**Abstract**— The purpose of the Government Reference Architecture (GRA) was to define a modular, open systems architecture that fostered reuse and technology insertion refresh with modular components and product line variants. A graphical model-driven development (MDD) approach was chosen to support rapid design and development of work products that could be quickly leveraged by radio developers within existing development processes with commercial modeling tools. The GRA has matured its model driven methodology through recent efforts transforming the OSSIE-based GRA testbed from an executable UML™ Platform Independent Model (PIM) to an executable Software Communications Architecture (SCA) Platform Specific Model (PSM). In GRA Phase 2, IBM® Rational® Rhapsody® was used to develop the UML PIM model for the purpose of GRA interface validation. The third step in GRA Phase 3 was to demonstrate an executable PSM implementation by combining the PSM work in Rhapsody with additional tool capability for the SCA CORBA® model using the SCA domain specific MDD tool, PrismTech Spectra CX. Although gaps were identified in the path to an integrated, standards-based, automated GRA modeling tool chain, productivity gains around improved collaboration and code development accelerations were realized. Evolving the GRA through model-driven PIM and PSM development tools, including the tool chain integration process, will be described in detail in this paper, along with analysis of MDD productivity gains achieved with this integrated tool chain approach.

**INTRODUCTION**

The purpose of this paper is to describe the model driven development process used to achieve a Software Defined Radio (SDR) while meeting the requirements of software reuse and technology refresh. The process is a graphical approach meeting the requirements of Object Management Group (OMG) SysML™ and UML™, and Software Communications Architecture (SCA) and Government Reference Architecture (GRA). The graphical Model Driven Design (MDD) approach realizes the promise of modular software technology evolution that started with Object Oriented Programming (OOP), CORBA® middleware and SCA Core Framework towards achieving deterministic software development cost and schedule, software reliability, and fielded SDR security.

The GRA evolved in 3 phases and involved the contribution of up to 6 industry partners in the “Above 2 GHz” defense industry:

1) GRA1 was primarily aimed at a text standard like the SCA and defined Standard Modules and Standard Interfaces [1].
2) GRA2 adopted an OMG SysML and UML language to bridge the gap between system and software engineering [2] and moved towards an executable OMG Platform Independent Model (PIM). [3]
3) GRA3 is creating a GRA Reference Design testbed suitable for developing GRA and SCA compliant SATCOM and Cognitive Radios using an OMG Platform Specific Model (PSM) approach.

The graphical MDD design chain used in this GRA effort is shown in Figure 1. The purpose of sharing the GRA evolution process is not to endorse the vendor tools used - IBM® Rhapsody®, Rhapsody®, PrismTech Spectra CX tools, and the open source Virginia Tech OSSIE tools - but to show that an MDD approach exists that offers productivity enhancements towards the GRA goals. This paper will describe the role and advantages of each tool and the changes that were required to make them function in an automated work flow to achieve the

![Figure 1. A GRA MDD Tool Chain](image-url)
maximum productivity benefits of rapid design, better collaboration, and automated code generation with improved quality of deliverables.

**MDD TOOLS IN DEFENSE INDUSTRY**

The IBM Rational Rhapsody SysML and UML-based tool is commonly used to document modular, use-case-driven System Engineering architecture in the Department of Defense (DoD) Preliminary Design Review (PDR) and Critical Design Review (CDR) phases. However, after exiting CDR, most DoD programs drop straight into text-based, monolithic implementation code, followed by big-bang integration. The Rhapsody Harmony process supports the iterative, Evolutionary and Agile software development process being considered for DoD software acquisition [4]. The model driven design was extended to include an SCA domain specific tool, PrismTech Spectra CX. We sought to achieve the success of Agilent Technologies in developing an SCA compliant radio platform [5].

JTRS SCA Next is considering an “SCA Model” as the common CORBA neutral abstraction tailorable to meet platform specific requirements [6]. The ultimate goal of MDD is to improve productivity, so regardless of which suite of tools is used, productivity generally increases. Rhapsody has been found to be a valuable tool for providing a common graphical language for communicating requirements and architecture to software implementation. However, Rhapsody is insufficient to work in SCA domain specific environments. Therefore, MDD was extended to include an SCA domain specific tool, PrismTech Spectra CX. The GRA consortium determined that the two-tool solution achieved the optimal productivity improvements.

