSSIE-based GRA Testbed

Eric Redding
Rockwell Collins, Inc.
Richardson, TX

Tom Rittenbach
U.S. Army CERDEC
Ft. Monmouth, NJ

Hiroshi Satake
SAIC
San Francisco, CA

Dr. Carl Dietrich
Virginia Tech
Blacksburg, VA

ABSTRACT

This purpose of this paper is to document an architectural validation approach for the Government Reference Architecture (GRA) for SATCOM terminals. The approach integrates the Virginia Tech OSSIE SCA Core Framework with a UML model of the GRA, coupled with low-fidelity models of the SATCOM components. This approach uses the Rhapsody tool to implement the GRA-based testbed from documentation through executable code generation. The end result is a system that demonstrates the portability of the GRA while simultaneously validating the GRA against a representative system. Furthermore, the system demonstrates the reduced development time required for architecture extensions through the incorporation of existing projects and automated code generation.

INTRODUCTION

This paper describes an inexpensive, open source testbed for developing and testing GRA-compliant software or hardware modules. This GRA testbed promotes a modular, building block approach with the advantages of upgradeability with new technology over the life cycle and reusability of software or hardware designs across product lines.

To leverage existing, open source components, the GRA reference design testbed consists of an OSSIE Core Framework augmented by a GRA infrastructure and waveform applications on a Linux platform as shown in Figure 1. The GRA infrastructure consists of replaceable CIL Modules (common Devices and Terminal Control Services) each with GRA specified interfaces and operations. In the GRA Testbed, any CIL Module can be replaced by an actual Device, Terminal Control Service, or Waveform Application to test for conformance with GRA specified APIs through a set of regression test cases.

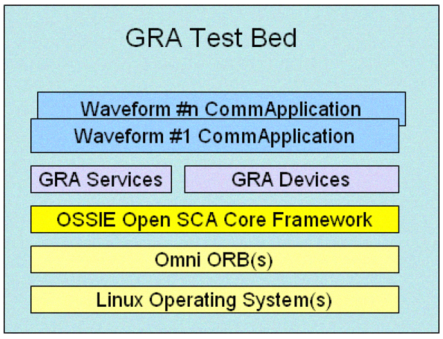

Figure 1. GRA Reference Design Testbed

GRA BACKGROUND

Modern SATCOM terminals are composed of several terminal variants to satisfy the comm-on-the-move (COTM), comm-at-the-halt (CATH), transit case, and man-packable terminal requirements. These variants support a wide range of waveforms that include CDL, XDR, XDR+, NCW, and 165A. Most of these “Above 2 GHz” waveforms have been developed and re-developed several times by various vendors on various terminal programs, always at great expense to the government. In the past, the practice of continuous and redundant development and redevelopment with each succeeding generation of SATCOM terminals was accepted as the unavoidable cost of progress. As the "Above 2 GHz" waveforms become increasingly complex, sophisticated and expensive to develop (often requiring a decade of intense development) it has become increasingly impractical to recreate the hardware and software that comprises these waveforms with each platform generation, particularly when a single platform must support not one, but an entire family of waveforms. With the advent of software radios, government software standards have been proposed, and are evolving, which are intended to facilitate reuse and portability. To date, these steps have achieved limited effectiveness in maximizing reuse and reducing cost, particularly in "Above 2 GHz" terminal designs. The fundamental problem that all new terminal programs face is the lack of an efficient methodology to leverage and/or reuse pre-existing efforts, without incurring substantial redevelopment costs particularly but not limited to

waveform development. Given the problem space described above, the Army is investigating a truly modular concept which will reduce terminal costs by facilitating a common Open Systems Interface software layer with an associated cooperative hardware Architecture / strategy for control, data, and bit oriented traffic. The desire is to provide the capability to compete special purpose modules through the mandated use of an open middleware layer, thus providing industry the ability to utilize and protect its IR&D intellectual property in the development of these modules. This will provide industry with the incentive to invest IR&D resources in optimized module designs because there will be a competitive business case to recoup and profit from IR&D efforts without having to give up the rights to proprietary, intellectual property. This paradigm delineates a Government Domain open communications middleware layer to which all modules would interface.