**QUALITY IMPROVEMENTS WITH MDD: THE SYSTEM TO SOFTWARE ENGINEERING INTERFACE**

The systems engineering to software engineering hand-off is one of the most common defect injection points in the entire software development process. One of the goals of model MDD is to use a variety of means to alleviate some or all of these defects. Since there is no specific point defined as the hand-off point, the GRA consortium elected to allocate the definition of the PIM interfaces to be that point. Moreover, the hand-off was not just the throw it over the fence type, but rather the interfaces had to be agreed upon by both the software and systems engineering groups, with the systems engineering validating the interfaces from the inter-module communication while software engineering approached the validation from the needs of the implementation. The initial effort of the GRA consortium was to validate these interfaces using a rudimentary implementation within the IBM Rational Rhapsody framework. Since this implementation was not geared towards any real-world example, the next step for the GRA consortium was to transform the initial effort into a PSM implementation that could leverage the SCA via the OSSIE Core Framework and waveform implementations.

We quickly observed that despite months of work on the PIM interfaces, once we began working with actual SCA implementations, there were several issues requiring rework of the reference architecture and the interfaces it defined. Once we got the code straightened out, we moved on to the SCA XML descriptors. We had three alternatives for the XML descriptors: 1) brute force hand coding, 2) using the Rhapsody Java™ Application Programming Interface (API) to write some Java code to extract information from the project and generate the XML descriptors, or 3) utilize a commercial-off-the-shelf (COTS) tool, such as PrismTech Spectra CX, to generate the XML. Although it might seem to be a simple choice to pick an already-tested COTS MDD tool, we learned in our ensuing trade study that tool chain integration without reasonable knowledge of the tools at hand and where they best intersect and in what order can be time consuming in itself. With our initial problems generating the XML by tying the tools together and some OSSIE platform limitations that were uncovered in the process, our approach was comparable to the level of effort with the Rhapsody Java API implementation.

The Java API had the advantage of immediate access to all of the information already captured in Rhapsody, including the awkward generalization of an SCA port to a UML port, plus all of the links were already specified. All of the COTS tools we evaluated required XMI to import the UML notation, and the XMI standard has yet to smoothly support UML migrations. The XMI importers can not pull in all required information to another UML tool like PrismTech Spectra CX, meaning a great deal of information, ports / interfaces, and connections, must be entered manually if you originate the data in Rhapsody. Our initial tooling struggles help explain why many experienced developers have yet to fully embrace a productivity tool for end-to-end development, let alone rely on a chain of tools, preferring instead to hand code most or all of an implementation themselves.

But in our determination to accelerate and expand on the advantages of model-driven development by leveraging MDD component-based tools, we continued working with PrismTech Spectra CX. We discovered that importing from the Spectra SCA environment to the less specific Rhapsody tools is possible as an SCA port can be generalized into a UML port, while the reverse is not true. Understanding the limitations of the XMI import process allows one to rethink the optimal design process and tool chain order. For instance, you can establish the system engineering use-cases and activity diagrams/sequence of events in Rhapsody then export only high-level blocks into a component-based implementation tool to lay out the actual software architecture. This architecture would be composed of SCA-based interfaces that will successfully import into the more generalized UML of Rhapsody.

The lesson learned is that there are a multitude of ways to use UML and component-based tools together – some far more productive than others. For the PSM realization of the GRA-OSSIE testbed and given our tight delivery schedule, we brute forced our way through by hand-entering the data already in
Rhapsody into PrismTech Spectra CX. Given a little time and our improved knowledge of tool capabilities, a more elegant solution could be achieved.

**MDD METHODOLOGY OPTIONS: UML-BASED VERSUS COMPONENT-BASED MODELING**

The use of UML-based and component-based modeling tools in the defense industry has increased in recent years. Both methodologies produce modular software, but component implementations are generally completely self-contained and meant to perform a particular service without modification. In this respect, it is truly considered reuse software, whereas modular software may only make use of generic software libraries that, in the end, may benefit productivity very little due to shifting APIs and the time required to sift through available libraries to find and verify relevant functionality. With GRA goals to promote reuse, there is ample reason to go with a component-based SCA-aware implementation tool.