OPEN SOURCE METHODS

Open source is often defined as an approach to design, development, and distribution offering practical accessibility to a product's source (goods and knowledge). An open source operation differs from the closed, centralized models of development model and decision process by allowing simultaneous input of different priorities, agendas, and approaches. Open innovation is an emerging concept which advocates putting R&D in a common pool. The Eclipse platform is openly presenting itself as an Open innovation network. Linux, GNU Radio, OSSIE, and the USRP are all open source projects.

GRA PROCESS

The GRA currently consists of three levels using terms defined by OMG, the Computation Independent Model (CIM), the Platform Independent Model (PIM), and the Platform Specific Model (PSM). The main efforts of the GRA to date have centered on the first two levels, using the CIM to define use cases distributed over black box modules and then the PIM to lay out the entire set of potential subsystem interfaces expected in a SATCOM system along with some utility and infrastructure functions. The PSM is expected to take the PIM and realize it into an operational system.

The open source methodology presented in this paper adds another layer to the basic GRA architecture, an integrated testbed layer called the GRA OSSIE USRP PIM, or "GROUP". The GROUP will allow open source exercise/demonstration of the entire GRA process from CIM to functional hardware. It should be noted that testbeds to support rapid prototype and architecture validation fall in between the concept of PIM and PSM. They make use of stub functionality that is well beyond the generic PIM, but is not a deployment to an operational system and therefore technically not a PSM. So the testbed gets referred to as a validation PSM.

The CIM focuses on the environment of the system and the requirements for the system, but the details of the structure and processing of the system are hidden or undetermined. Likewise, a CIM does not show details of the structure of systems.

The PIM is an architecture of a software system that supports the requirements defined by the CIM. The PIM is independent of specific technological platforms that can be used to implement it, thus limiting it to abstract interface definitions for the most part, although some functionalities specified in the CIM will be generic enough to be realized in the PIM. At a minimum, the GRA PIM defines the structure of the CIL Modules.

The hybrid model used to create this testbed brings in low-fidelity stub code to give implementation to the abstract interfaces defined in the PIM for validation purposes. One might call that a PSM, but the validation is not intended to be a playtform specific validation, so no clear definition exists for it. The key point is that it was desirable to separate module interfaces from implementation details for validation, but the validation effort needed to remain with the PIM side for distribution and as a reference model example. Note that the validation is done at only at the module boundary / interface level (i.e. only the abstract interfaces of the PIM are exposed to other module developers / subcontractors). The implementation for validation purposes has to meet the same interfaces as any operational implementation

In general terms, a GRA PSM would be a fully realized GRA PIM, complete with implementations for the intermodule infrastructure as well as implementations internal to the modules themselves. The PSM architecture binds platform specific technology layers to the PIM. Examples of these specific technology layers include operating system, technology platform (CORBA, .Net, J2EE, or PHP), databases, remote distribution vs. local collocation, and programming language. For the testbed purposes, low-fidelity models of the functionality stand in for full blown implementations for all of the auxiliary functionality and while the main modules are developed with functional software centered on the OSSIE framework implementation and the USRP modem device drivers.

GRA TESTBED METHODOLGY

The GRA Reference Design Services, Devices and Waveform Application modules are developed and generated in a Windows station using the Rhapsody tool (Graphical SysML/UML development and simulation environment). Over the past year, 5 industry partners refined the definition of the GRA standard interfaces and operations, developed a Platform Independent Model (PIM) for each CIL Module, developed 14 Validation Test Cases, and developed representative implementations of the PIM using executable state charts, activity diagrams (flow charts) or stub implementation code. The testbed is compiled natively for Windows for software-only versions or cross-compiled to the OSSIE Linux target for hardware in the loop versions. As shown in Figure 2, the cross-generated source code targeted for the Linux profile can be loaded into the GRA Testbed either manually or deployed from the Rhapsody platform into the LINUX based Testbed.