Another reason for selecting a component-based SCA tool for implementation has to do with the shortcomings of the SCA when using UML. For instance, the lack of a service class definition in SCA as opposed to a type definition creates havoc in UML because it does not follow the normal approach (Figure 2). A large portion of the functionality in the reference architecture was decomposed into services to be offered by a platform, but realizing the GRA extensions to the base service concept required a traditional class-based approach. In other words, dealing with Services as structure types instead of class types makes UML-based extensions to the SCA concept much more difficult. GRA addresses this by simply defining a Service class that exists in parallel with the CF::DeviceManager::ServiceType typedef, but since the SCA-aware tools take this into account for the developer, implementing in an SCA-aware tool is definitely more productive. The next issue, as mentioned before, is the non-commutative mapping of SCA ports to UML (2.0) ports. This issue can be overcome for the most part by simply understanding the limitations, but it still plays havoc in the XMI import process. Updates in the tools to allow tweaking the import would help. However none of the issues are showstoppers. Since UML is predominantly used in an architectural process and SCA is predominantly used in an implementation process that builds on that architecture, the use of multiple tools in a MDD approach to waveform design is logical. And, as with services, SCA-aware tools take this into account when they generate the UML or when they export XMI to another tool just to view the UML. In this instance, the UML can be exported from the SCA-aware tool and shown in the non-SCA-aware tool, but the reverse is not true in the current set of tools. This is due to use of XML properties attached to the UML defining its SCA roles, and these properties are not understood by the non-SCA-aware tools.

**MDD GENERIC TOOL OPTION: RHAPSODY AND SYSML/UML TOOL CHAIN SPECIFICS**

Rational Rhapsody provides a visual design environment to create requirements and model systems using the UML™ diagrams and SysML™ diagrams.

To realize the Computation Independent Model (CIM), PIM, and PSM system designs, we used the IBM Rational Rhapsody® software. Rational Rhapsody provided us the system architecture and design support as well as automatic source-code generation. The PIM capability of Rational Rhapsody, which supports language-independent and operating system–independent modeling, allowed us to develop the design before hardware was available, and enabled us to validate functional behavior.

Subject matter experts developed the CIM, which are SysML models of use-case requirements. Software engineers developed the executable PIM implementation, an executable UML model. The CIM focuses on the environment of the system and the requirements for the system, with the details of the structure and processing of the system hidden or undetermined. The user of the CIM, the domain practitioner (systems engineer), uses the block oriented SysML palette and is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM.

Rhapsody’s automated verification and validation capability generated the state machine from our design that enabled us to verify correct operation. In the example below, Figure 3, the CallControl state machine identifies the state-based behavior of instances of CallControl when the system receives call requests from users and connects calls.

```
From SCA's IDL
interface Device : Resource { ... }
interface DeviceManager : PropertySet,
    PortSupplier {
        struct ServiceType { ... } }
From GRA's IDL
interface I_Device : CF::Device { ... }
interface I_Service : CF::Resource { ... }
```

Figure 2.IDL Snippets for Services

Figure 3: Rhapsody State Machine
To test, collaborate and demonstrate the model’s behavior, we extensively used the Rhapsody Webify feature to control and manage the model by remotely invoking events, calling operations, and viewing the changing data. This capability allowed us to automatically generate and host a Web site that served as a graphic interface to access the components of the model through a built-in Web server.

The PIM is a model of a software system that is independent of specific technological platforms that can be used to implement it. The PIM is a realization of the requirements specified in the CIM, with only UML artifacts present in the PIM. Using the Rhapsody process flow shown in Figure 4 for the GRA2, hardware and software engineers developed the PIM (described below) using SysML and UML models of interfaces captured in interfaces spreadsheets.

PSM architecture binds a platform-specific technology layer to the PIM. This specific technology includes operating systems, databases, programming languages, etc. The PSM realizes the PIM as it applies to a specific platform. The module (service or device) is decomposed into one or more sub-module levels as appropriate. It is the first level of decomposition of GRA Common Integration Layer (CIL) functions. Classes have state charts to model the behavior of the module. State charts inherit the basic device state chart of the PIM with sub-states added as necessary for specific behavior.

**MDD TOOLS WITH DOMAIN SPECIFIC SUPPORT: PRISMTECH SPECTRA CX AND CORBA TOOL CHAIN SPECIFICS**

To overcome these SCA problems with the UML ports in Rhapsody, the PrismTech Spectra CX tool was chosen to model natively with SCA CORBA ports. The SCA model is built using standard UML diagrams (primarily Component Diagrams) and elements (primarily Ports and Connectors). However each Spectra port behaves in accordance with the CORBA standard and connections are constrained by the SCA standard.

Modeling with SCA elements graphically is productive and fun. Figure 5 shows some of the elements in the PrismTech Spectra CX SCA modeling palette, including Components, Applications, AssemblyControllers, Nodes, DeviceManagers, DomainManager, PortTypes, Ports, Operations, Parameters, and Attributes. PrismTech Spectra CX provides wizards to organize projects, packages, components, models, and diagrams; connect Ports using Connectors graphically; generate or cross-generate Interface Description Language (IDL) and XML descriptor and Code files; and build and run the model natively.