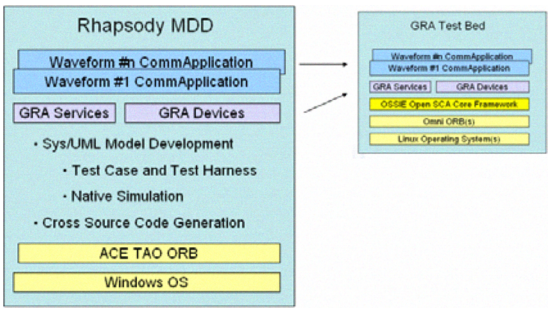

Figure 2. Rhapsody Model Driven Development Station

If a Testbed user has a Rhapsody license, the User can also use Rhapsody to develop and test replacement Devices, Services, and Waveform Applications. Without a Rhapsody license, waveforms can be developed natively on the Linux Testbed using the OSSIE Waveform Development (OWD) tool.

Execution in the GRA Testbed can be monitored either through the OSSIE ALF on the Linux Testbed or logging events can be sniffed on the (IP network) by the Rhapsody Station.

Figure 3 shows a possible deployment of GRA Waveforms, Devices and Services to the GRA Testbed. Some components may be deployed to Single Board Computers (SBC) such as a Commercial Off-The-Shelf (COTS) programmable modem. Some devices like the INFOSEC, Terrestrial Wide-area Network (WLAN), Satellite Wireless LAN (WLAN or Network Interface Module NIM) may be deployed as daughterboards or separate modules with their own programmable processor and address spaces. For testbed purposes, another SBC is added to connect a test harness for data insertion and extraction.