PrismTech Spectra CX, like its Zeligsoft Component Enabler (CE) and PrismTech Spectra predecessors, is able to generate the SCA XML descriptor files from the graphical model, that proved to be very valuable. In addition the PrismTech Spectra CX model can be validated against SCA rules at any time during the modeling process.

Like IBM Rhapsody, PrismTech Spectra CX has a steep learning curve even for developers familiar with the SCA standard who may not be aware of the legacy of some informal concepts. For example, PrismTech Spectra CX represents each SCA component as a triad of classes (Interface, Realization, Impl) as shown in Figure 6.
On the other hand, the PrismTech Spectra CX tool interpretation of ambiguous and informal SCA concepts helped in defining the deployment of GRA components in the SCA “node”-based “platform” architecture, “boot node” startup, and replacing the static ports and monolithic connections in the GRA2 with dynamic “findby” ports.

Some of the learning curve time associated with PrismTech Spectra CX will be addressed with the next release where some early usability difficulties are corrected and new capability is delivered. The version of PrismTech Spectra CX 3.1 used in GRA3 modeling is a first release of the now joint product efforts of PrismTech and Zeligsoft to deliver a best-of-breed SDR SCA development tool suite.

**MDD WITH TARGET PLATFORM TESTBED TOOLS: OSSIE WAVEFORM WORKSHOP AND SCA CORE FRAMEWORK TOOL CHAIN SPECIFICS**

The OSSIE Waveform Workshop is a collection of graphical user interface (GUI)-based tools for rapid development and run time configuration and debugging of OSSIE signal processing components and waveform applications [7].

The OSSIE Eclipse Feature (OEF) provides a GUI-based/drag-and-drop interface for developing OSSIE signal processing components and waveform applications within Eclipse. OEF allows launching the nodeBooter (which launches Domain and Device managers), and other OSSIE Waveform Workshop tools from a menu.

Developers can use OEF to create C++ or Python programming language components that run with the OSSIE Core Framework. A developer specifies ports and interfaces used by each port, and then specifies reconfigurable properties, their data types, and their default values. OEF then autogenerates skeletal source code and XML files. The developer adds appropriate signal processing code.

OEF also allows developers to rapidly create new waveform applications. The developer uses a drag-and-drop interface to select components for inclusion in the application, connect components, and optionally to override default component property values. This information is stored in a software assembly descriptor (SAD) XML file, mandated by the SCA. OEF also lets a developer specify hardware to be managed by one or more device managers, and specify deployment of components to devices. This information, called the device assignment sequence (DAS), is stored in a second XML file.

Other tools in the Waveform Workshop include the Waveform Dashboard (WaveDash) and a waveform visualization and debugging environment called “ALF”. WaveDash generates a customizable GUI for any OSSIE waveform application by using information from the CORBA naming service and waveform and component XML profiles. ALF allows developers to display block diagrams of waveform applications, inject and monitor signals at component input and output ports, and launch individual components as self-contained waveform applications.

**MDD TOOL CHAIN INTEGRATION: GRA TOOL CHAIN GAPS**

In the GRA Modeling Tool Chain, two gaps were identified that required manual workarounds: 1) on the front end interface to Rhapsody, there is a gap in UML Modeling Interchange and 2) on the back end interface to OSSIE, there is a gap in the build process.

Currently PrismTech Spectra CX uses IBM Rational Software Modeler (RSM) and its proprietary EMX format for graphical UML modeling. IBM Rhapsody, having been bought recently from Telelogic (and before that from iLogix), uses a proprietary RPY/SBS format. While IBM Rhapsody can export UML models in XMI format and IBM RSM can import XMI formatted models, and OMG XMI Interchange Compatibility testing showed a high degree of compatibility, the export-import mechanism failed with the GRA PIM model and the interchange compatibility is known to be a major problem with the OMG XMI standard.

The workaround was to redraw and re-enter each of the 26 Rhapsody GRA Component Diagrams in RSM. During this process, each UML Class in a Rhapsody component was documented in the User Guide to the OSSIE 0.7.4 Domain Manager.