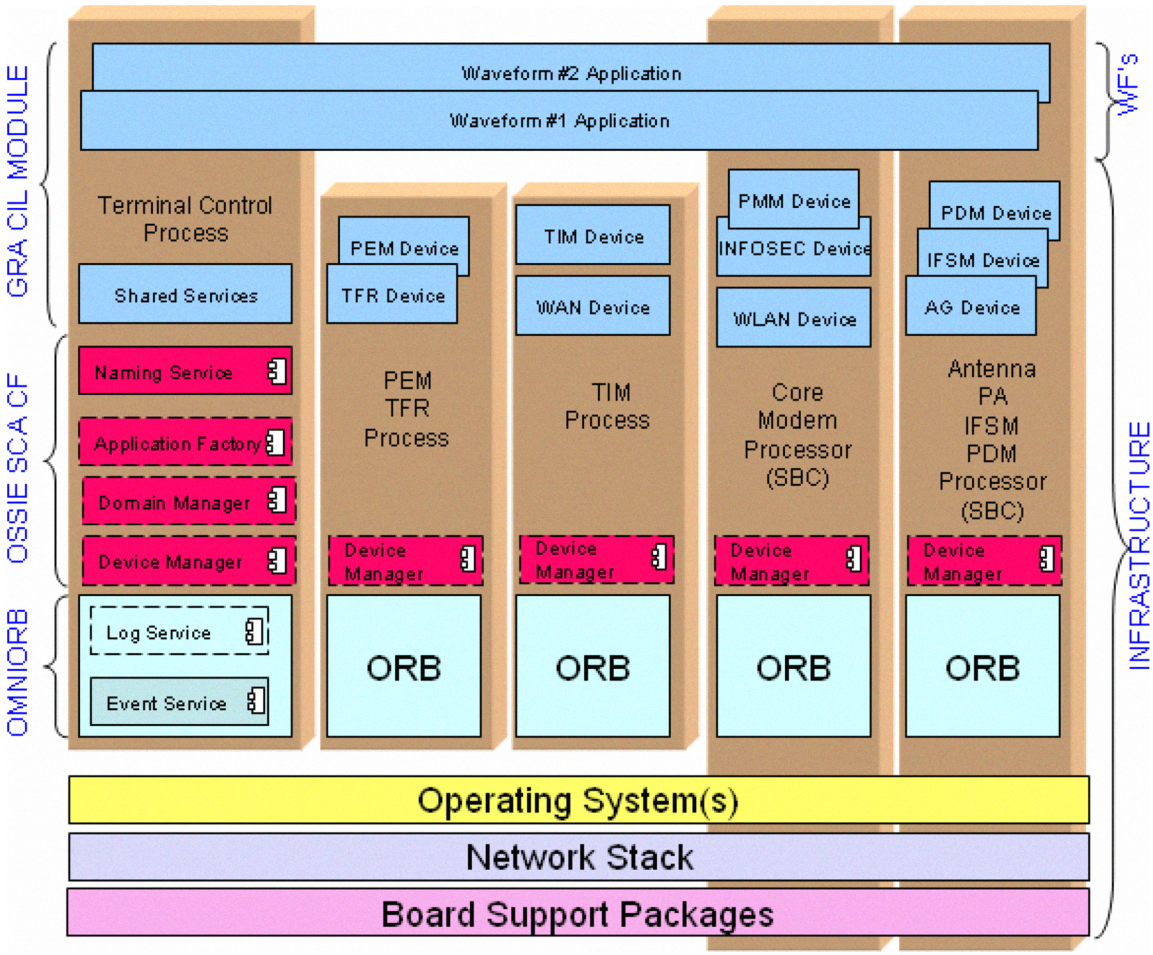

Figure 3. Example GRA Deployment to Testbed and SBCs

RHAPSODY INTEGRATION: GRA SYSTEM LEVEL INTERFACES

The GRA makes extensive use of the existing standards in communications and software development. The architecture of the terminal system utilizes the UML and SysML standards from OMG along with associated process concepts such as CIM and PIM discussed previously to create a reference model generalized for many different SATCOM or other radio terminals. The key point of the GRA PIM is that the system level interfaces are standardized such that upgrades to any particular subsystem module can go through the acquisition process independently. The implementation of the infrastructure centers on SCA to maximize reuse of existing infrastructures freeing the GRA to focus on subsystem interfaces specific to SATCOM terminals.

GRA REFERENCE MODEL

The GRA reference model is captured in the Rhapsody modeling tool from IBM. This tool enables model–driven design, capturing all aspects of the system from high-level use cases and associated activity diagrams to subsystem interface definition and associated UML ports to stub implementation for validation of the architecture. Furthermore, details of the system are captured within the tool through notes at each level and are used to compile an overall documentation set.

RHAPSODY STANDARD DATA INTERFACE CONSTRUCTS

The Rhapsody tool incorporates IDL and C++ code generation, allowing increased productivity through a heterogeneous combination of hand coding of basic operation implementation coupled with automated coding of UML diagrams, IDL interfacing, and class structure. For GRA purposes, this means that a user can create standard data interface constructs, tagged as IDL, and then have the tool realize those interface constructs into the final implementation language (C++ for this effort) to which operation implementations can be attached. More specifically, the productivity improvements come through the tool automating the implementations to match the name-mangling that the CORBA IDL compiler

uses, while being able to draw UML class diagrams with mixed IDL/C++ for illustration of the class hierarchy.

ROLE OF UML AND CORBA

The ultimate productivity improvement comes via the integration of the SCA port concept with the UML 2.0 port concept. With this critical link, we are able to construct UML structure diagrams that detail the intermodule interfaces using UML ports. The UML connector bridging these interfaces is actually implemented in CORBA, but the UML provides a standardized way to diagram CORBA interfaces. The SCA port comes in to play in that each CORBA interface is built as a client / server pair. The client side C++ concrete class derives from the SCA port IDL abstract interface with Rhapsody automating the realization. The server side then realizes the GRA-defined terminal-specific interfaces, again via a C++ concrete class derived from the terminal-specific abstract IDL interfaces, with Rhapsody automating that realization as well.

PIM, PSM, AND ARCHITECTURE VALIDATION

It should be noted that the GRA PIM as defined by the reference model is strictly IDL. The GRA makes no requirements as to how the PSM is implemented. For the purposes of the architecture validation, we have elected to use the previously described technique since the productivity improvements allowed us to quickly assemble an executable implementation of the reference model for validation purposes only. However, the technique of masking the SCA port with a UML port is appropriate for any operational software as well. In other words, the GRA infrastructure coupled with an SCA infrastructure works with either a low-fidelity model of the module or with an operational version of the model.

MODEL FIDELITY AND UNIT INTEGRATION

This aspect of the GRA brings the benefits of modular systems to fruition. From the perspective of a module developer, the GRA presents a set of standardized interfaces to be met. No development effort is complete without exercising the software against an operational system, but attempting big-bang integration, where all of the operational pieces are thrown together simultaneously, is costly in terms of debugging. But simple interface stubs are not usually possible in complex communications software systems because of the interdependencies between modules - you may be able to test interfaces individually but you usually cannot test the implementation of a use case without having at least low-fidelity models of the other modules in place. The architecture of the GRA and its clear, standardized CORBA-based interface points, along with supporting systems design, make it possible to execute the system with a combination of low-fidelity models, higher-fidelity models, or operational software. The CORBA middleware is also key to this modularity in that models can be run on general purpose computers alongside operational software being run on proprietary hardware. Therefore, the benefit to the module developer is that a set of low-fidelity models of every module is already available as part of the reference model since they were created to support validation. The module developer can utilize these models to do a more detailed unit integration test of their individual module before attempting to integrate their module with other operational modules. Thus the GRA helps capitalize the efficiency goals of every SATCOM terminal production program.

GRA AND THE SCA COUPLING

To this end, the OSSIE Core Framework becomes invaluable as a stable, open-source SCA Core Framework implementation available for use in a testbed. The GRA architecture focuses on removing coupling between the GRA and the SCA implementations. Again, the goal is efficiency in module development, and having a stable testbed in which a module can be exercised is a huge step in efficiency. Using a stable, open-source Core Framework on the testbed during development reduces debugging overhead when a proprietary Core Framework implementation is in concurrent development. Having the GRA and SCA CF decoupled then allows for quick substitution of the operational CF for the open source CF once the module is proven to work in the low-fidelity testbed.

Furthermore, Rhapsody has a simple interface by which proprietary libraries and associated header files can be used. The application programming interfaces of the library can be incorporated into the Rhapsody model either directly or through reverse engineering of existing code, while the implementations can remain in a compiled library. This allows other pieces of the model to easily utilize the functionality of the library without resorting to hand-coded source files. Third-party waveform developers can provide the API's to a module developer along with the corresponding libraries to incorporate into the makefiles and the module-level code that controls that waveform. Incorporating these API's into Rhapsody provides a less error-prone utilization by presenting documented API's side by side with the implementations of the users of the API's.

GRA: MODULE DEVELOPERS SPRINGBOARD

A second focus of the GRA reference model is to springboard module developers at the onset of a program. In this respect, the GRA is much more than a standardization of the inter-module interfaces. Each of the interfaces specified by the GRA has supporting systems design diagrams detailing how the interfaces were expected to be used. Of course the IDL leaves open the possibility of other implementations or future upgrades, but the idea behind the GRA is that a detailed system design that includes UML and SysML diagrams such as activity diagrams and state charts, in addition to the class diagrams / block definition diagrams and structure diagrams / internal block diagrams will allow module developers to have a good starting point from which they can begin to design the internals of the subsystem modules. One of the primary slowdowns in communications software development when using standardized interfaces is that the expected flow of control of the software is not well understood and must be ascertained by careful study of the verbage surrounding the interface. By utilizing precise diagrams such as activity diagrams, the environment in which an operation is expected to be utilized is immensely easier to understand. But, as stated before, the module developer is free to modify the flow during the course of design and implementation. Only the interface must be adhered to in order to produce modular software.