The second gap between Spectra and OSSIE is due primarily to known incompatibilities between the PrismTech Spectra CX’s SCA BuildEnvironment for linuxOS-x86 processor-omniORB-OSSIE CF and OSSIE 0.7.4 due to CF SCA implementation issues. Zeligsoft has a list of recommended changes needed for an earlier version of OSSIE CF to correct these areas. Also, there was limited time available to get a working GRA testbed and a lack of training and time to work with PrismTech Spectra CX. The workaround was to leave executable code generation to the Rhapsody tool and manually make the patches documented in the User Guide to the OSSIE 0.7.4 Domain Manager.

Eventually the Zeligsoft and GRA changes to the OSSIE CF will be reconciled with the current OSSIE 0.8.0 code base. With the GRA testbed running with the current manual workarounds and with sufficient time and training code generation using the PrismTech Spectra CX omniORB-OSSIE CF BuildEnvironment should become automatic.

The other induced limitation is XML generation from Spectra being merged with the existing XML for OSSIE with the potential clash of IDs and the lack of integrated cross-references. In hindsight, the existing XML in OSSIE should have been imported into Spectra prior to automated XML descriptor generation. For the sake of time, the merge process, taking snippets from Spectra-generated XML and plugging into OSSIE XML to support additional resources launched under OSSIE, was simply done manually.
MDD TOOL CHAIN PRODUCTIVITY OUTLOOK

One may try and choose to implement MDD using a single tool such as Rhapsody or PrismTech Spectra CX and it would be a difficult decision. Rhapsody is an excellent tool for laying out general embedded software architectures using UML notation. It is a good general tool for any type of embedded software, but it lacks any specialization to SCA-based waveform implementations. The PrismTech Spectra CX is a specialized tool for component-based SCA waveform and platform implementation including SCA descriptor and component code generation, but it lacks native support for SysML. Rhapsody’s default implementations can have rather large memory footprints because of their generalized nature; most component-based waveform implementation tools like PrismTech Spectra CX generate very streamlined code.

An important integration aspect of both Rhapsody and PrismTech Spectra CX that we leveraged only minimally is their mutual support for the Eclipse Integrated Development Environment (IDE). Eclipse is a mature industry standard IDE for tooling platforms and provides many integration points for tool capabilities supported within its pluggable framework, including shared projects, components, libraries, source control management, and build environments. Both Rhapsody and Spectra are implemented as Eclipse plug-ins, meaning both tools are offered as capabilities within a single Eclipse workspace and can operate within the same Eclipse projects. The IDE provides a transition platform between tool capabilities, allowing the user to leverage best-of-breed tools for different tasks throughout the development cycle.

Eclipse also provides a single workspace with drag-and-drop capability between tools, but UML model “sharing” is still limited by XMI immaturity. For the GRA efforts, we would like to see XMI support improved to establish a process where the PIM interfaces would be created as the final artifact in the systems engineering model, and the XMI import process was modified such that PrismTech Spectra CX could import the ports and connections and generate the CORBA code and translators as needed. This enhanced XMI approach seems very promising but is not quite there. In the mean time, we will utilize our experience with our chosen tools and how best to “chain” them together by leveraging the shared Eclipse framework and exercising import capabilities in the currently supportable directions to utilize PIM layout and XML generation in Spectra, then pass back via Eclipse projects to Rhapsody to do final implementation.

CONCLUSION

This paper describes the automated MDD process flow with tool support that can be used to develop SDRs and that was used to develop the GRA3 Platform Specific Model. Many DoD programs use Rhapsody for use-case driven design through PDR and CDR as we did in the GRA2 phase and achieve productivity benefits in the design and collaboration phases of development. GRA2 demonstrated that Rhapsody SysML for system engineers and UML for software engineers are effective graphical languages for developing and handling over requirements, architecture and interfaces from CIM to PIM. However, after CDR, when software engineers proceed to coding a distributed SCA CORBA-based implementation, a UML-based modeling tool such as Rhapsody when used alone hinders productivity.

In GRA3, we selected a second, component-based, SCA CORBA-aware modeling tool, PrismTech Spectra CX. In this paper, we described using these COTS tools with the open source OSSIE CF and Waveform Workshop and our efforts to achieve maximum MDD productivity gains with a tool chain process integration. As discussed in the paper, we identified integration gaps between Rhapsody and PrismTech Spectra CX and between PrismTech Spectra CX and OSSIE including using workarounds to bridge some of the gaps. We believe this MDD process with tool chains appropriate for full cycle and domain specific development returned productivity benefits and we recommend COTS tool-supported MDD for any GRA or other radio architecture implementation efforts.

REFERENCES