PLATFORM SPECIFIC LINKING

The Rhapsody tool includes various capabilities to allow the integration of the systems and software architectures and designs. The tool allows for establishing associations between SysML blocks and UML classes to provide traceability. Likewise, it has the capability to provide traceability to a DOORS database for requirements. For GRA purposes, the Rhapsody Harmony process was loosely followed, so the CIM is focused on use cases while the PIM is focused on physical allocation, but there are many annotation points in the CIM that link to points in the PIM, establishing traceability even without direct inheritance. Platform specific modeling of the subsystem modules can be linked more tightly than the intermodule interfacing since the subsystem design is more focused on specific functionalities that are implemented in software rather than the functional decomposition of the system through use cases as seen in the CIM.

OSSIE INTEGRATION

The proof of concept GRA demonstration uses the Open Source SCA Implementation - Embedded (OSSIE). The OSSIE project is an initiative by Wireless @ Virginia Tech, an interdisciplinary research group, to provide a software defined radio (SDR) platform that is simple, easy to expand, and open-source for the development of waveforms based on the Software Communications Architecture (SCA) specifications under the Joint Tactical Radio System (JTRS) program as well as work by the Object Management Group (OMG). OSSIE illustrates essential aspects of service oriented SDR architectures (Domain and Device Managers, Resources, Devices, Factories, Profiles, etc.). OSSIE has been used for research, education, and rapid prototyping by engineers from organizations that include large and small businesses, nonprofit research institutions, and government laboratories, as well as universities.

OSSIE PROJECT INITIATIVES:

The OSSIE project includes four major initiatives:

1. An SDR core framework based on the SCA

The OSSIE core framework is written in C++ and uses the openly available omniORB CORBA ORB. Current development is primarily focused on the Linux operating system. As of April 2009, the released version of OSSIE runs on the general purpose processor of most PCs using a recent version of Linux like Fedora Core 9. A release that includes enhanced support for embedded processors is planned for fall 2009. Experimental embedded versions have been ported to the several platforms including ARM and PowerPC processors.

2. The Waveform Workshop

The waveform workshop is a collection of software tools for rapid development, visualization and debugging, and interactive control of SDR components and waveform applications, as well as node-specific profiles that specify devices to be managed by a Device Manager.

The tools are written in Python and Java, and include a plug-in for the Eclipse open source integrated development environment. Figure 1 illustrates the use of the OSSIE tools to interactively run and control an SDR waveform application.

3. An Application Library

This evolving library includes pre-built components and waveforms for demonstration and for use in laboratory exercises.

Components can be written in either C++ for rapid execution or in Python for rapid prototyping of less computationally intensive components.

4. Free laboratory/tutorial exercises

These exercises have been developed in cooperation with the Naval Postgraduate School and are suitable both for use in courses and for self-paced independent study. They introduce new developers to the use of the tools to develop new components and waveform applications, and provide a basis for more advanced work. The applications include broadcast receivers and will soon include a simple digital radio. Two of the laboratory/tutorial exercises illustrate OSSIE's capability to support applications distributed over multiple processors, and this capability is exploited in the GRA demonstration. Screen capture videos of some of the exercises are available on the OSSIE web site.

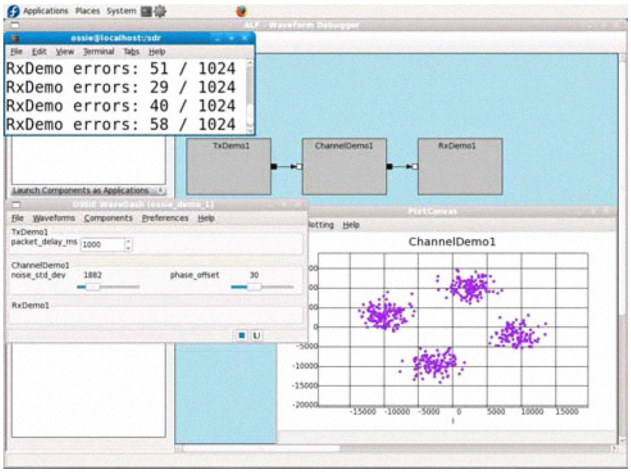

Figure 4. OSSIE

A simple OSSIE application can be launched and displayed using the ALF waveform visualization and debugging environment, and controlled using the Waveform Dashboard (WaveDash), which provides an interactively customizable user interface for prototyping waveforms.

USE OF OSSIE IN THE GRA DEMONSTRATION

Multiple nodes with the OSSIE Domain Manager on one node and one OSSIE Device Manager per node are supported. The GRA is used to start the Domain Manager and Device Managers and to install and run waveform applications. The testbed utilizes the GRA infrastructure for the control plane of the terminal. OSSIE-developed devices comprise the data plane and react to command, control, and configuration commands from the GRA applications and device drivers received via the SCA Core Framework operations realized by the OSSIE devices. Hardware in the loop is incorporated by tying in a USRP board with OSSIE-defined devices to the GRA-defined programmable modem implementation which is itself an SCA CF device but interfaces to the remainder of the modem via the SCA CF composite device interface.

As stated before, the purpose of GRA is not to prescribe the exact implementation of a terminal, but rather provide a reference model, framework, and stable modular platform upon which waveforms may be deployed. By coupling this architecture with a representative PSM for validation purposes, a testbed is created that can be used for unit integration test during development of GRA-compliant terminals. The purpose of bringing OSSIE in to this testbed is the productivity improvement of having a functional, platform-specific implementation to test against while developing implementations for the various subsystems of the GRA. The reduced cost and increased visibility of open source software is the driving force in selecting the OSSIE tool to do the data plane implementation.

FUTURE DIRECTION: PROPOSED OPEN SOURCE METHOD APPLIED TO GRA

The OSSIE SCA GRA-based Testbed physically demonstrates end to end the CIM-PIM-PSM process. It has provides a working example of concise system engineering process that goes far beyond the current generation of "style guide" standards which add significant cost, but questionable value. Wide spread adoption is necessary for such a standard to produce maximum benefits. As such it is an ideal candidate for open source development. Linux, GNU Radio, OSSIE, and the USRP are all open source projects. Taking the GRA to the next level and making an open source GRA Terminal Operating Environment is a natural follow on. Because the GRA was created in support of military radios, any defacto functionality must be made as reliable as possible so that certification can be assured. But creating a fully implemented operating environment, complete with operating system, would allow the develops of communications applications to build those applications without regard to most of the hardware modules, focusing instead only on the modem or other proprietary implementations required. Security management and general terminal management could be standardized into a government-provided default terminal operating environment. Open source is the only way to proceed on project like that. In order to build higher user

confidence, certification is mandatory. Certification can be applied to the simple components used by developers to build the simple modules or to an entire software system. Obviously, the entire military radio cannot be made entirely open source without compromising its purpose, to provide assured communicationss. Portions of it dealing directly with information security must treated separately, but the GRA architecture allows for Closed Source modules (Black Box) that have proprietary waveform software and hardware. Only the common bus (CIL) and its utility structures need to be made open source. One feature of open source is its inherent security. Because many eyes guide its development and constantly review and test its components, it becomes unlikely that a "trojan horse" or "worm" can be built into it unnoticed. These are but some of the issues that need to be considered.

CONCLUSION

The GRA represents a new paradigm for military standards based terminal development. From its goals of reuse of existing standards like SCA coupled with modular, open subsystem architecture, terminal systems may be procured at the subsystem level and still be able to integrate with one another. Utilizing UML-based reference models and SysML-based detailed system design to present the required interfaces allows a much greater understanding of the development needed. Utilizing tools like IBM Rhapsody to accomplish the detailed design leads to productivity improvement and more accurate scheduling of efforts. Further productivity improvement is realized through the creation of a testbed that not only is used to validate the GRA architecture but is extended into the hardware-in-the-loop operational testing by incorporating the OSSIE waveform development tool and its various capabilities. By keeping the reference models, system design, and the testbed components in an open source environment as much as possible leads to better competition for operational systems procurement as well as to reduced cost in the execution of those programs.

REFERENCES

[1] OMG CIM/PIM/PSMs
[2] OSSIE User's Guide
[3] GRA RFP Material